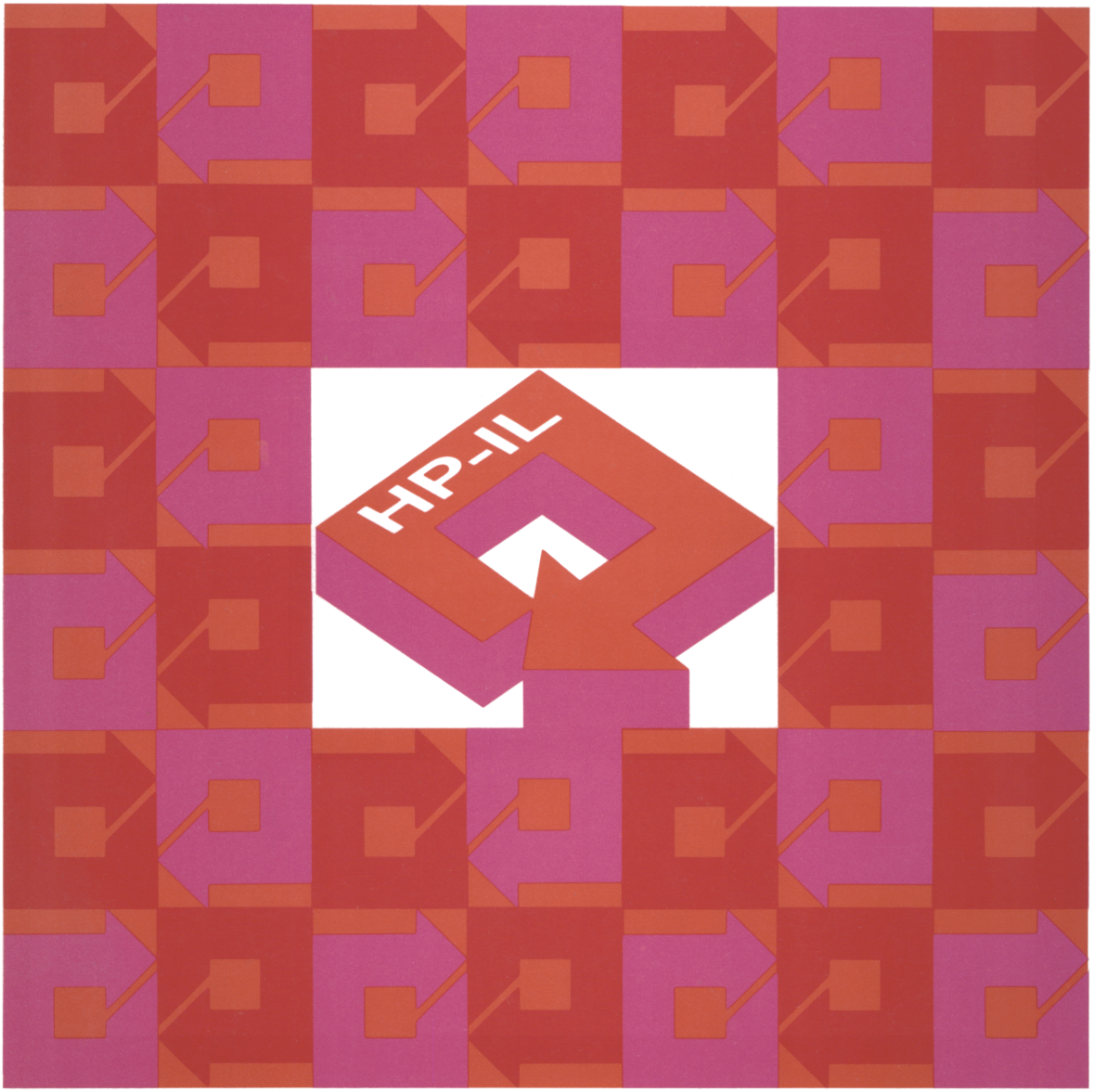
OSBORNE/McGraw-Hill

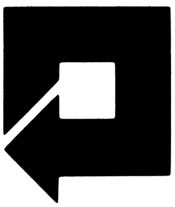
Gerry Kane
Steve Harper
David Ushijima

THE HP-IL SYSTEM: An Introductory Guide to the Hewlett-Packard Interface Loop



THE HP-IL SYSTEM

THE HP-IL SYSTEM: An Introductory Guide to the Hewlett-Packard Interface Loop



Gerry Kane
Steve Harper
David Ushijima

OSBORNE/McGraw-Hill
Berkeley, California

Published by
OSBORNE/McGraw-Hill
630 Bancroft Way
Berkeley, California 94710
U.S.A.

For information on translations and book distributors outside of the U.S.A., please write OSBORNE/McGraw-Hill at the above address.

THE HP-IL SYSTEM

Copyright © 1982 by McGraw-Hill, Inc. All rights reserved. Printed in the United States of America. Except as permitted under the Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a data base or retrieval system, without the prior written permission of the publisher.

1234567890 HCHC 898765432

ISBN 0-931988-77-2

Cover design by Lorilee Willis

Text design by K.L.T. van Genderen

DISCLAIMER

The state diagrams that appear in this book are subject to change. To guarantee complete accuracy you should obtain a copy of the HP-IL reference specification.

CONTENTS

CHAPTER 1

An Introduction To HP-IL

HP-IL (short for Hewlett-Packard Interface Loop) is an interface system. It defines the way a wide range of stand-alone, desk-top, and handheld devices can work together as a unified system.

The characteristics of HP-IL are formally defined in a document known as the HP-IL reference specification. This specification defines both a physical link and a message protocol between a number of devices and controllers. Before we go any further, perhaps it would help to place some of these terms in perspective.

THE NEED FOR COMMUNICATION

Communication is a very vital concept in computing systems. Most systems are comprised of a number of different devices. Each device is typically capable of executing a specific task. If these devices are to work together in an orderly fashion, they must have the ability to communicate with each other. To do this, they must be physically connected and, equally important, they must speak a common language.

THE PHYSICAL CONNECTION

A hardware interface allows a device to be connected to other devices via a physical link. The nature of this link varies from device to device (see Figure 1-1).

A parallel interface is shown in Figure 1-2. In this case, the parallel interface consists of eight data lines and a set of control lines. Information is transferred from Device A to Device B in parallel, eight bits at a time over the eight data lines. The control lines are used to synchronize the

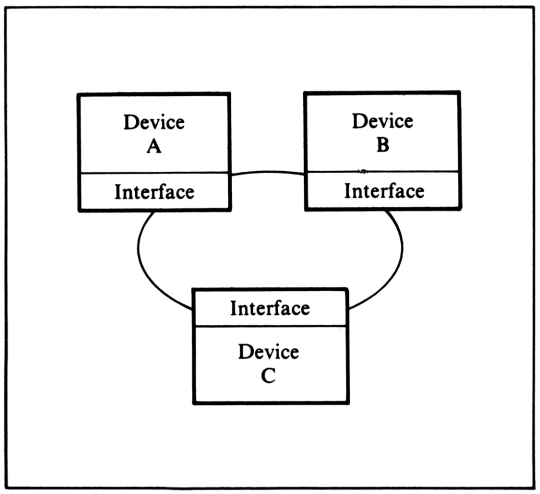


FIGURE 1-1. Three devices linked through a common interface system

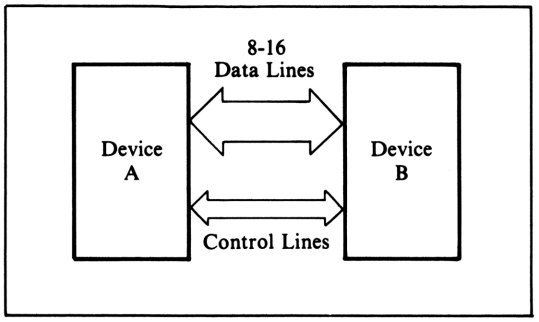


FIGURE 1-2. A parallel interface

data transfers between devices, a process often known as "handshaking." Typically, Device A will signal Device B once it has placed valid data on the data lines. Once the data has been accepted, Device B will acknowledge receipt of the data by asserting the appropriate control line.

A serial interface consisting of a single data line and a reference line is shown in Figure 1-3. Information is transferred from Device A to Device B one bit at a time via the single data line. You can easily see that a serial interface limits the rate at which data can be transferred. However, it is simpler and, for most low- to medium-speed applications, the data rate of a serial interface is more than adequate.

Configurations

Once the general nature of the physical link is determined to be parallel or serial, there are a number of ways to configure an interface system. One of the most common hardware configurations is the bus-type system shown in Figure 1-4. Parallel bus-type interfaces are usually used in cases where speed is a critical consideration. Information is transmitted along the bus and each device residing on the bus has equal access to all bus information simultaneously (or as simultaneously as is physically possible). Usually the limiting factor in bus systems is the length of the transmission medium. Longer distances tend to degrade the quality of the signal and thus introduce errors.

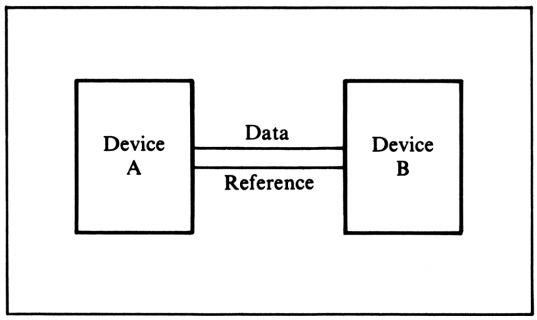


FIGURE 1-3. A serial interface

Another interface configuration, commonly called a "ring" or "loop" interface is shown in Figure 1-5. Information is sent over the loop and passes from device to device, finally returning to its source. For example, suppose Device A in Figure 1-5 sends out information intended for Device D. This information travels over the loop, passing through Devices B and C, finally to arrive at Device D, where the information is stored in Device D's local storage. The information, meanwhile, continues to traverse the loop until it ends up back at the source, in this case Device A. At this point, Device A could compare this newly received information with that which it sent to verify and check for errors.

Obviously, the limiting factor in a loop or ring configuration is the time it takes information to travel completely around the loop. However, with the present communication technology, adequate transmission rates can be achieved.

Two primary benefits of a serial-loop configuration are low cost and low power consumption. Since there are only two wires in the connecting link, there is also a much lower chance of a hardware failure, compared to the bus configuration of Figure 1-4.

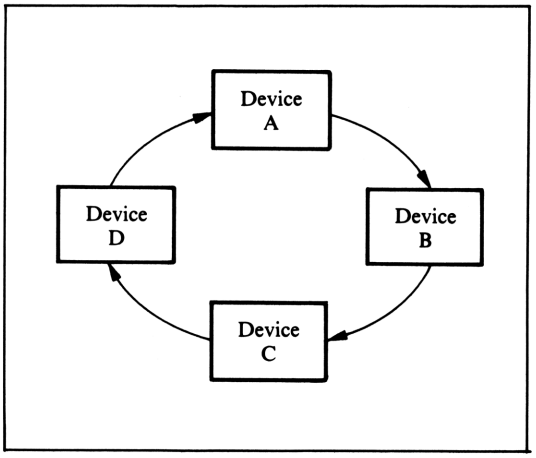


FIGURE 1-5. A ring or loop interface

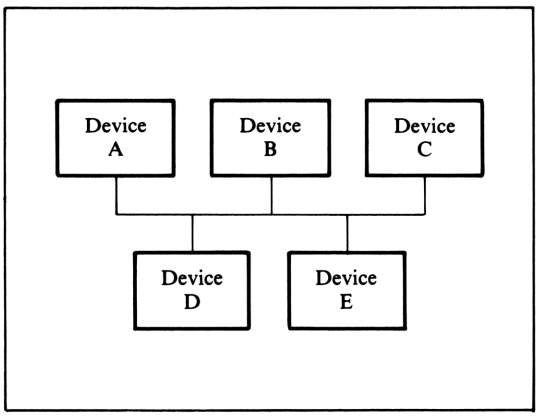


FIGURE 1-4. A bus interface system

PROTOCOLS AND COMPATIBILITY

More than just a common physical link is needed to support communications between devices. In order for devices to understand each other, they must send information out in a form that is understood by all devices. They must also all agree to support a common set of rules governing the way communications will occur over the physical link. This is particularly true in the case of a serial interface where there are no control lines to support the handshaking functions. The common set of rules and the format of individual

messages are often defined in what is known as a communications protocol.

Various manufacturers tend to utilize their own protocols for their own equipment. In most cases, the protocols used by different manufacturers are not compatible. For instance, information sent using an IBM protocol such as SDLC, may not be compatible with information sent from a piece of DEC equipment and vice-versa.

To overcome this problem, interface standards have evolved over the years. This allows manufacturers of communications and processing equipment to independently manufacture their own equipment, yet maintain some semblance of compatibility. One such standard is the IEEE-488 also known as the GPIB (short for General-Purpose Interface Bus).

FROM HP-IB TO HP-IL

The IEEE 488 was originally developed at Hewlett-Packard (under the name HP-IB) for use with its own line of computers and measurement devices. It is a parallel-type bus interface that allows all devices on the bus to communicate with each other under the direction of one or more Controllers. In 1975, the Standards Board of the Institute for Electrical and Electronic Engineers (IEEE) elected to adopt HP-IB as a standard. The IEEE 488 currently supports more than 1000 pieces of equipment manufactured by companies around the world.

The IEEE 488 was originally intended to support a wide range of equipment, from the very fast to the very slow. Since its design, technology has spawned the growth of medium-speed, portable equipment. Devices such as the handheld computer and its associated peripherals are now the trend. With this shift toward medium performance, low-cost, and lightweight equipment, new standards are being developed to reflect and support the new trends in technology. One such development is the Interface Loop developed by the Hewlett-Packard Corporation.

While not intended to replace the IEEE 488 Interface, HP-IL is a natural outgrowth of the trend toward smaller, portable, and low-cost equipment. It is primarily aimed at the small-system user, whether in the laboratory, office, or home.

AN OVERVIEW OF HP-IL

An easy way to describe an HP-IL system is to simply look at a typical configuration. Figure 1-6 shows an HP-IL system comprised of a Controller and two other devices: a printer and a magnetic tape drive.

Since HP-IL is a serial communications interface, information travels one bit at a time over the two-wire cable connecting devices and Controllers. All devices communicate by sending messages over this two-wire loop. Each message is comprised of 11 bits as shown in Figure 1-7.

To ensure that devices and Controllers speak the same language, Hewlett-Packard has defined a message structure; that is, a common set of messages that all HP-IL devices must understand. These messages define the way a device must operate if it

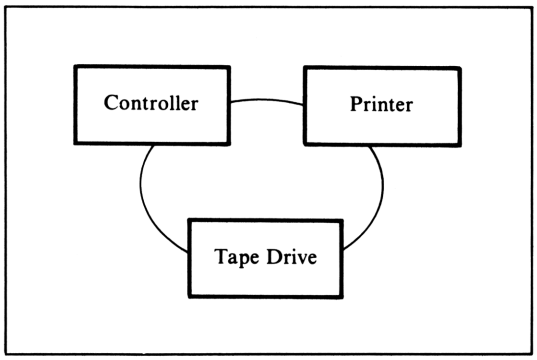


FIGURE 1-6. A simple HP-IL system

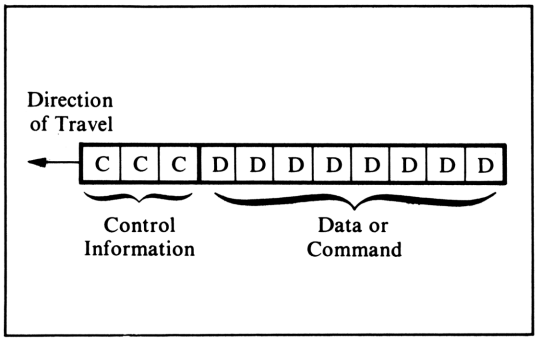


FIGURE 1-7. Structure of an HP-IL message

wishes to send information out over the loop.

Messages originating at a Controller or device circulate around the loop to each device in turn, finally arriving back at the source. This is illustrated in Figure 1-8.

HP-IL systems can be comprised of three types of devices: Talkers, Listeners, and Controllers. Talkers are devices that send data over the interface loop. Listeners are devices that receive data from a Talker or commands from a Controller. The role Talker or Listener is assigned to a device by a Controller.

Controllers are in charge of all loop operations. They are typically responsible for assigning addresses to devices, assigning device roles, servicing device requests, and initiating the transfer of data from Talker to Listener(s).

BASIC SYSTEM FEATURES

Now that you know a little about what HP-IL is, here is an explanation of some of its capabilities.

The Maximum Number Of Devices

HP-IL will support a maximum of 31 devices on a single loop using the standard method of addressing (a topic which will be covered later in the book). An extended form of addressing (involving the issuance

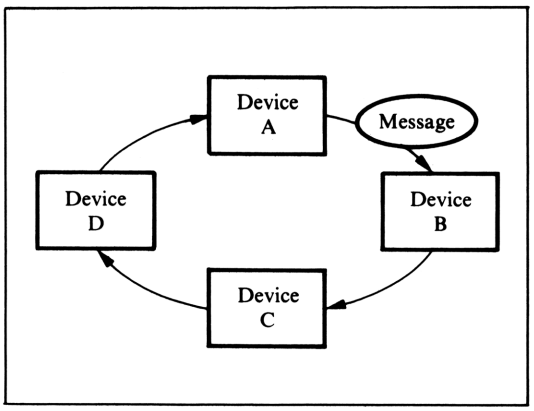


FIGURE 1-8. An HP-IL message on the Interface Loop

of a two-byte address) allows the HP-IL interface to support up to 960 devices.

Data Transfer Rates

The rate at which data may be transferred over the loop interface is theoretically limited to 20K (20,000) bytes per second. Using equipment currently available (at the time of this printing), speeds of about 2K bytes per second are achievable. This translates to about half of a page of text (on an 8½" × 11" piece of paper) per second.

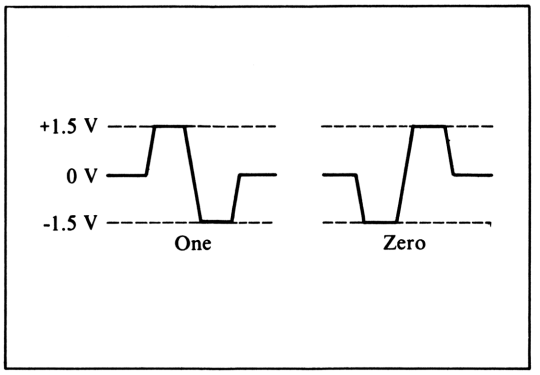


FIGURE 1-9. HP-IL transmission code

Mode Of Transmission

HP-IL transmissions are implemented in hardware using the three-level code shown in Figure 1-9. A logic one is represented by a high pulse of 1.5 volts followed by a low pulse of -1.5 volts. A logic zero is represented by a pulse of -1.5 volts followed by a high pulse of +1.5 volts. A level of 0 volts is used to represent the "quiescent condition" (no line activity).

The transmission line is electrically isolated from the device by means of a pulse transformer which acts both as a level translator and as a means of isolating the transmission line from device logic levels. Drivers and receivers are specified as two-wire balanced-pair devices. An electrical diagram of an HP-IL transmission line is shown in Figure 1-10.

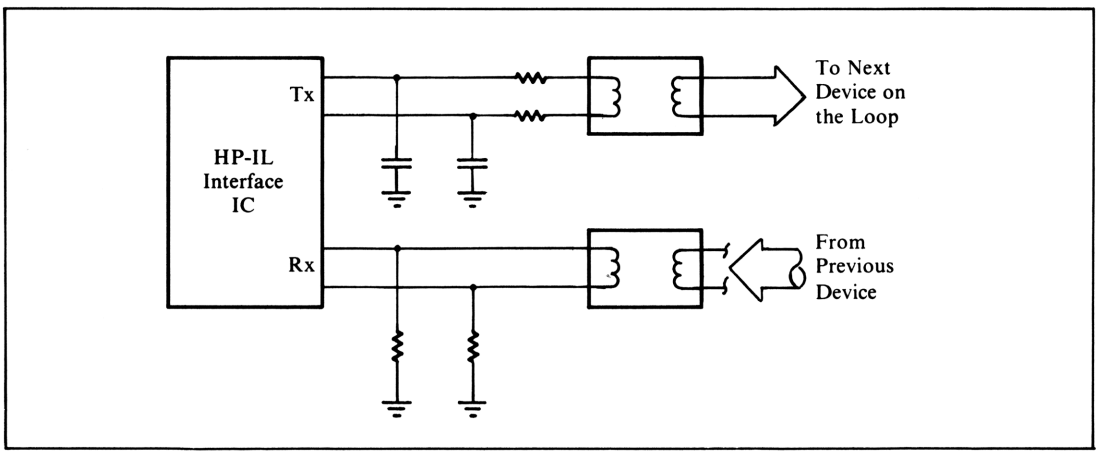


FIGURE 1-10. Diagram of an HP-IL transmission line

Transmission Medium

The HP-IL specification allows for a maximum distance of up to 10 meters between devices using simple two-wire cable. Distances of up to 100 meters are allowed when using shielded, twisted-pair cable.

Additional Capabilities

In addition to the basic list of HP-IL features, a number of additional functions have been defined. These functions allow devices to go to a power-down (or standby) mode and to wake up on command from a Controller.

The power-down mode of operation allows a Loop Controller to place all devices into a state where power requirements are minimal (a feature very important for battery-powered systems). If the Controller has the capability to institute real-time wake-up calls, it can wake up all devices on the loop to make measurements periodically or perform control functions. Once these functions or measurements are completed, the Controller can put the system back to sleep.

Device Triggering

Device Triggering is a function which lets a Controller initiate an action at a remote device. For example, a signal from a Controller to trigger a measurement reading at an appropriate time could be used in a laboratory system to make a series of readings at predefined intervals.

Addressing

HP-IL Controllers have the capability to initialize devices on the loop and to assign addresses to devices (a feature called Auto-Addressing). In this way devices may be added to the loop without concern for addresssing details (such as setting address switches).

APPLICATIONS

Because of its extremely lightweight design and minimal power requirements, HP-IL is particularly well-suited to a wide range of applications where portability is a factor. A few examples will illustrate some of the possibilities.

Portable Data Collection

Figure 1-11 shows a portable data-collection system comprised of HP-IL devices. Such a system could be used in situations requiring a portable unit for use in the field. A handheld unit such as the HP-41C could be programmed to accept information entered by field personnel. Information such as inventory counts, meter readings, or individual responses to marketing surveys could be directly entered into the handheld units as the information is collected.

Once the information has been gathered and entered into the handheld unit, it can be brought into the office. The data analysis system of Figure 1-11 then can be used to permanently file and analyze the field collected data. The data may be transferred to magnetic tape or disk via the HP-IL interface. Once it has been stored, you can analyze the data in any way you choose by running an appropriate program on the microcomputer (Controller). A copy of the results (inventory lists or the results of the

marketing survey) then can be printed out on the printer.

Remote Measurement Analysis

The system shown in Figure 1-12 is suitable for collecting data at a remote site with or without the assistance of a human operator. This field site could be in the remote reaches of a wilderness area, where environmental data is being collected or it could be an area of the laboratory set aside for a particular experiment. In either case, the portable Controller (an HP-41C, for example) could be programmed to take measurements at preset intervals and store the information for later analysis. Because its power consumption is extremely low, this remote system could operate for weeks at a time, totally unattended by a human operator.

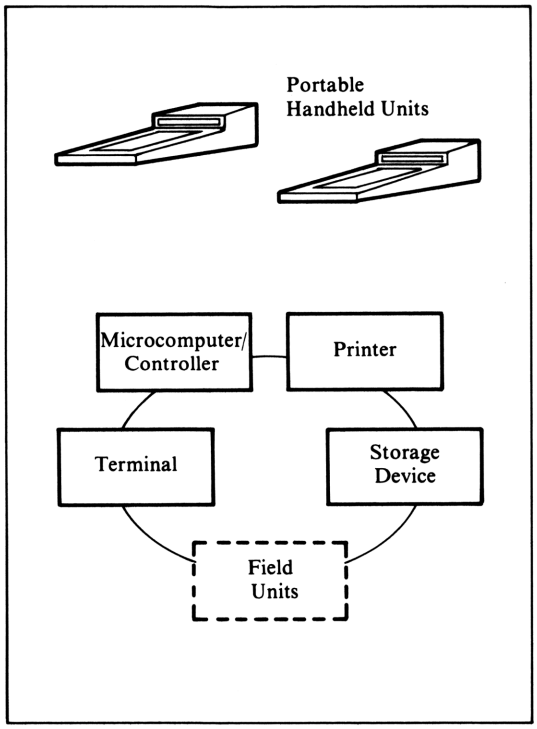


FIGURE 1-11. A portable data collection and analysis system

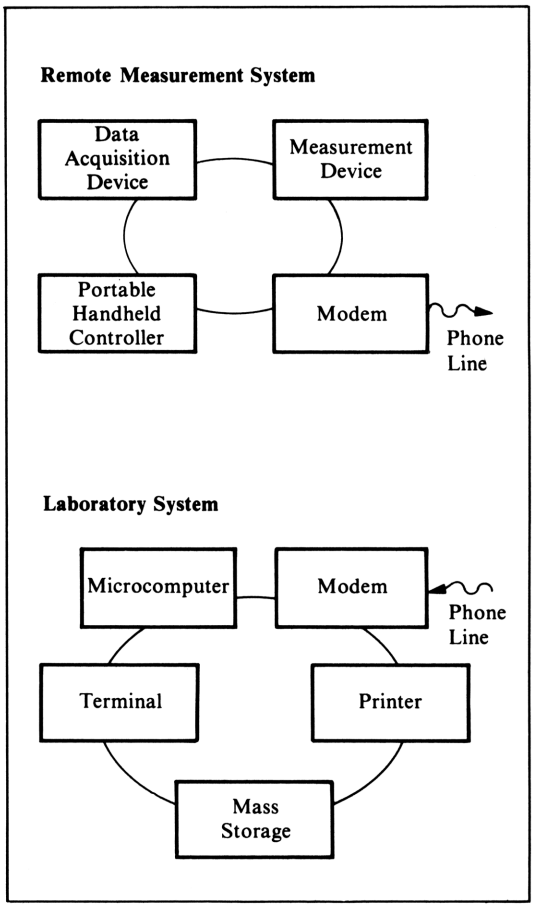


FIGURE 1-12. A remote measurement and laboratory analysis system

Once a sufficient amount of information has been collected, you could simply unplug the portable Controller and take it back to the laboratory system to store and analyze the data. After running the proper analysis programs, the results could be printed out. Alternatively, the data could be transferred via a modem over a telephone line for analysis in the laboratory.

Of course, these are only a few examples of the possible things you can do with an HP-IL system. Applications are certainly not limited to those requiring portability. Many personal or business applications also are ideally suited to the HP-IL system simply because of its low cost and simplicity.

For example, a system such as the one shown in Figure 1-13 could be used to prepare a financial model or perhaps to prepare a budget report. Once complete, the information could be sent to a central data base computer located thousands of miles away. Transmission of data via phone and satellite lines is possible through the use of a modem adapted to the HP-IL Interface Loop.

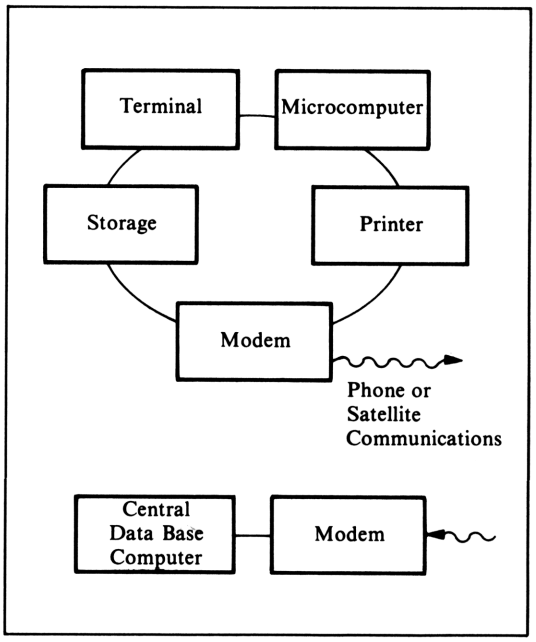


FIGURE 1-13. A possible business application involving data transmission

USING THIS BOOK

Since there are a number of ways to approach HP-IL, we have organized the book to address a wide range of users. Your level of involvement will depend upon your application and may range from a casual overview to a detailed knowledge of the inner workings of the HP-IL message protocol.

We began in Chapter 1 to give you an idea of what HP-IL is and how it can be used. Those of you considering the use of a system comprised of HP-IL products will find sufficient information in Chapters 1 and 2 to allow you to intelligently configure a small system.

While many will be satisfied with a cursory knowledge of what actually happens on the loop, often a more detailed knowledge of the inner workings of HP-IL is necessary. For instance, should you need to add a device to your system which was not designed to be used specifically with HP-IL or if you are interested in integrating or designing an HP-IL interface into a device of your own, you will certainly need to understand the workings of HP-IL messages.

Chapters 3, 4, and 5 introduce and cover the topic of HP-IL messages and how they are used in a typical loop system. These chapters provide a framework in which you can begin to familiarize yourself with the HP-IL message structure. Messages are also listed in a Glossary at the end of the book. A brief summary of the way messages are organized and classified appears in Appendix A.

Those interested in the design of HP-IL compatible equipment will find a basic hardware interface explained in Chapter 4. The details of the HP-IL functional specification necessary to begin designing an interface are introduced in Chapters 6 and 7. While we present enough information to allow you to feel comfortable with the HP-IL reference specification, this book is not intended to replace that document.

CHAPTER 2

Getting on The Loop

HP-IL is a system comprised of a number of devices linked through a common interface loop. There are a variety of ways to configure such a system. Deciding which devices you will need and how they are to interact will be determined by the requirements of your own application, and limited only by your imagination. The level of complexity you must deal with increases as you venture into new and more complex applications.

As a starting point, you can easily configure an HP-IL system from off-the-shelf HP-IL products. Systems configured in this way may be brought up with a minimum of effort and will require only that you connect the devices with the proper cables and turn them on.

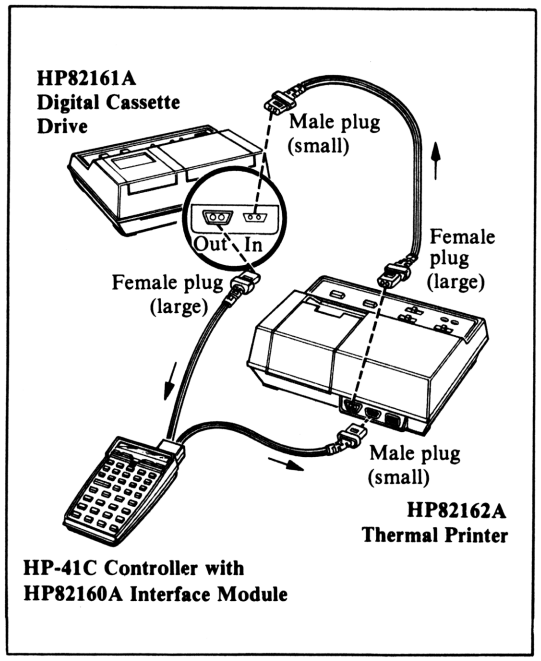
As you begin to see the possibilities for expanding your system, you may want to add a non-standard device to the loop. An adapter such as HP's 82166A HP-IL Converter lets you attach a wide range of non-standard devices to the interface loop.

Finally, should you decide you want to build an HP-IL interface into a device of your own design, there are a number of ways you can proceed.

But first, we'll configure a simple loop system with off-the-shelf devices.

A SIMPLE HP-IL SYSTEM

Perhaps the easiest way to introduce you to an HP-IL system is simply to put together a few of the HP-IL devices offered by Hewlett-Packard. Figure 2-1 shows a system comprised of an HP-41C Handheld Computer, an HP 82160A Interface Module, an HP 82162A Thermal Printer, and an HP 82161A Digital Cassette Drive.



REPRINTED COURTESY OF ELECTRONIC DESIGN

FIGURE 2-1. An HP-IL System configured from off-the-shelf HP-IL devices

By simply connecting the devices, as shown in Figure 2-1, the system is configured and ready.

An HP-IL system, such as the one shown in Figure 2-1, can provide you with an extremely lightweight and portable computing system. It would be appropriate for any application requiring on-line mass storage and print capabilities such as the portable data-collection system described in Chapter 1.

To begin, here's a description of each of the off-the-shelf devices shown in Figure 2-1.

The HP-41C is an extremely versatile, handheld computer. A closer look at some of its features may help to give you a better idea of its capabilities. The 12-character HP-41C display is capable of displaying alphanumeric strings of up to 24 characters in length. (Strings longer than 12 characters are horizontally scrolled across the display.) The keyboard allows for the entry of alphabetic and numeric character strings as well as supporting the standard calculator-type functions.In addition, you may redefine each key of the HP-41's keyboard to execute functions or programs stored in program memory. Memory is expandable via plug-in modules up to a maximum of 320 registers (1000 to 2000 program lines). With the addition of the HP 82160A HP-IL Interface Module, the HP-41 is capable of acting as an HP-IL System Controller.

The HP-IL Interface Module shown in Figure 2-1 plugs into one of four HP-41 expansion ports. The Interface Module consists of a set of routines in ROM to support the basic HP-IL functions and the interface electronics necessary to implement the physical interface to the HP-IL. With the Interface Module attached, the HP-41 is capable of communicating with and controlling other HP-IL devices.

The HP 82161A Digital Cassette Drive is shown in Figure 2-2. Programs and data are stored on HP mini-data cassettes. Each cassette is capable of storing up to 131K bytes of data. Data is stored on the tape in records of 256 bytes each, (up to 512 records may be stored per cassette). Data is read and written at a speed of 9 inches per second and the tape is searched or rewound at 30 inches per second. When used with

PHOTO COURTESY OF HEWLETT-PACKARD

FIGURE 2-2. HP 82161A digital tape cassette drive

the HP-41, all tape functions are executable either through direct entry from the keyboard or under program control.

The HP 82162A Thermal Printer, shown in Figure 2-3, is a portable device capable of printing up to 24 single-width characters or 12 double-width characters on 2¼"-wide HP thermal paper. The set of characters which the 82162A is capable of printing includes the standard 96-character ASCII character set, as well as an additional 127-character alternate character set. The 82162A is also capable of printing bar codes, plots, and graphics.

While all communication between these devices takes place over the interface loop, you will not need to know the details of the messages which are sent over the loop. The actual generating and handling of HP-IL messages will be taken care of by the Controller and the devices themselves. In the case of the system of Figure 2-1, all HP-IL communication is achieved through the functions contained in the HP-IL Interface Module's ROM. A program to control the interaction of several HP-IL devices would, therefore, consist of a series of calls to these functions. Alternately, devices

may be controlled directly through a series of keystroke commands.

As an example of this system's operation, suppose you want to execute a command from the keyboard of the HP-41C to search the tape and print a list of files found. You simply would enter the "DIR" function into the HP-41C. All communications over the interface loop would be invisible to you. The Controller translates your commands into the appropriate set of messages and handles all the details necessary to read the directory from the tape and pass this information to the printer.

To illustrate the use of a stored program to control the system of Figure 2-1, the following is a short routine which reads a data file from the tape drive and prints the file on the printer.

```
01  LBL "SYS1"   (program name)
02  "DAT"        (name of file to read)
03  READR        (read file into registers)
04  PRREG        (print registers)
05  END
```

Line 01 is a label which identifies this program; in this case the name of our program is SYS1. Line 02 places the name of

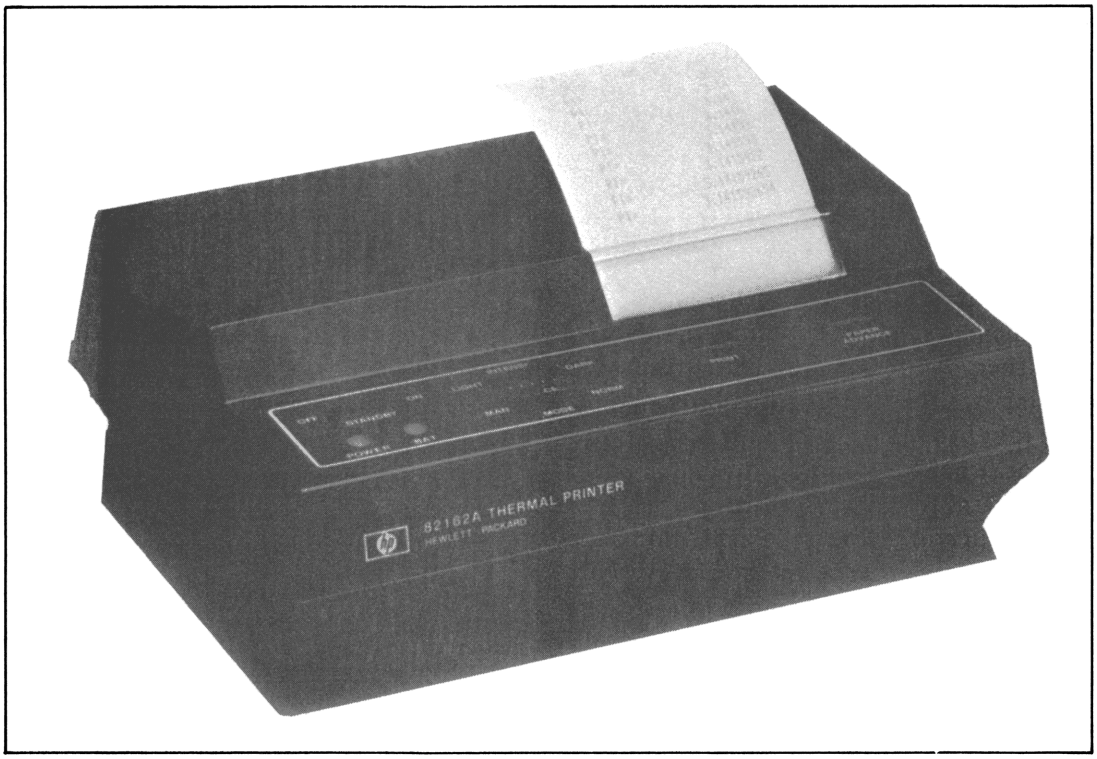


PHOTO COURTESY OF HEWLETT-PACKARD

FIGURE 2-3. HP 82162A thermal printer

the file we are going to read in the Alpha register of the HP-41C. Line 03 specifies the execution of the function READR, a function that reads data from a tape file into the HP-41's data registers. Line 04 causes the execution of the function PRREG, a function that causes all data registers to be printed on the printer.

While it is not our intention to explain the programming of a Controller such as the HP-41C, you should notice that the control of loop operations (when using off-the-shelf devices) is handled by the functions which are built into a particular Controller. Knowledge of the actual HP-IL messages sent on the loop is not really necessary.

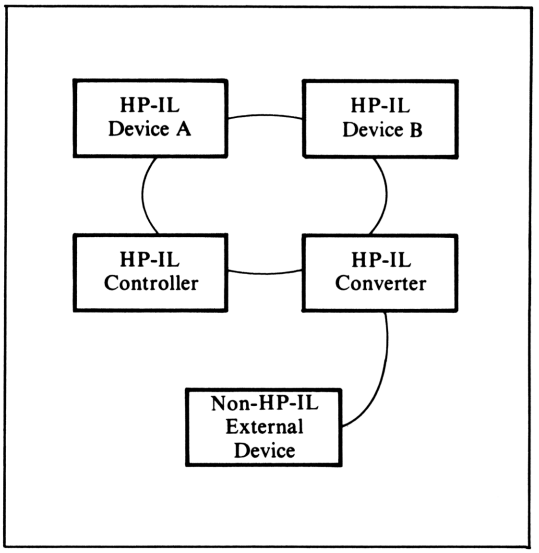


FIGURE 2-4. Interfacing a non-HP-IL external device to the Interface Loop

INTERFACING DEVICES

At some point you may need to add a device that does not have an HP-IL interface. An HP-IL Converter Module lets you do this without getting too far into the details of the HP-IL message structure.

An HP-IL converter is shown in Figure 2-4. It attaches to the loop as would any standard HP-IL device. A message bound for an external device is interpreted by the converter and output in a form suitable for use by the external device. Similarly, messages sourced by the external device are translated into the standard HP-IL format before they are sent out over the loop.

An external device capable of sending and receiving 8 or 16 bits of parallel data can be attached to the interface loop through the use of an HP-IL to parallel bus converter. An example of such a converter is the HP 82166A shown in Figure 2-5. Messages intended for the external device are sent to the converter via the loop interface and converted to a parallel data format for use by the device. Similarly, parallel data from the external device is passed to the converter where it is formatted into a standard HP-IL message. Handshake lines provide for the orderly transfer of data between the converter and external device.

If the device you wish to add to the interface loop has an RS232 serial interface, you could use an HP-IL to RS232 converter. Figure 2-6 shows such a configuration. While this figure shows a converter attached to a modem, any RS232 device could be similarly attached.

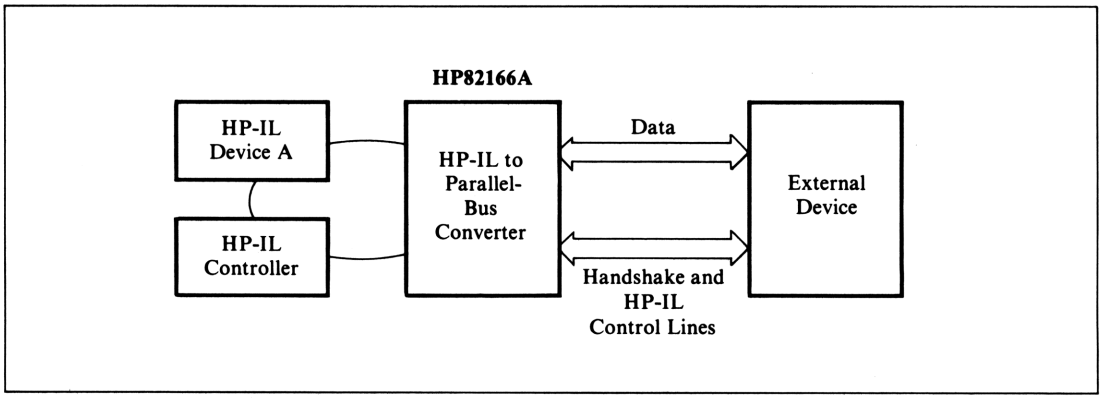


FIGURE 2-5. Interfacing an external parallel device to the Interface Loop

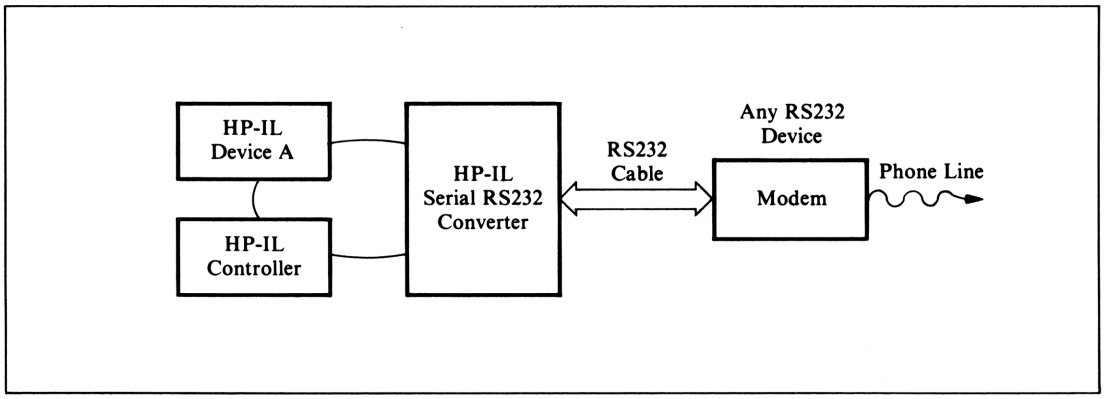


FIGURE 2-6. Interfacing to an external RS232 serial device

Interfacing an external device to the interface loop via a converter requires some knowledge of the interface signals necessary to pass data to and from a device. To give you a better idea of what is involved in setting up a converter interface, Figure 2-7 diagrams an HP 82166A Converter to Centronics-type parallel printer interface.

Converters such as the HP-82166A also require a small amount of programming to set up the proper operating parameters for the type of device being used. This entails defining such things as the number of bits (8 or 16) in the data interface, the number and sense (positive or negative) of the handshake lines, and the definition of the status word. This programming is usually in the form of HP-IL messages sent from the Controller to the converter. In the case

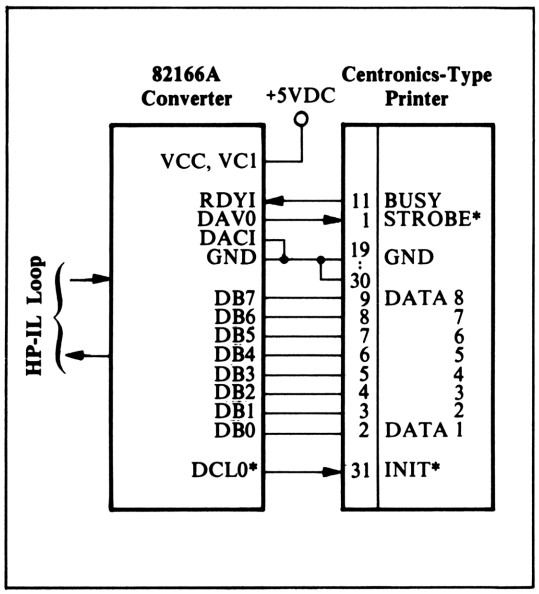


FIGURE 2-7. HP 82166A Converter to Centronics-type printer interface

of the interface of Figure 2-7, a message called "Device Dependent Listener 0" must be sent to the converter to initialize a control register to the value 10011010. The programming of a converter similar to the one shown in Figure 2-7 will be covered in greater detail in Chapter 3.

For maximum flexibility, you will need a general understanding of HP-IL messages in order to program a Controller to interact with a converter. The level of knowledge presented in Chapters 3 through 5 is certainly adequate to allow you to use and program such a converter. In many cases, however, a Controller's pre-programmed functions may be sufficient to allow you to implement your application without a knowledge of HP-IL messages.

BUILDING A DEVICE INTERFACE

Should you need to build an HP-IL interface of your own, either to interface an external device or to design into a device of your own composition, there are a number of ways you can proceed. The level of knowledge that you will require will be more detailed than that presented in this book. In fact, the information presented here will serve as an excellent introduction and overview of the HP-IL structure, but you definitely will need a copy of the formal HP-IL specification.

To start, you might consider designing an existing HP-IL converter into a product of your own. Converters such as the HP82166A can be easily integrated into a device package. Figure 2-8 depicts a con-

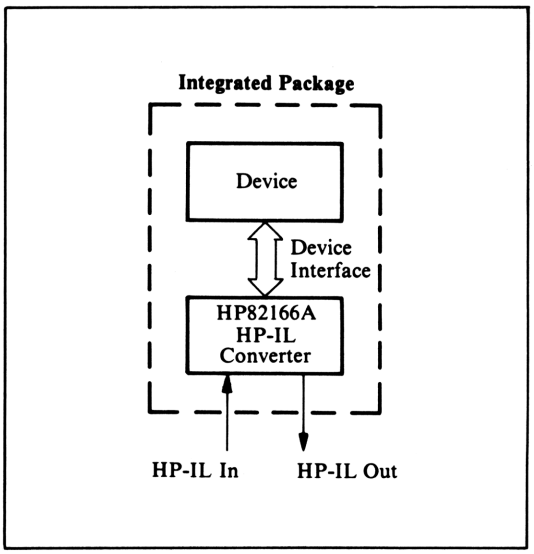


FIGURE 2-8. Integrated HP-IL Converter and device

verter integrated into an existing product. In this case, your device need only communicate with the converter via the appropriate parallel or serial interface. The converter provides all the necessary support to allow your product to be connected to an HP-IL system.

If you decide you need to actually design your own HP-IL interface, there are a number of basic functions you must implement in your design.

In Figure 2-9 the functions required to implement an HP-IL interface are broken into two basic groups: the loop interface functions and the device interface functions. Of the two, the loop interface functions are the most complex. Your implementation of the loop interface functions must provide for the proper handling of all types of the HP-IL messages.

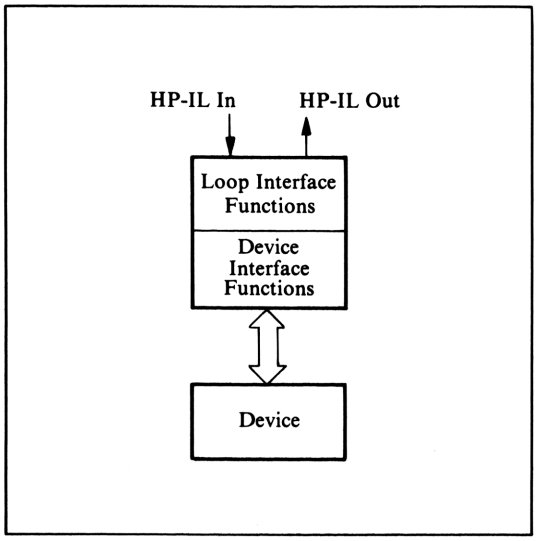


FIGURE 2-9. Basic functions of an HP-IL interface

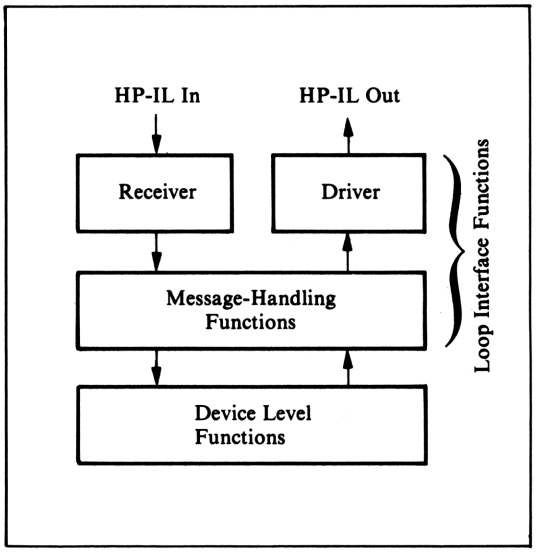


FIGURE 2-10. HP-IL Loop interface functions

Figure 2-10 illustrates an easier way of visualizing the loop interface functions. HP-IL messages are received by the block labled "Receiver" and sent out via the block labled "Driver." The characteristics of these blocks are described in detail in HP's functional and electrical specifications. HP's mechanical specifications cover the details and physical dimensions of the connectors used to attach a device to the interface loop.

The block labeled "Message-Handling Functions" in Figure 2-10 is the real core of the loop interface. It is responsible for such things as determining whether a message is addressed to this device and, if so, what signals need to be sent to the device. If the device needs to send a message, the message-handling functions must properly format

the message, send it, and verify that it was correctly received.

An interface such as this could be implemented in discrete logic or more conveniently in a single LSI interface controller chip.

The requirements of such an interface are covered in much more detail in Chapters 3 through 7 and are outlined in even greater detail in the HP-IL reference specification.

CHAPTER 3

Using a General-Purpose Interface to HP-IL

This chapter will discuss how you might connect a device to an existing HP-IL "system" using some type of general-purpose interface adapter. Thus, your level of involvement with the loop, and your understanding of loop operation, must be greater than if you simply bought a device from Hewlett-Packard. Before explaining some details of loop operations at this level, let's look at what characteristics and capabilities a general-purpose adapter to HP-IL might have.

The adapter to be discussed here allows devices with an 8-bit parallel interface to connect to HP-IL. Different adapters might permit devices with other interfaces (16-bit parallel, IEEE 488, RS-232, and so forth) to be connected.

A GENERAL-PURPOSE ADAPTER

Figure 3-1 is a block diagram of a hypothetical adapter that could be used to interface "foreign" devices to HP-IL. A discussion of each of the functional blocks will explain how this adapter would work.

Serial/Parallel Conversion

Information travels around the loop in a bit-serial manner. Since many devices use a bit-parallel method for data transfers, we will want the interface to perform serial-to-parallel conversion of information it receives from the loop and parallel-to-serial conversion of information it is putting back out onto the loop.

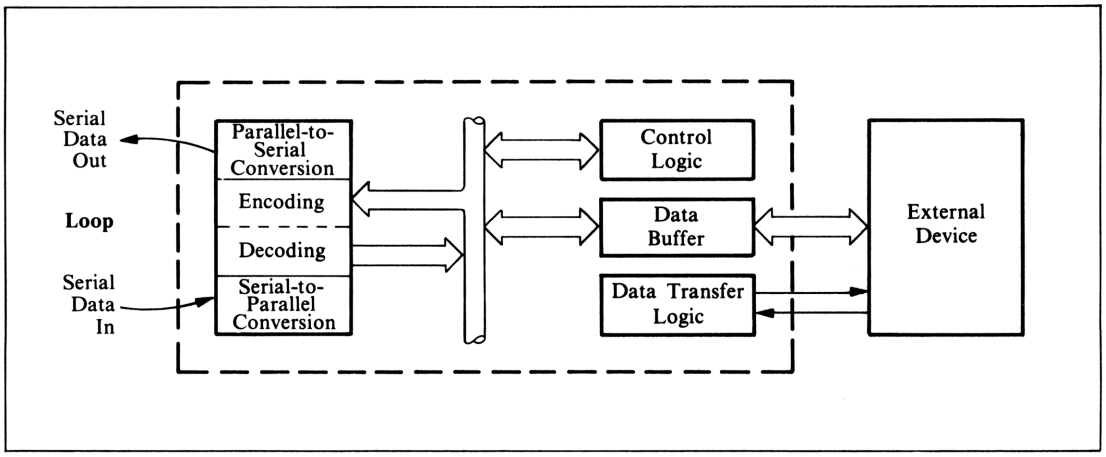


FIGURE 3-1. Functional block diagram of a general-purpose adapter to HP-IL

Message Encoding/Decoding

As the adapter receives the serial-bit stream comprising a message from the loop, it must perform some decoding of the message-bit pattern to determine such things as

- Is this message for me? (address decoding)
- Should the message simply be retransmitted?
- What category of message is this?
- Do I need to respond to this message?

Similarly, when the device utilizing the adapter needs to put a message out onto the loop, the adapter must be able to encode the information into the correct message format for transmission by the parallel/serial conversion logic.

Data Buffering

Since the data-transfer rates of devices can greatly vary from one type to another, and since it would be unreasonable to expect devices to be overly dependent on the data-transfer rate on the loop (and vice versa), a useful general-purpose adapter offers some data-buffering capability. This provides the greatest flexibility for devices, as well as minimizing degradation of data-transfer rates on the loop that might occur if a particular device is excessively slow in accepting or supplying data.

Device Interface Logic

The transfer of information between the external device and the adapter must proceed relatively independent from activities and exchanges between the adapter and the loop. Assuming that this adapter is intended to interface to devices that present a parallel interface, Figure 3-1 shows the adapter providing an 8-bit parallel interface to the external device. Some data-transfer logic also would be included to

synchronize and control exchanges of information between the adapter and the device.

Control Logic

This is sort of a "catchall" functional category that would include the logic necessary to control timing sequences, determine the appropriate response and reactions to various categories of messages, and so on. This logic might consist of PLA (Programmable Logic Array) devices, ROM (Read Only Memory) devices, and so on. This is the logic that provides the "intelligence" of the adapter. At this point, rather than being concerned with how this intelligence is implemented, simply assume that some appropriate logic is provided to accomplish the required functions. Chapter 4 will dig deeper into this subject to show how this logic might be implemented.

THE DEVICE AS A LISTENER

Assume that the device you are going to interface to HP-IL via the adapter is a thermal printer. Before discussing how this printer will interact with the loop, the characteristics of the device and its role as part of the loop will be defined.

First, without knowing anything else about the printer, you can assume that it is purely an output device. The system (that is, other devices on the loop) will send information to the printer—the printer won't be sending information to other devices on the loop. Thus, you can expect the printer to primarily play the role of Listener. As a Listener, it has certain responsibilities to the system, certain ways in which it must respond to messages, and certain message sequences it must adhere to. A discussion of each of the messages that might be exchanged between the printer and the HP-IL system (via the adapter) will illustrate the rules that govern loop operation. It will also give some insight into the logic that you might have to provide in your printer to enable it to operate within the system.

But first, let's define a few of the characteristics of this printer since this will determine, to some extent, the way in which it acts out its role as Listener.

You want to be able to put the printer into a low-power, standby mode by issuing commands from the System Controller. This would give it the capability of using the printer in unattended applications where the printer need be activated only intermittently for such things as logging readings. Of course, the printer must also respond to a "power-up" command from the system prior to any printing operation.

The printer should be flexible in its capabilities. For example, it should have more than one character set that it can use, be able to use different line spacings, and so on. This implies that it must have some degree of programmability and, therefore, will receive information from the system in addition to the data characters to be printed.

The printer must be able to inform the system about its status; for example, if it is out of paper, if it is currently involved in a printing operation, what its current mode

of operation is, and so on. In most systems, you would want all devices to be capable of providing at least minimal status information. This capability requires that a device also be able to play the role of Talker on the loop, since a Listener is only a recipient of information—never an originator. Thus, even a pure output device like a printer must be able to assume the role of Talker occasionally.

Now, consider the messages you can expect the printer to encounter during loop operations. Not all of the HP-IL messages will be described in the following paragraphs; only those messages that the printer responds to. All other messages that the adapter/printer receives will simply be passed on to the next device on the loop.

COMMAND GROUP MESSAGES

The command group messages are used to establish initial operating conditions on the loop, change the operating modes and roles of devices on the loop, and generally maintain discipline and control of the loop system. The following are the command group messages that a printer, such as the one defined here, might be expected to receive via an interface adapter:

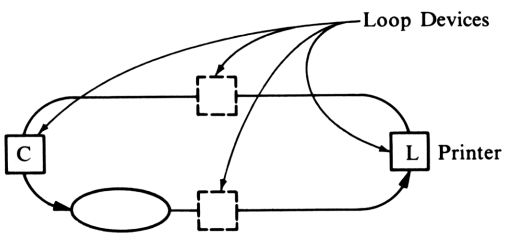
- Interface Clear (IFC)
- Device Clear (DCL)
- Loop Power-Down (LPD)
- Listen Address (LAD)
- Unlisten (UNL)
- Talk Address (TAD)
- Untalk (UNT)

Before actually discussing the messages, let's describe an illustration to be used in this and subsequent discussions of loop operations.

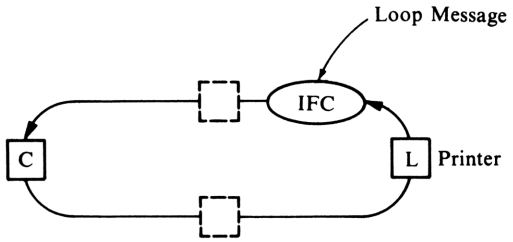
Loop Nomenclature

The examples that follow will use square boxes drawn with solid lines to represent devices on the loop that have been assigned active roles such as Controller or Talker, or that may soon be involved in the transaction/operation being discussed. Square boxes drawn with dotted lines indicate other devices on the loop that have either not yet been assigned roles or whose presence on the loop is insignificant in the operation or transaction being described.

For example, in the following illustration, the device represented by the box on the left is the System or Loop Controller (C) and the device represented by the box on the right is an active Listener (L).



Messages that are being passed around the loop are represented by ovals. For example, the message being passed around the loop in the following illustration is the Interface Clear (IFC) message.

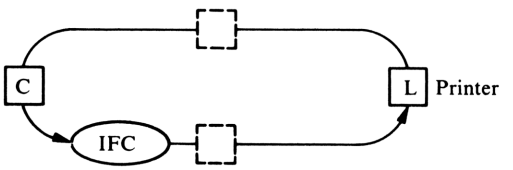


Some of the symbols used in these illustrations may be enhanced as the discussion progresses, but the extensions of the nomenclature will either be quite obvious or explained.

Interface Clear (IFC)

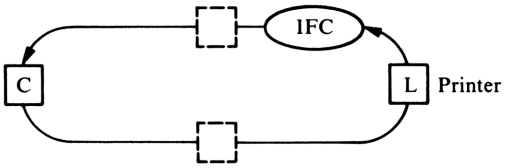
The IFC message is equivalent to a System Clear message. It can only be sourced by the System Controller, which may or may not be the active Controller on the loop, and allows the System Controller to take control of the loop at any time. Typically, the System Controller will send a series of IFC messages around the loop when the system is first powered-up to put all devices on the loop into an inactive or "idle" condition.

When the printer's interface adapter receives and decodes the IFC message, it will simply remove itself from the active Talker or Listener state if it happened to be in one of those states at the time the message was received. Thus, it will effectively be in an idle state waiting for the Loop Controller to assign it a new role.



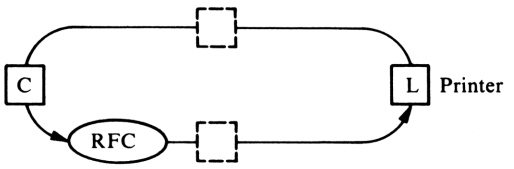
Note that this Interface Clear message has no direct effect on the printer itself. It is only the interface adapter that is affected.

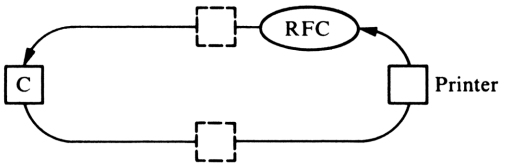
Since the IFC message is directed to all device interfaces on the loop, it might require a lengthy period of time for the message to once again reach the System Controller, if each interface waits until it has cleared itself before passing on the message. To speed up this process, each interface is expected to pass on the message immediately and then proceed to put itself into the idle state. Thus, when the IFC message has returned to the System Controller, it does not necessarily mean that all interfaces on the loop have already been cleared.



In this situation, the Controller will send a special Ready For Command (RFC) message, which will be described shortly.

This RFC message will be held up by each interface on the loop until it has completed any previous clearing operation.





This same sequence (that is, immediately passing on the command and then holding up the RFC message until ready to receive the following command or other message) is used for all command group messages to speed up processing on the loop by all devices. Other messages are simply held up by each device until they are ready to receive another message. The simple rule the Controller will obey is to send an RFC message as soon as each command message returns. This will be discussed in more detail in the section on ready group messages.

Device Clear (DCL)

The DCL message can be used by the Loop Controller to reset a device (as opposed to a device's interface). Note that all interfaces on the loop must be capable of responding to the IFC message, since it is the means by which the System Controller takes control of the loop. However, not all devices need be capable of responding to the DCL message. Furthermore, the action that a particular device takes in response to the DCL message is entirely left up to the device. Its only responsibility to the loop is that it pass the DCL message on to the next device.

The printer then should respond to the DCL message. Therefore, when the printer's interface adapter receives and decodes the message, it must send some indication to the printer electronics to cause the appropriate device clear or reset functions. This can be illustrated as shown in Figure 3-2.

When the interface adapter has decoded the DCL message, it sends the CLEAR signal to the printer. Within the printer, the CLEAR signal would cause such actions as positioning paper to top-of-form, moving the printing mechanism to home position (carriage return), clearing its internal data buffer of any previously received data, and so on. All of these actions are device-dependent and of no concern to the loop or even to the printer's interface adapter.

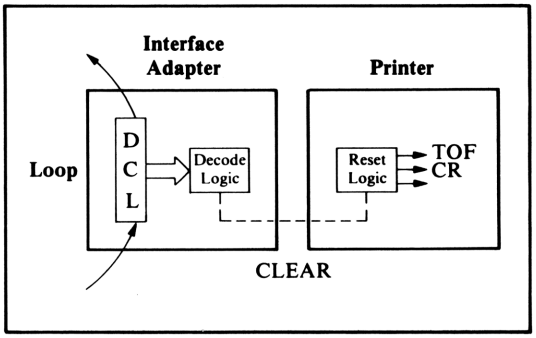


FIGURE 3-2. The effect of a DCL message

Just as with the Interface Clear message, devices on the loop should not have to wait for preceding devices to complete their "clearing" operations before passing on the DCL message. Instead, each device should immediately forward the DCL message and then proceed to establish its CLEAR conditions. The Controller will follow the DCL message with the Ready For Command (RFC) message to determine when all devices have completed execution of the DCL message.

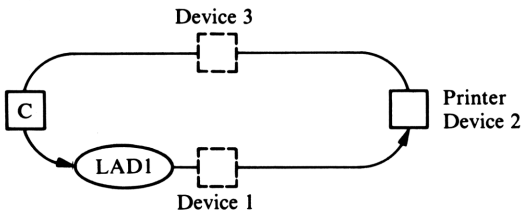
Listen Address (LAD)

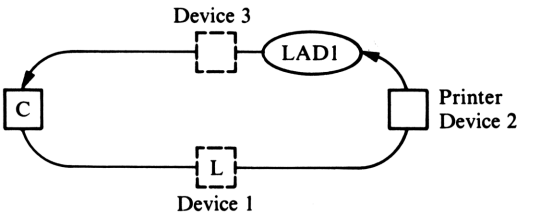
The LAD message is sent by the Loop Controller to cause a particular device to become an active Listener. In order for a

device to receive data from other devices on the loop, it must first be addressed as a Listener. Since there may be times when the Controller wants to send the same information to more than one device, the loop can have more than one active Listener at a time. However, since the LAD message includes an address, it is directed to only one device at a time. Thus, if there were to be more than one active Listener at a time, the Controller would have to send out a separate LAD message for each device. Each LAD message would, of course, be followed by an RFC message.

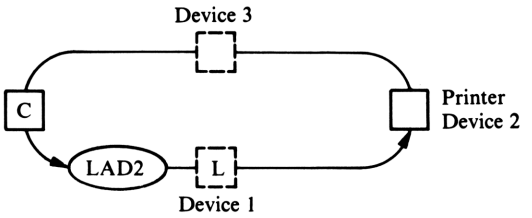
There is another Listen Address message defined in the HP-IL specification called My Listen Address (MLA). This message is simply the LAD message containing an address which matches that assigned to the device receiving the LAD message. For example, assume that the printer interface has been assigned an address (using switches installed on the card or some programmable method) of 2.

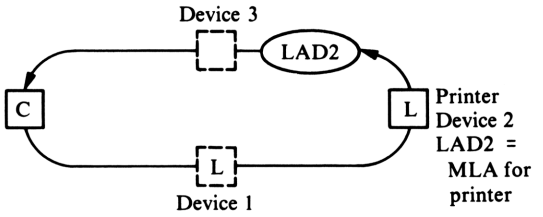
If the LAD message is received containing an address of 1, then the interface simply passes the LAD message on to the next device. The LAD message in this case was not directed to the printer interface, but to Device 1 which assumes the role of Listener.





When an LAD message is received that contains an address of 2, however, the printer interface decodes this as My Listen Address (MLA) and takes the necessary steps to prepare itself for the role of Listener. (Notice that Device 1 has also retained its Listener role as illustrated.)





Thus, the differentiation between LAD and MLA is made simply to clarify the different actions that the interface logic will have to accomplish, based on whether an address match is made.

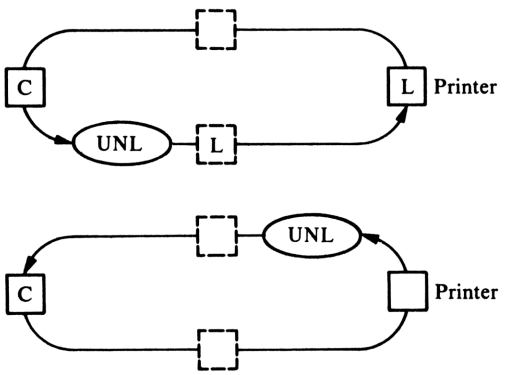
Once the printer interface is an active Listener, it can then accept data coming in from the loop. This data could consist of

characters to be printed, control information for the printer, or programming information for the interface adapter. Interpretation of the contents of a data message is up to the device designer (in this case, the adapter designer and the printer designer).

Unlisten (UNL)

The UNL command is used to return all addressed Listeners to an idle state. Thus, it effectively deselects any devices that previously had been addressed as Listeners using LAD messages.

Note that although you can only select or address one device at a time using the LAD messages, the UNL message causes *all* Listeners to be reset or deselected. The Loop Controller would therefore use this command whenever it has to address a new Listener (or Listeners) for another data transmission.



When the printer interface receives the UNL message, it would simply place itself in the idle state where it is still on the loop, but is playing no active role.

Loop Power-Down (LPD)

The LPD message is used by the Loop Controller to place devices on the loop in a powered-down or standby state to conserve power. This capability can be quite useful in systems where power conservation is important, such as in unattended remote applications where the equipment needs to be fully functioning only during certain small intervals of time. For example, if the printer was being used to log readings that were taken only every few hours or so, it would be useful to be able to put it in power-down mode when it was not actually being used by the system.

The LPD command message is a universal (unaddressed) command message. Thus, when the Loop Controller sends this message out, it is directed to all devices on the loop. However, all devices are not required to include this capability. Only those devices that have implemented the power-down function will respond; other devices will simply pass on the LPD message to the next device in the loop.

Our printer has this low-power, standby capability and is designed with a three-position power switch: OFF-STANDBY-ON. When the switch is OFF, all power is removed from the printer and from the interface adapter. If it were on the loop at this time, the loop would be inoperative, since the printer's interface would not be able to pass messages along the loop. When the switch is ON, the printer is fully operational, but it will not respond to the LPD message—it simply passes on the message. When the switch is set to STANDBY, the

printer is fully operational and can respond to the LPD message. When an LPD message is received, the printer will be put into the low-power state. Only that circuitry on the interface adapter needed to respond to a subsequent message on the loop will still have power applied.

When the printer's switch is on STANDBY and the interface adapter receives the LPD message, the interface prepares to enter the low-power state. The LPD function is not actually executed, however, until a Ready For Command (RFC) message is received. When the interface receives the subsequent RFC message, it causes all unnecessary power to be removed from the printer.

Notice that the way this device reacts to the "command message-RFC message" sequence is different from the previous sequences we have discussed. In the other situations, the device immediately would pass on the command message to the next device and then begin executing the command. With the LPD message, however, the device waits until it receives the subsequent RFC message before it executes the LPD command.

There are two reasons for this difference. First, if you are powering down the loop, it will probably be for an extended period of time. Therefore, there is no real reason to try to speed up the operation by a few fractions of a second. Second, there is no special command or message that causes devices to be powered up again. Instead, devices that are powered down must reactivate themselves as soon as they begin to receive any message on the loop. Thus, if the device had already completed execution of the LPD command when the RFC command arrived, it would immediately power up itself. So, devices must wait until the subsequent RFC message is received before they begin executing LPD.

This power-down/power-up scheme is simple and efficient, but it means that all devices on the loop that are capable of responding to the LPD message will be in the same state; you cannot selectively power a single unit up or down.

READY MESSAGES

The command messages just described can only be sourced by the Controller. The ready messages to be discussed here can be sourced by either the Loop Controller or by a device that is an active Talker. These messages are generally used to coordinate and synchronize the transfer of commands and data between devices on the loop. The following are the ready messages to which a printer might be expected to respond:

- Ready For Command (RFC)
- Send Status (SST)
- End of Transmission OK (ETO)
- End of Transmission Error (ETE).

Ready for Command (RFC)

The RFC message is used by the Loop Controller to determine if an immediately preceding command message has been executed. To understand the need for this message, the focus of this discussion must be broadened. Normally, there is only one message at a time traveling around the loop. The Controller or Talker that sent the

message waits until it successfully receives back that message before it sends another message. Usually, a device does not pass on a message until it is ready to receive another message. This technique ensures that Talkers and Controllers do not send messages faster than the other devices on the loop can handle them.

Obviously, if there are a number of slow devices on the loop and they each delay the message until they are ready to receive another, the loop speed may be very slow. Data messages are usually intended for only one device and other devices immediately pass on the message. Thus, satisfactory throughput can be maintained.

Command messages, however, are often directed to all devices on the loop at the same time. In order to speed up execution of these command messages, devices are expected to pass on the message immediately and then begin execution of the command. So, all devices on the loop can be executing the command more or less in parallel.

When the Controller receives the command message after it has been around the loop, the message does not indicate that the command has been *completed* by all devices. It only indicates that it has been *received* by all devices. The Controller then sends out the RFC message. This message will be held by each device and not retransmitted onto the loop until execution of the previously issued command message is completed. Thus, it is only when the RFC message has returned to the Controller that completion of a command message is ensured.

Send Status (SST)

The SST message is used by the Controller to cause the addressed Talker to send some status information back to the Controller. Remember, we stated at the outset of the description of the printer's characteristics, that it would have to be able occasionally to assume the role of Talker in order to send status information out on the loop. Loop protocol says that if you want to generate some information and send it around the loop, then you have to assume the role of Talker. Never mind that 99% of the time a printer will be just a Listener. The issue of "mostly Listener-occasionally Talker" will be explained after the discussion of Send Status messages.

The SST message is used to request some information from a device (our printer/interface adapter combination). The information that the Controller is requesting will typically be rather device-specific information. For example, for the printer, the Controller may want to know such things as

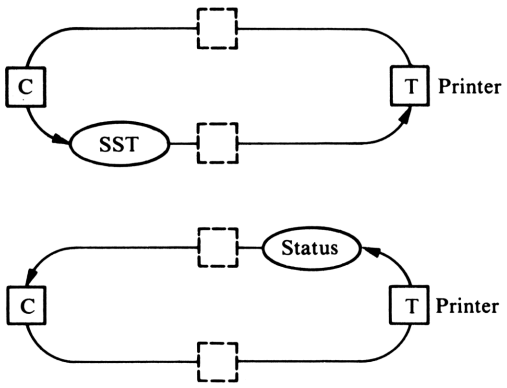
- Are you busy printing?
- Are you advancing paper?
- Do you have an error condition?
- Is your buffer empty?
- Are you in some sort of special or escape mode?

The SST message does not include any device address information. The message is always directed to the active Talker. Since there can only be one active Talker on the loop at any time, there is no problem in determining who the SST message is

intended for. When devices that are not the active Talker receive the SST message, they simply pass it on around the loop. Thus, if there were no active Talker on the loop, the Controller would receive back the unchanged SST message. (Of course, you would not expect this to happen, since the Controller should always know what roles are being played by the various devices on the loop and would not send the SST message out unless there was an active Talker out there.)

Assume that the printer interface adapter has previously been assigned the role of Talker.

When it receives the SST message, it does not pass the message on to the next device on the loop as is done with most messages. Instead, it substitutes its own message—one or more bytes of status information—and sends this information out on the loop.



As other devices further on along the loop receive the status bytes of information sent out by the printer, they simply pass them along until they finally return to the Controller. The Controller, of course, is not expecting to get the SST message returned, but rather the status byte or bytes from the printer.

There are some general guidelines, some specific rules and some suggestions in the HP-IL specification about the format for the status bytes that devices send back to the Controller. This level of detail is not relevant to this discussion, however, and will be described in detail in Chapter 4.

Figure 3-3 is a more detailed block diagram of an interface adapter that might be used with a printer. You will notice that we've added a few blocks that were not included in Figure 3-1. Since the interface adapter will be expected to respond to the SST message, a status register is included. This status register will be used to hold information from the printer about its status and might also contain information about the status of the interface adapter itself. Two holding registers have been added: one for incoming messages, and one for outgoing messages.

Figure 3-4 is a block diagram showing what happens when the interface adapter receives the SST message.

The SST message is simply discarded after it has been decoded. The interface adapter then sends a Data Byte (DAB) containing its status information out on the loop to the Controller. (DABs will be discussed in a few more paragraphs.)

The interface can send as many bytes of the status data as it wants. Remember, it is now the active Talker on the loop.

As the Controller receives the status byte or bytes from the printer, it processes them

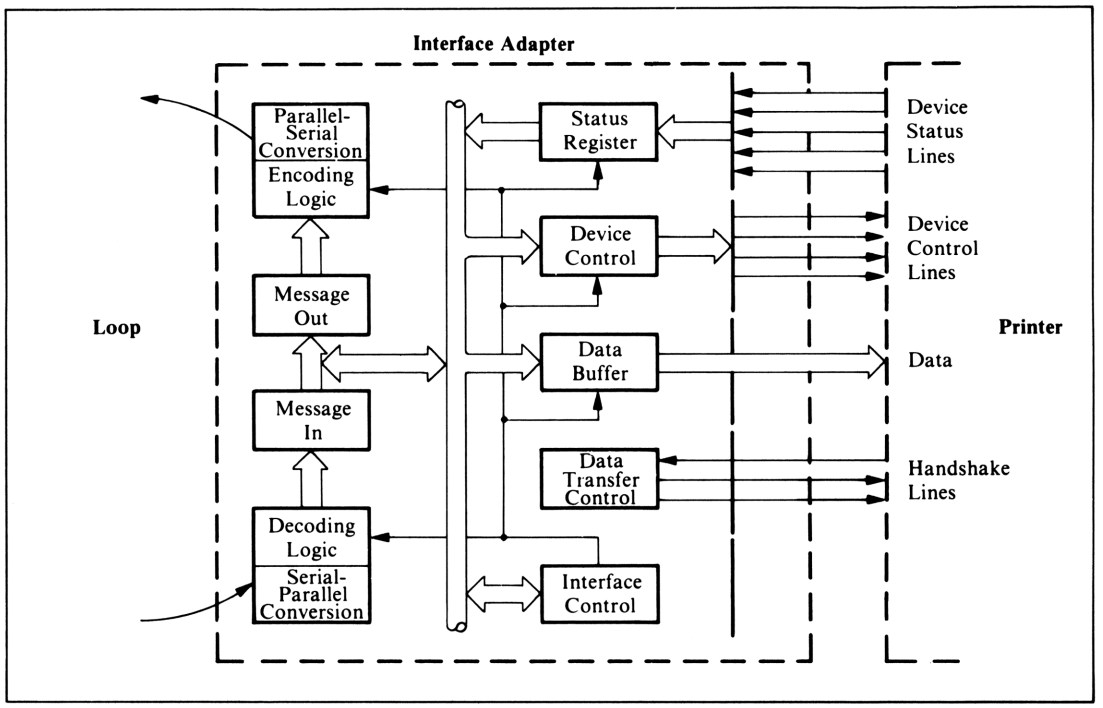
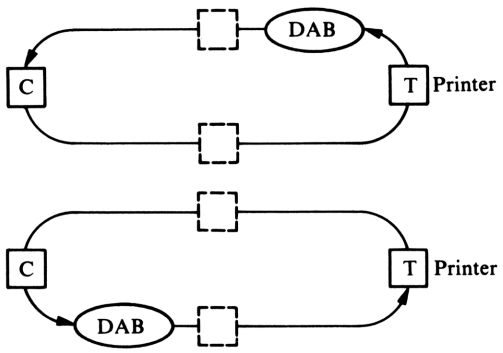


FIGURE 3-3. Detailed block diagram of an interface adapter for a printer

in whatever way is appropriate to your system and then passes the status byte back around the loop to the printer. This is the printer's indication that it can send the next byte.



Consider one final point about the SST message. The sequences just discussed comprise what is known as "serial polling." This is a technique that can be used by the Controller to ascertain the status of each device on the loop one at a time by individually addressing it as a Talker, sending it the SST message, deselecting it as a Talker, and proceeding to the next device on the loop. Another method of polling—"parallel polling"—will be discussed in later chapters.

End of Transmission (EOT)

After the printer/interface adapter has sent and received back its status bytes, it must

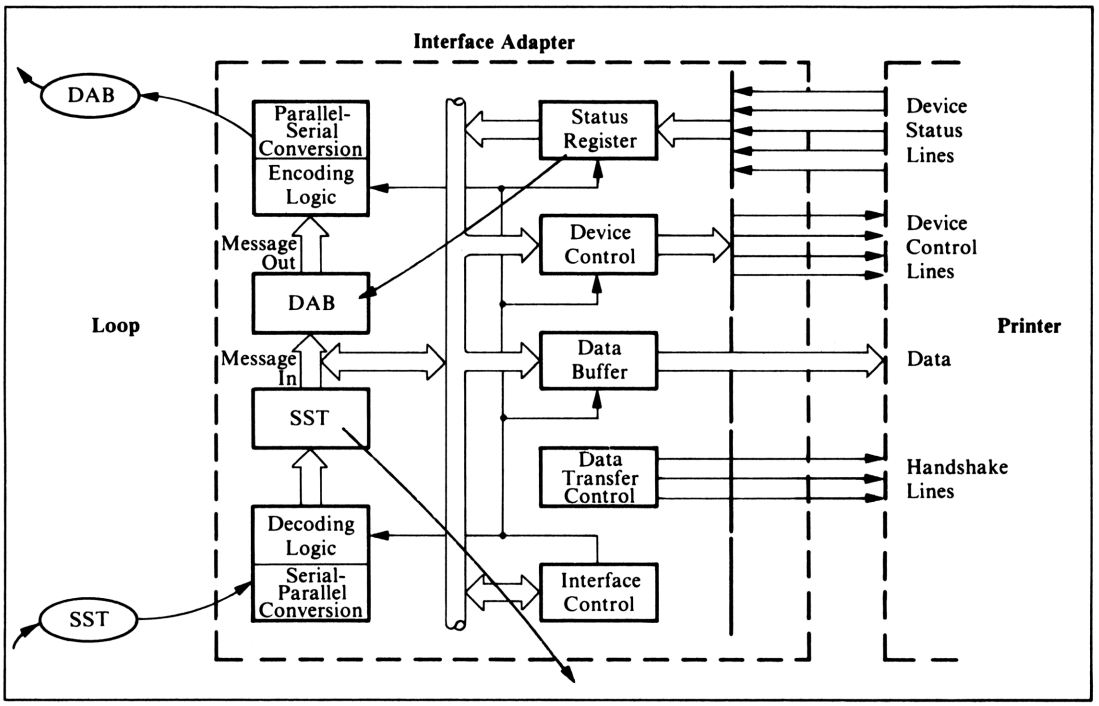


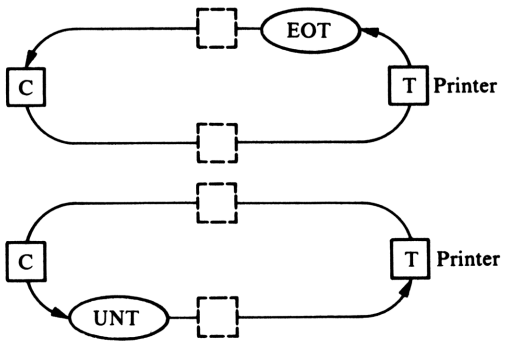
FIGURE 3-4. Effects of the SST message

send a special message to the Controller to let it know that the printer has finished with its current role as active Talker. The message used to signal this completion is the End Of Transmission (EOT) message.

Remember, when the Controller receives each of the status bytes from the printer it passes them on around the loop back to the printer.

When it finally receives the EOT message, however, it does not pass this message on. Instead, since EOT signals the completion of the printer's role as active Talker, the Controller now proceeds to issue another command.

For example, the Controller might issue the Untalk (UNT) command to cancel the printer's role as a Talker (the UNT command will be explained later).



There are two forms of the End of Transmission message: End of Transmis-

sion, OK (ETO); and End of Transmission, Error (ETE). These messages are always sourced by the active Talker and sent to the Loop Controller. As shown, these messages do not travel all the way around the loop, but stop when they reach the Controller. As data bytes are returned to the Talker, it can check them to determine if the bytes returning are the same as those sent out. If they are, the Talker would make the ETO message the terminating message. If any bytes returned incorrectly, the Talker should terminate the transmission with the error form of the message, ETE. The Controller could then react appropriately to this error indication, for example, by attempting a retry.

Note that devices on the loop are not required to check returning bytes for errors. If they don't do error-checking then they are not permitted to source the ETE message. Obviously, however, good system design procedures would incorporate error-checking. Note also that the Controller's response to an error indication is not specified. Recovery procedures are left entirely up to the system designer.

THE TALK COMMAND MESSAGES

As explained earlier, the printer must play the role of Talker when it wants to put status information out on the loop. There are two talk-related command messages and they correspond to the two listen command messages described in preceding paragraphs—the Listen Address (LAD) and Unlisten (UNL) messages. The two talk commands are: Talk Address (TAD) and Untalk (UNT).

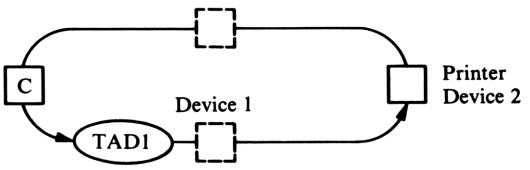
Talk Address (TAD)

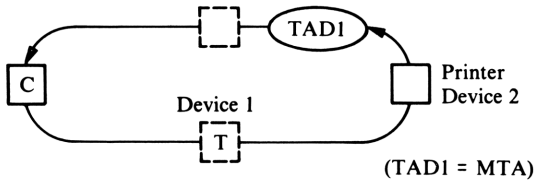
The TAD message is sent by the Loop Controller to cause a particular device to become the active Talker. In order for a device to be able to put any information out onto the loop, it first must be addressed as a Talker. While there can be multiple active Listeners on the loop at the same time, there can only be one active Talker on the loop.

Like the Listen Address (LAD) command, the TAD command message includes an address and is directed to a single, specified device on the loop. Also, as with the LAD message, there is a form of the TAD message command that is defined as My Talk Address (MTA). This message is simply the TAD message that contains an address which matches that assigned to the device receiving the TAD message. For example, assume that our printer interface has been assigned an address of 2.

If the TAD message is received containing an address of 1, then the printer interface simply passes the TAD message on to the next device.

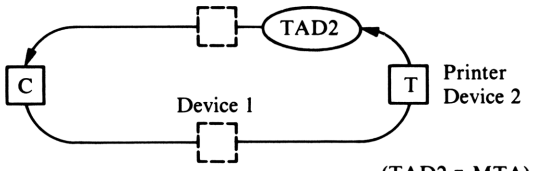
The TAD message in this case was not directed to the printer interface, but to Device 1, which assumes the role of Talker. In this illustration, Device 1 decodes the TAD message as My Talk Address (MTA).





(TAD1 = MTA)

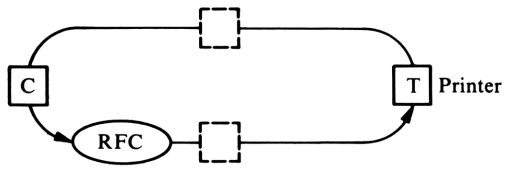
When a TAD message that contains an address of 2 is received, however, the printer interface decodes this as My Talk Address (MTA) and takes the necessary steps to prepare itself for the role of Talker.



(TAD2 = MTA)

Notice that Device 1 is no longer shown as a Talker (T), since there can only be one active Talker on the loop at a time. In fact, when any device receives a TAD address message that contains an address other than its own (not MTA) the device must automatically deselect itself as a Talker. When a device receives a TAD message whose address does not match its own address, the device decodes this message as OTA (Other Talker Address).

Of course, the TAD message must be followed (just as with the LAD message) by an RFC (Ready For Command) message to ensure that the devices have completed the process of adopting their new roles.



Untalk (UNT)

The UNT command message is used to return the active Talker to the idle state. As discussed in the preceding paragraphs, the active Talker can also be returned to the inactive state simply by issuing another TAD message addressing a different Talker. So, you rarely will have to use this message. It might be useful in some special circumstances, however, such as when you need to have no active Talker device on the loop. Therefore, the specification has included this command message to ensure that there are no logical or functional "holes" that might cause unforeseen difficulties under special circumstances. Since this command message will be infrequently used, it will not be discussed any more at this point.

DATA GROUP MESSAGES

All of the messages discussed so far are used to maintain system discipline, establish system configurations, and so on. They might, therefore, be considered system overhead, since the purpose of a system obviously is not to simply maintain itself, but to accomplish some task. The task that an HP-IL system is expected to accomplish is the transfer of information or data between devices that comprise the system. The data transferred around the loop are contained in Data Byte (DAB) messages and these messages will constitute most of the traffic on the loop.

Data Byte messages are sourced by the active Talker on the loop and sent to the

active Listener or Listeners on the loop. Data Byte messages do not include an address. If you are a Listener and you receive a Data Byte message, then you must assume that it is directed to you; that is, the primary function of a Listener is to receive data bytes.

The information that is contained within a data message is left entirely up to the system designer. This information and its interpretation is completely device-dependent. You can also code the information in any way you want, although it is strongly recommended that ASCII be used whenever possible to provide maximum compatibility with other devices.

When devices that are not Listeners receive Data Byte messages, they must simply pass them on around the loop. Listeners receiving Data Byte messages also pass them on around the loop while retaining the information for their attached device. Thus, if our printer's interface adapter has been addressed as a Listener, all data messages that it receives would be passed on around the loop and it would also send them to the printer.

There is a special type of data message called the End Data Byte (END). This message is the same as the Data Byte just described and also contains a byte of data, but it can be used to indicate an end-of-record condition to the Listener. The effect of the END message on a device is up to the designer and it is treated exactly the same as any other Data Byte by the system. It might be used for such things as an end-of-line indicator for some devices.

It is important that you not confuse the END message with the EOT messages described earlier. The END message does not terminate a transmission. The Talker must always send an EOT message when it has finished. Also note that use of the END message is optional. If a device does not require it, you need not use it.

CHAPTER 4

A Component Level Interface to HP-IL

This chapter will discuss how you might create an interface to HP-IL using some type of LSI (Large Scale Integrated) component or "chip." This level must begin to deal with the details of the HP-IL specification even more intimately than in Chapter 3. Once again, the discussion will begin by describing some of the characteristics and capabilities that an LSI HP-IL interface component might have.

AN OVERVIEW OF AN INTERFACE CHIP

Figure 4-1 is a block diagram of a hypothetical chip that could be used to interface "foreign" devices to HP-IL. If you compare this figure to Figure 3-1 in the preceding chapter, you will see many similarities. This should not be surprising, since the functions provided by the chip must be quite similar to those provided by the adapter described in Chapter 3.

The level of detail shown in Figure 4-1 is greater than that in Figure 3-1 because HP-IL operations will be discussed in more detail in this chapter. But first, the functions provided by our interface chip should be described.

Receiver Control Logic

This logic converts the input signals from the loop to logic levels and controls loading of the serial-bit stream into the input buffer. Every message on the loop is sent as a sequence of 11 bits. These 11 bits comprise what is called a message "frame." The first bit, called the "sync bit," is coded in a special way so that devices on the loop can recognize the beginning of a message frame. The input detector logic must, therefore, monitor the inputs from the loop to detect the sync bit marking the beginning of a message frame.

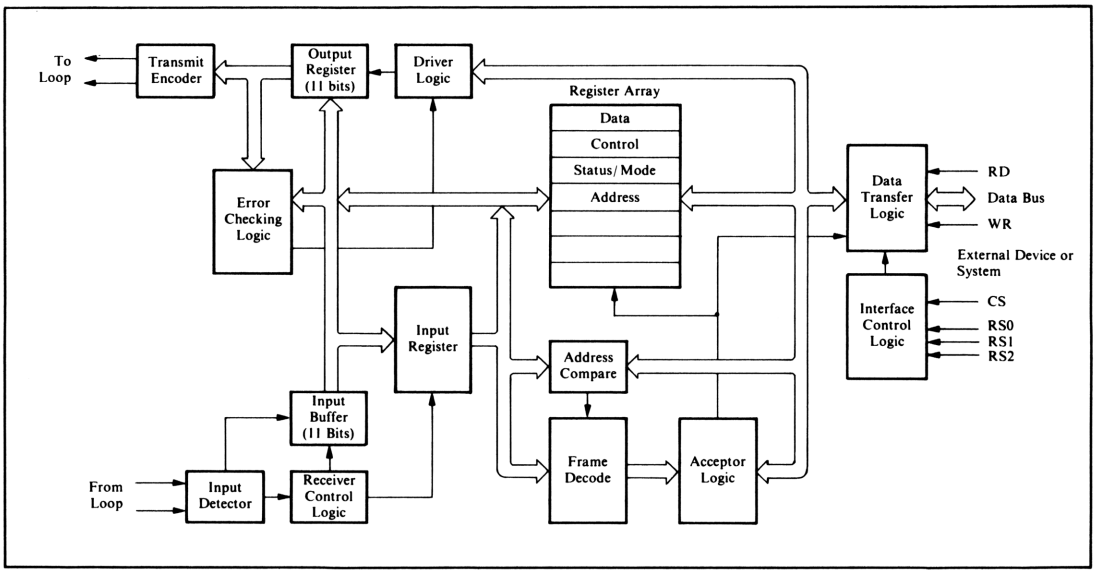


FIGURE 4-1. Block diagram of a component-level interface to HP-IL

When the sync bit has been detected, it and the bits that follow will be loaded into the 11-bit input buffer under control of the receiver logic. The use of the word "receiver" to describe this logic function is quite deliberate. Earlier chapters discussed such device roles as Talker, Listener, and Controller. These roles are actually just a few of what the HP-IL specification defines as "interface functions."

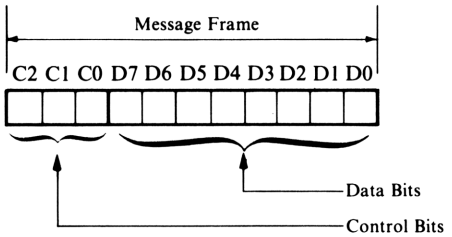
There are 17 different interface functions defined, and all of them will be described in detail in Chapter 6. At this point we simply want to introduce the concept, since these functions shape the characteristics of interfaces to the loop. The Receiver interface function is responsible for receiving messages from the loop. All devices on the loop obviously must have this capability regardless of whatever primary role (Talker, Listener, or Controller) they are playing. As you'll see later, however, many of the interface functions are optional.

After the entire 11 bits have been loaded into the input buffer, they are transferred to the input register by the receiver control logic. From there, the input message can be decoded and transferred around between the various resources of the chip.

Frame-Decoding Logic

As stated in preceding paragraphs, each message frame consists of 11 bits. The three most significant bits define the particular category of message; for example, command message, data message, and so on. The remaining eight bits are data bits which may further define the type of mes-

sage within one of the categories or which may simply contain a byte of ASCII data.



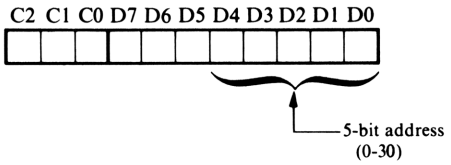
Message frames travel around the loop with the most significant bit first. Thus, after the first three bits (C2,C1,C0) have been received, it would be possible to determine the category of the message being received. In some cases, this might be used to speed up loop operations, since a message need not always be fully decoded before determining whether it might be passed on to the next device on the loop. If you want to take advantage of this fact, you can add some logic to pre-decode these first three bits of the message between the input buffer and input register in Figure 4-1.

Address-Compare Logic

Chapter 3 described two commands, Listen Address (LAD) and Talk Address (TAD), that included an address as part of the command. As you'll see later in this chapter, there are other commands that also include address information. When a message that includes an address is received by a device which has been assigned a different address, that device typically will take no action other than passing on the message.

Thus, Figure 4-1 shows some address-compare logic. This logic will compare the address contained in the message with its own address contained in an on-chip address register. If the two addresses match, the frame-decoding logic would be enabled. Otherwise, the message would simply be passed on along the loop.

When a message contains an address, the address is in the five least significant bits (the last five bits received) of the message frame.



Note that valid addresses are in the range of 0 to 30, not 0 to 31. An address of all 1's (31 decimal) is not permitted since this address is reserved for special commands (Unlisten, Untalk). This special usage will be discussed later, along with an "extended-addressing" capability that can be used to have as many as 961 devices on the loop.

Acceptor Logic

The acceptor logic shown in Figure 4-1 is the primary control logic for our chip. It receives the decoded message frame information and determines what action should be taken by the chip (for example, retransmit the message, pass data on to the external device, and so on). This logic would also control reading and writing of information into the chip's register array. The action that the acceptor logic directs the chip to perform would depend, of course, not only on the contents of received message frames, but also on the role that the chip and its attached device are playing at

the moment. So, the actions taken might be different if the chip were a Talker instead of a Listener.

Previous paragraphs introduced the concept of interface functions during the discussion of the receiver logic shown in Figure 4-1. The term "acceptor" is another that is deliberately chosen for discussion here because it is another of the interface functions defined by the HP-IL specification. Once again, the detailed discussion of the acceptor interface function will be deferred until Chapter 7.

Register Array

The register array block shown in Figure 4-1 is used to hold on-chip information such as device address, status or mode, control, and data. Additional "scratch" registers might be provided for other special functions or data manipulation. The data register would be used to hold data being transferred between the loop and the external device interfaced by the chip.

The control register might hold information needed by the chip to perform its various internal functions; for example, information about the message currently being processed or about the type of interface to the external device.

The status/mode register would contain information about the role of the chip and its attached device. That is, "am I an active Talker, the System Controller," and so on.

The address register would contain the address assigned to the chip and its attached device. This address typically would be loaded into the register by the attached external device and would be used by the address-compare logic to determine if addressed messages were intended for this device.

Data Transfer and Interface

These logic blocks implement the interface between the chip and the attached external device. Data transfers between the chip and the external device would typically proceed rather independently, or at least asynchronously from transfers between the loop and the chip. For example, when the loop requires information from the device, the chip would signal the device, obtain the data, and then put the information back out onto the loop.

Assume that the external device connected to the chip is some type of microcomputer. You might, therefore, also assume that the interface the chip provides would be a generalized "microcomputer-type" interface. While it is beyond the scope of this book to provide any sort of a detailed discussion of such an interface, here are a few general characteristics you would expect to encounter.

First, the chip would provide a bidirectional, eight-bit data path for transfer of information between the microcomputer and the chip. These transfers of information would be synchronized by the read (RD) and write (WR) strobe signals generated by the microcomputer.

Second, you might expect a microcomputer to treat the HP-IL interface chip simply as some type of input/output device. Therefore, we have provided a chip-select (CS) input to the chip to enable the microcomputer to distinguish the chip

from other devices that might also be connected to the microcomputer system. We have also provided three register-select inputs (RS0-RS2) to allow the microcomputer to transfer information between itself and a particular register within the chip's register array.

Driver Logic

This logic is responsible for transmitting messages out onto the loop. Once again, the nomenclature used here is quite intentional: "Driver" is one of the interface functions defined by the HP-IL specification. While most of the discussion of this and other interface functions will be provided in Chapter 7, let's look briefly at how the Driver interface function differs from the more primary function of Talker.

A Talker on the loop must be able to generate and transmit information out onto the loop. Therefore, if a device is assigned the role of Talker, it must also implement the Driver interface function. However, even if a device is not a Talker, it must still be able to at least retransmit messages it receives back out onto the loop to the next device. Therefore, all devices would have to include the Driver interface function.

At this point, perhaps the concept of interface functions as distinguished from the primary "roles" of loop devices is beginning to take shape. Interface functions are a way of conceptually dividing all the tasks that have to be accomplished by devices on the loop into various categories or functions. A particular device may implement some of the functions based on its own characteristics and needs, and is required to implement certain of the interface functions if it is to participate at all on the loop.

Output Register

This register holds the 11-bit message frame that is going to be sent out to the next device on the loop. This message might simply be the message that was just received from the preceding device on the loop and could be obtained directly from the chip's input register. Alternately, it might be coming from the microcomputer via the register array, or from the register array under control of the chip's own logic. The source of the outgoing message depends on what mode the device is operating in and/or on the preceding message that was received.

Error-Checking Logic

The error-checking logic included in Figure 4-1 is an intrinsic and important part of the HP-IL scheme. Since all devices that can source a message (that is, generate an original message and send it out on the loop) typically wait until that message returns before sourcing another, error-checking can be easily implemented. The HP-IL specification does not require that any device implement error-checking. It strongly advises, however, that all devices include this capability. This advice should usually be heeded. Since you usually have to wait for a message that you've sent out to return before you can do anything further, why not check to see if it returned exactly as you sent it?

Transmit Encoder Logic

This small section of logic simply converts the parallel message frame contained in the output register to the serial-bit stream at the voltage levels required by the loop.

MICROCOMPUTER AS CONTROLLER

Assume in this chapter that the device interfaced to the loop via the chip is a microcomputer. Further, assume that this microcomputer is going to perform as the Controller of the loop. Since HP-IL is intended for low-speed, low-power applications, you might expect that a relatively powerful device such as a microcomputer would be the Controller on the loop.

As Controller, the micro (for the sake of brevity, microcomputer will be abbreviated as "micro" for the rest of this chapter) will have a great deal more responsibility than did the thermal printer described in Chapter 3. The printer was primarily a Listener. The micro, on the other hand, must be able to assume all of the roles (Talker, Listener, and Controller). And, as Controller, its tasks require more intelligence. Let's discuss the various messages that the micro must handle.

COMMAND GROUP MESSAGES

As you saw in Chapter 3, the command group messages are used to establish initial operating conditions on the loop, change the operating modes and roles of devices on the loop, and generally maintain discipline and control of the loop system. Here are the command group messages that a micro might be expected to source in its role as Controller:

- Interface Clear (IFC)
- Device Clear (DCL)
- Loop Power-Down (LPD)
- Listen Address (LAD)
- Unlisten (UNL)
- Talk Address (TAD)
- Untalk (UNT)

NOTE: All of these messages were discussed in Chapter 3, but will be discussed again here as they relate to the micro in its role of Controller.

- Selected Device Clear (SDC)
- Go To Local (GTL)
- Remote Enable (REN)
- Not Remote Enable (NRE)
- Parallel Poll Enable (PPE)
- Parallel Poll Disable (PPD)
- Parallel Poll Unconfigure (PPU)

Interface Clear (IFC) Message

The IFC message can only be sourced by the System Controller. Assume that our micro not only can assume the role of Loop Controller, but also that it is the System Controller. As described in Chapter 3, the IFC message will typically be sent around the loop when the system is first powered up to put all the other devices on the loop into some initial idle state.

Step back for a moment and see how this initial IFC transaction might be handled between the micro and the interface chip.

When the system is first powered up, assume that the chip itself is in some sort of neutral or idle state: it is playing no active role on the loop. The micro would, therefore, write information into the chip's status register telling the chip that it (and its attached micro) were going to perform as System Controller for the loop.

Next, the micro would send data representing the IFC command to the chip and the chip would send the IFC message frame out on the loop. When the IFC message frame has returned from its trip around the loop, the chip would inform the micro of this fact and the micro would then send the RFC command message out onto the loop to ensure that all devices had completed the Interface Clear command. (Refer to Chapter 3 for a thorough discussion of the IFC and RFC commands.)

Selected Device Clear (SDC)

The Device Clear message was discussed in some detail in Chapter 3 and you should refer to that discussion for an explanation of how it might affect devices on the loop. Now, the Selected Device Clear (SDC) message can be introduced. The DCL message caused all devices (that were capable of responding to the message) to conduct their own particular clearing activities when the message was received. The SDC message differs slightly in that only devices which are currently playing the role of Listener on the loop will conduct the clearing activity.

This differentiating can be quite useful since there can be multiple Listeners on the loop at any one time, but only one Talker or Controller. Therefore, if the micro wants to clear one or more devices (but not all devices indiscriminately), it would send out the SDC command and accomplish clearing of all currently addressed Listeners. Note that there is another command which would be used if you simply want to deselect all Listeners—the UNL command described in Chapter 3. The SDC command would cause all Listeners to clear themselves, but not necessarily take themselves out of the Listener role. For example, SDC might cause a printer to perform a top-of-form and a cassette drive to go to the beginning of the tape.

Remote/Local Commands

Some devices have front panel knobs and switches that can be used to control their operation; a voltmeter is a good example of this kind of device. Sometimes it is useful to operate these devices manually from their "local" controls, and other times it is handy to operate them automatically over the loop in "remote" mode. Three commands permit the Controller to switch devices back and forth between these two modes: Remote Enable (REN), Not Remote Enable (NRE), and Go To Local (GTL). Most devices have no local controls, or their local controls are always active (such as with a printer or a cassette tape drive). These devices simply ignore these commands and pass them on to the next device on the loop.

When the voltmeter, for example, is first powered on, it is in local mode. It communicates normally on the loop, but the

instrument settings (voltage range, for example) are controlled by the voltmeter's front panel controls and cannot be changed by programming data from the loop.

One of the first things the Loop Controller usually does is send the Remote Enable (REN) command. By itself, REN really doesn't do much of anything. Devices still respond to their local controls. But now, the entire loop is in a state that allows switching from local to remote and back again as necessary.

Now, any time after the voltmeter receives the REN command and then receives its Listen Address (LAD) command, it will immediately switch to remote mode. This makes sense, because the device must become the Listener anyway, in order to receive the proper knob settings from the loop. This also allows the Controller to select the device or group of devices it wants to control remotely, while leaving the others in local mode. Once in remote, the device will remain in this mode even though the Controller later sends the Unlisten (UNL) command and then makes some other device a Listener.

When the Controller wants to put the voltmeter back into manual mode, it sends the Go To Local (GTL) command. This command is similar to the SDC command described earlier in that only those devices currently addressed as Listeners pay any attention to it. Once again, this permits the Loop Controller to choose which device(s) it wants returned to local mode.

If the Controller wants all devices returned to local, it can send the Not Remote Enable (NRE) command. This not only puts all devices into local mode at the same time, but also takes the loop out of Remote Enabled state. Now all devices will stay in the local mode (even though they receive their LAD command) until they get the REN-LAD sequence once again.

PARALLEL POLLING

There are three command messages in this group and, since they are all interrelated, they will be discussed together.

Chapter 3 discussed the Send Status (SST) message and mentioned that it might also be described as "serial polling," since the Controller could use that message to discover the status and needs of any particular device on the loop. Obviously, this serial-polling procedure could take quite some time if there are many devices on the loop. Therefore, the specifiers of HP-IL have provided a more time-efficient method of discovering the needs of devices on the loop: parallel polling.

The SST message not only is addressed to a single device on the loop (serial polling), but it can also be responded to only by devices that are currently addressed as active Talkers on the loop. The parallel-poll capabilities that can be implemented by devices on the loop allow the Controller to quickly discover the needs of all devices (that is, all devices that implement this function) on the loop. So, you don't have to first address a device as Talker and then ask it for its status.

Parallel Poll Enable (PPE)

This command is sent by the Controller to a specific device on the loop to activate that

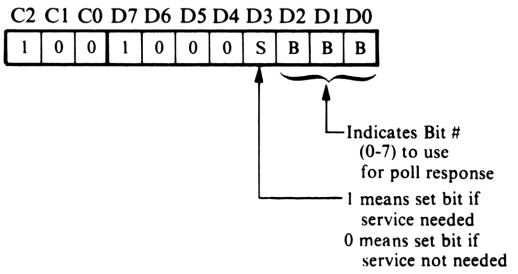
device's parallel-poll response capability. Though a device has been designed with the parallel-poll response capability included, it must specifically be enabled by the Controller before this capability is utilized. This allows the Controller complete flexibility in configuring and controlling the loop. For example, certain polling operations might apply to only certain device types. In this case, the Controller would only want responses from devices of that particular type: responses from other devices at that time would decrease the efficiency of the polling operation.

Remember, the PPE message is directed to a specific device on the loop. The message does not contain the address of the device to be enabled. Instead, the PPE message is always assumed to be directed to the active Listener on the loop. You will recall from Chapter 3 that there can be more than one active Listener on the loop. Therefore, you would usually want to send the Unlisten (UNL) message out first to deactivate all Listeners. Then you would send the Listen Address (LAD) message out to the device that is going to be enabled for parallel polling. Finally, you would send the PPE message around the loop to actually enable that device's poll-response capability. The sequence of these messages can be illustrated as follows:

- UNL – Unlisten message to deactivate all Listeners.
- RFC – Ready For Command ensures that the UNL command has been completed.
- LAD2 – Device 2 addressed as active Listener.
- RFC – Ensures that the LAD command has been completed.
- PPE – Causes Device 2 to enable its poll-response capability.
- RFC – Ensures that the PPE command has been completed.

This entire sequence, including the UNL message, would have to be repeated for every device on the loop that is to have its poll-response capability enabled. This sequence may seem very cumbersome, and you may be asking yourself if it wouldn't be easier to just perform serial polling of the individual devices on the loop. Remember, however, that this sequence is the configuration portion required to set up the devices for parallel polling. You might only have to perform this sequence once at the time you power up the system or, at most, at quite infrequent intervals during system operations. The actual polling operation is, as you'll soon see, quickly and easily accomplished.

While the PPE message does not contain the address of the device that is to be enabled, it does contain some configuration information that determines how the device will subsequently respond to a parallel poll. Here is the format of the PPE message:



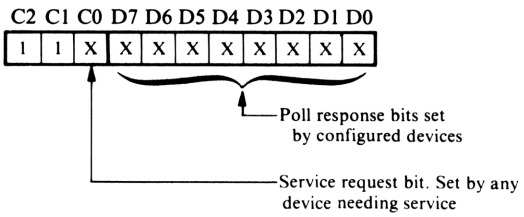
The seven most significant bits contain the bit pattern for the PPE message. The four least significant bits tell the device that is being enabled for parallel polling how it should respond to subsequent polling operations. Bit D3 (the "S-bit") defines the sense or polarity of the device's response to a poll. If the S-bit is "1," the device must respond to a poll by setting the appropriate bit in the poll word if it needs service. If the S-bit is "0," the device responds to a poll by setting the appropriate bit in the poll word if it does not need service.

The three least significant bits in the PPE message define which bit in the poll word the device is supposed to set during its response to polling. For example, if these three bits are zero (000), the device would set the least significant bit in the poll word and if these three bits are one (111) the device would use the most significant bit of the poll word (bit D7) to respond.

The last several paragraphs have referred to the poll word that the device manipulates during its poll response. In actuality this poll word is part of the Identify (IDY) message.

Identify (IDY)

The Identify message is sent out by the Controller to determine if devices on the loop need service.



Any device requiring service from the Controller must set the service-request bit (C0). Note that this bit can be set by devices that do not have Parallel Polling Enabled. The IDY message is also used in serial-polling operations. When the IDY message completes its circuit of the loop and returns to the Controller, bit C0 can be checked to see if *any* device is requesting service. If C0 is set, it indicates that somebody out there needs service. If there are devices on the loop that have Parallel Polling Enabled, they would not only set bit C0, they would also set their assigned bit in the data portion (D7-D0) of the IDY message. Thus, the contents of the IDY data byte can be checked when the message has been returned to the Controller to determine which of the devices capable of responding to parallel polling has requested service.

Note that it is quite possible that more than one device at a time could respond to the parallel poll. Each device requiring service uses its own specific bit to respond.

You should also note that it is not one of the inherent functions of the Controller to decode the IDY message when it returns in order to determine which device or devices require service. Any decoding of the IDY message, and the subsequent course of action to service devices, would depend on the system you have designed. For example, you would expect the interface chip to decode the fact that the service-request bit is set when the IDY message returns. The chip would then pass this information along with the IDY data byte to the micro. It would then be up to the micro to decode the data byte, determine who needs service,

and then initiate the appropriate loop operations to provide that service.

Parallel Poll Disable (PPD) And Unconfigure (PPU)

These two messages terminate the parallel-polling function that was set up by the Parallel Poll Enable (PPE) message. The PPD message causes devices which are currently addressed as Listeners to no longer respond to the parallel polls. If you subsequently want to reenable one of these devices, you must send it the PPE message. However, devices that are not currently addressed as Listeners, but which have been enabled for parallel polling, are not affected by the PPD message and retain their ability to respond to polls.

The Parallel Poll Unconfigure (PPU) message is used by the Controller to disable all devices on the loop from responding to parallel polls. When a device receives this command, regardless of whether it is currently addressed, it disables its parallel-poll response capability. Thus, the PPU message can be viewed as a system clear or reset message for parallel polling, while the PPD message is more like a Device Clear operation.

READY GROUP MESSAGES

The ready group messages coordinate and synchronize the transfer of messages and data between devices on the loop. As noted in Chapter 3, the ready group messages can be sourced by devices other than the Controller. In the discussions that follow, however, you can usually assume that it is our micro/interface chip combination that is generating the messages.

Here are the ready messages that our micro might source:

- Ready For Command (RFC)
- Send Status (SST)
- End of Transmission, OK (ETO)
- End of Transmission, Error (ETE)

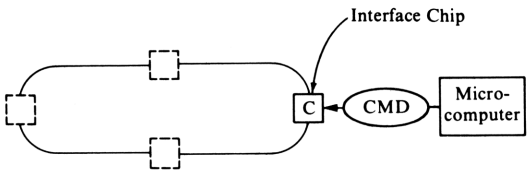
NOTE: The above messages were discussed in Chapter 3, but will be discussed again here as they relate to the micro in its role as Controller.

- Send Data (SDA)
- Not Ready for Data (NRD)
- Send Device ID (SDI)
- Take Control (TCT)

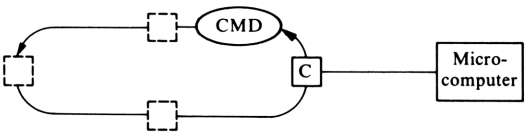
Ready For Command (RFC)

Chapter 3 described how the RFC message is used by the Loop Controller to determine when an immediately preceding command message has been executed. The command message-RFC sequence will occur quite frequently, since HP-IL protocol demands that every command message be followed by the RFC message—this is part of the Loop Handshake procedure. Because this sequence will always be required, we would not expect our micro to have to deal with it. Instead, we would expect the interface chip to automatically send out the RFC message when it has received back a command message that it has previously sent out to the loop. The interaction between the chip and the micro would thus be as follows: The micro would place the chip in the Controller mode (if it were not already there)

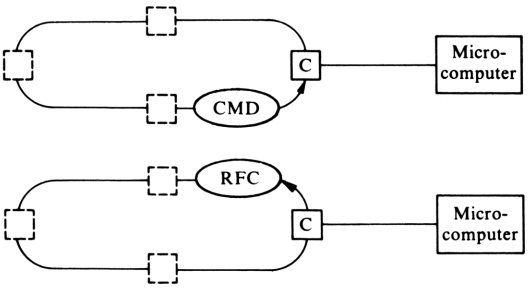
and would then send the chip the command message (CMD in our illustration) that is to be sent out on the loop.



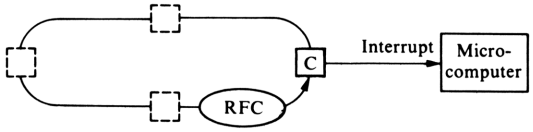
The chip would then proceed to send the command message out to the loop without further intervention or attention from the micro.



When the command message (CMD) returns to the chip, the chip would automatically generate the RFC message and send it out on the loop. Once again, this would be done autonomously by the chip without any need of supervision by the micro.



When the RFC message finally returns to the chip, the chip informs the micro (using an interrupt input to the micro or some other predefined method) that the previously issued command had been successfully completed. The micro could then proceed with the next desired loop operation.



Send Status (SST)

Remember that this message is used by the Controller to obtain one or more bytes of device-oriented status information from the device that is the active Talker on the loop. When the active Talker receives the SST message, it does not retransmit this message, but instead sends one or more data bytes (DABs) back to the Controller.

Note that the interpretation of these data bytes being returned in response to the SST message is entirely up to the system designer. Therefore, you would not expect our interface chip to perform any decoding or analysis of these data bytes. Instead, it would simply pass them on to the micro for processing.

End of Transmission (ETO, ETE)

Chapter 3 described how these messages are always sourced by the active Talker and directed to the Controller. The ETO message indicates that the Talker correctly received back all of the data bytes it sent. If the chip received this message from the Talker, it would simply indicate that the transmission was complete and the next loop operation could therefore proceed.

If the chip received the ETE message

indicating that there were errors detected by the Talker in the returned data bytes, the chip would report this error status to the micro. The action taken at this point would be entirely up to the micro. The chip would likely take no other action beyond reporting the error. Typically, the micro would initiate some recovery procedure such as retrying the transmission or, if that failed, requesting operator intervention.

Send Data (SDA)

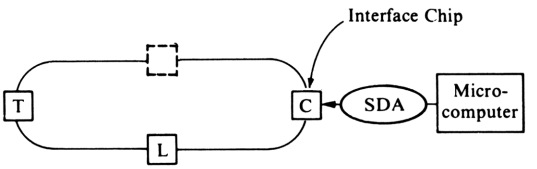
This message is sent by the Controller to the active Talker and causes the Talker to begin transmitting data bytes (DABs) on the loop. Since there can only be one active Talker on the loop at any one time, the SDA message does not contain any address information. It is always directed to the one device on the loop currently acting in the role of Talker.

When the Talker receives the SDA message, it does not pass the message on around the loop back to the Controller. Instead, it substitutes its first data byte (DAB) for the SDA message and thus begins its transmission of data. After the Talker has sent its last byte of data it sends the appropriate End of Transmission message (ETO or ETE) to inform the Loop Controller that another loop operation can begin.

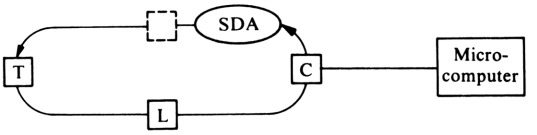
The data being sent around the loop by the Talker is passed on by all devices until it has returned to the Talker, which then replaces that byte with the next byte. The data is not addressed to any particular device; it is directed to all devices on the loop currently addressed as Listeners. Therefore, it is up to the Controller to ensure that only devices for whom the data is intended are active Listeners when the SDA message is sent out.

Here's a sample of a possible sequence involving the SDA message:

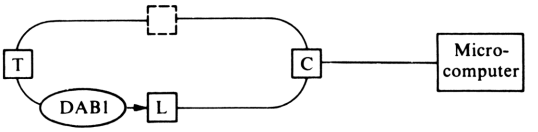
The micro sends the SDA message to the chip, having previously set up the chip as Controller and having defined an active Talker and active Listener on the loop.



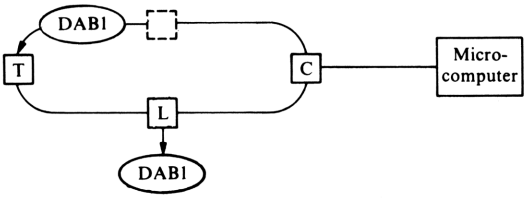
The chip then transmits the SDA message out onto the loop and the message is by definition directed to the Talker.



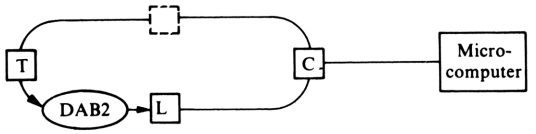
When the Talker receives the SDA message, it does not pass it on, but instead substitutes its first data byte (DAB1).



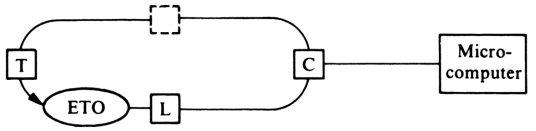
The active Listener(s) takes a copy of the data byte to do with it whatever is appropriate for that device. For example, if it is a printer, it would put the message into the print buffer. The data byte is also passed on around the loop back to the Talker.



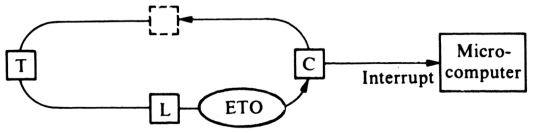
When the Talker gets the data byte (DAB1) back from its trip around the loop, it can then send out its next byte of data (DAB2). Note, the Talker should also be checking each data byte for errors as it returns.



After the Talker has received its last byte of data back, it sends the ETO message (or ETE if there were errors) around the loop to the Controller.



When the Controller (our interface chip) receives the ETO message from the Talker, it would notify the micro (perhaps using an interrupt input) that the transmission was complete and that another loop operation could, therefore, be started. Note that the ETO message does not change the role of any of the devices on the loop. The Talker, Listener, and Controller all maintain their assigned roles until they are told otherwise.



There is one other point worth noting regarding to the transaction just described.

The interface chip should be able to assume more than one role at a time. For example, it could be the Controller and also a Listener. If this were the case in the preceding illustrations, then the interface chip, in addition to passing the data bytes it gets from the Talker on around the loop, might also pass each of the data bytes on to the micro. For example, if the Talker was a tape cassette and was sending data to a Listener that was a printer, there might be times when the micro also wanted to examine that data while it was available on the loop. In this case, the micro would simply address both the printer and the interface chip as Listeners before issuing the SDA message.

Not Ready For Data (NRD)

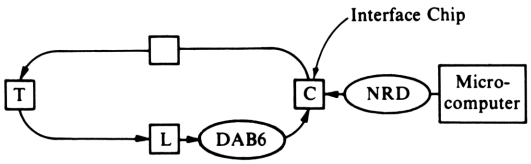
The preceding description of the Send Data (SDA) message showed that once the Talker had received that message, it began sending data and continued to completion signaled when the Talker sent the ETO or ETE message. However, in any system involving multiple devices, there often will be situations where you want the system (the loop) to be able to interrupt processes that are in progress for other activities of higher priority. This is precisely the capability that the Not Ready for Data (NRD) message provides for the loop.

The NRD message allows the Controller to interrupt a Talker-to-Listener transmission so that it can use the loop for some other operation. As is the case with the Send Data message, the NRD message

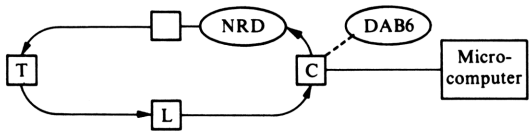
contains no address information, since it is always directed to the currently active Talker (and there can be only one active Talker on the loop at a time).

Here's a typical sequence involving the NRD message to see how it is used.

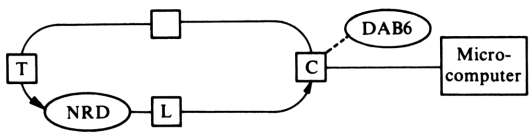
Let's assume that the Talker is sending its sixth byte of data (DAB6) around the loop when the micro decides that it needs the loop for some other operation and sends the NRD message to the interface chip.



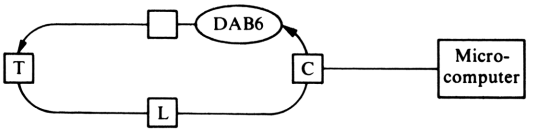
When the chip subsequently receives the data byte, it retains the data (in an internal register) and instead sends out the NRD message on the loop.



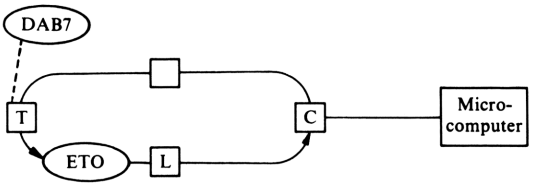
When the Talker receives the NRD message from the Controller instead of the expected data byte, it must note that fact and then pass the NRD message on around the loop.



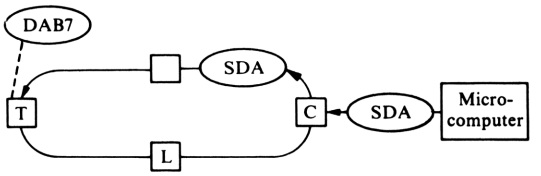
When the chip receives the NRD message back from the loop, it sends the data byte (DAB6) that it has been hanging on to back out on the loop to the Talker.



The Talker, having taken note of the NRD message on the preceding go around, now knows that it has received its preceding data back and must terminate the data transmission for now. It then sends out the ETO (or ETE, if appropriate) message. If there is data still left to send (for example, DAB7,8, and so on) the Talker is expected to retain that data for later transmission.

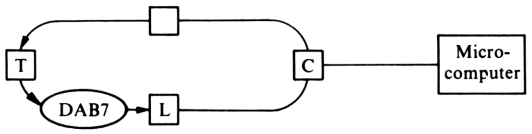


After the micro has finished using the loop for the operation that was of higher priority, it can subsequently cause the interrupted transmission to be resumed at the point where it was interrupted. The micro would issue another SDA (Send Data) message to the interface chip which would then send the message out onto the loop.



When the Talker receives this new SDA

message, it should resume its interrupted data transmission where it had left off by sending out the next data byte (DAB7) on the loop. Usually, if the Controller had changed the role of the Talker while it was using the loop for another operation, the Talker is expected to be able to resume a previous transaction (and a former role) where it had been interrupted. But responsibility for role assignments and continuity during interruption belongs to the Controller.



Send Device ID (SDI)

This command is used by the Controller to obtain detailed information from the currently active Talker about its identity and characteristics. It is thus similar to the Send Status (SST) command, described earlier in this chapter and in Chapter 3.

Once again, no address information is needed as part of the SDI message, since it is always directed to the sole active Talker on the loop.

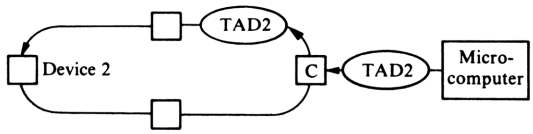
The Talker handles this message in the same way it handles the SST message. It does not pass the message on around the loop, but instead substitutes data bytes which are directed back to the Controller. When the Talker has sent all of the identification data bytes (and received them back from their circuit of the loop) it must send the ETO (or ETE if there were errors) to inform the Controller that the transmission is complete.

Just as with the Send Status message, interpretation of the identification information is entirely up to the designer, but usually includes the manufacturer's initials and model number of the device (for example, HP3468A).

Take Control (TCT)

This message allows the active Controller to designate another device as Controller and pass control of the loop to that device. The device to which control is being passed must first be addressed as the active Talker. When the Talker subsequently receives the TCT message, it assumes the role of Controller. The sequence involved in transferring control is illustrated as follows.

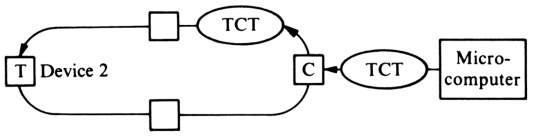
First, the micro sends the Talk Address (TAD) message to the interface chip which then sends the message out onto the loop. In this illustration, we have shown the TAD message being sent to Device 2 on the loop. Of course, if Device 2 were already the active Talker, this step would be unnecessary.



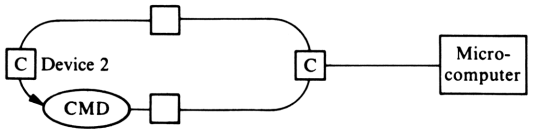
The Controller must, of course, follow the TAD message with an RFC message. But since this step is always required, it is not necessary to illustrate it.

After the RFC message has returned to the Controller, the TCT message can be sent. The TCT message does not contain any address information, since it is always directed to the active Talker and there can

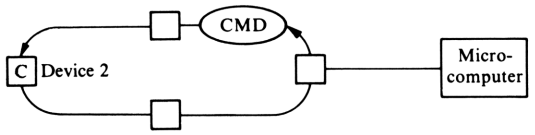
be but one Talker on the loop at any one time.



As soon as the Talker receives the TCT message, it immediately assumes the role of Controller. It does not pass the TCT message on around the loop, but instead replaces it with its first operation message (CMD in this illustration).



The original Controller, our interface chip, retains its role of Controller until it receives a message. If the first message it receives is anything other than the TCT message, then it knows that there is a new Controller and it abandons that role.

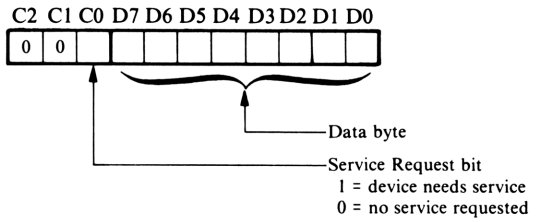


If the TCT message returns to the original Controller, it indicates that the intended new Controller could not assume that role for some reason. In this case, the original Controller retains that role.

SERVICE-REQUEST DATA MESSAGES

Two types of data messages (DAB and END) were described in Chapter 3. There is one aspect of these messages that was not discussed, however: that is the ability to send a service-request to the Controller using the data messages. The mechanism that is used is essentially the same as described earlier in this chapter in the section on parallel polling and the related messages.

To see how this mechanism works with data messages, examine the bit pattern for these messages.



The eight least significant bits contain the actual data that is to be transferred. The two most significant control bits (C2, C1) indicate that this is a Data Byte message. Bit C0 is the service-request bit. If this bit is set to "1" it indicates that some device on the loop requires attention from the Controller.

The Controller monitors every Data Byte message that transits the loop. If it detects one that has the service-request bit set, it knows that something needs service. However, it does *not* know which device requires service. Although Data Byte messages are generated by the active Talker on the loop, any device on the loop can set the service-request bit in the Data Byte message as that message passes by that device on the loop.

This approach provides a rather efficient

method for devices on the loop to interrupt operations in progress if the system demands this. Devices need not wait until nothing is happening on the loop before attempting to attract the Controller's attention. All they have to is set the service-request bit the next time a Data Byte message comes by.

Also recognize that the action the Controller takes when it discovers that some device is requesting service is not defined by the HP-IL specification. This reaction is up to the system designer. Typically, you would interrupt the data-transfer operation in progress (unless it were of very high priority or of a critical nature) and conduct some sort of survey of the devices on the loop. This survey might consist of a serial- or parallel-polling operation, depending on the system configuration.

AUTO-ADDRESS MESSAGES

Devices have specific loop addresses and some messages are directed to devices using an address. Chapter 3 suggested that these addresses might be assigned using some sort of switches on the device or that the addresses might be programmable.

As it turns out, HP-IL provides a mechanism that allows you (the Controller) to assign and reassign addresses to devices on the loop. Before describing how this is done, however, we must point out that devices on the loop are not required to have the capability of having addresses programmed under loop control. You can design a device that has a predefined, "hardwired" address and it can also be used on the loop. However, the loop is much more flexible if addresses can be dynamically assigned and reassigned.

There are two auto-addressing techniques that can be used on the loop: the primary auto-addressing mode allows up to 31 devices to operate on the loop while the secondary, or extended mode, allows as many as 961 devices to be connected to the loop.

Primary Addressing Mode

Two messages are used in this addressing mode to automatically assign addresses to loop devices and to reconfigure the loop as required. The Auto-Address (AAD) message causes devices that have this capability implemented to accept a new device address and then pass a new address on to the next device on the loop. The approach is quite simple and straightforward.

The first device to receive the AAD message assumes the address of "1," increments that address (contained within the AAD message) by 1 and passes the message on to the next device. Thus, the loop devices will be numbered sequentially from 1 to a maximum of 30 with the first device on the loop (in relation to the Controller) being Device 1 and the last device on the loop, before you get back to Controller, having the highest address.

When the Controller needs to reassign addresses, it sends out the Auto-Address Unconfigure (AAU) message to prepare all the devices for new address assignments.

Secondary (Extended) Addressing Mode

This mode of addressing lets you assign addresses to as many as 961 loop devices by using a two-byte address. In order to activate this mode, the Controller uses some other address-assignment messages similar to AAD. Each device receives not only a primary address which can range from 0 to 30, but also a secondary address which can range from 0 to 30. In order to make one of these devices the active Listener or Talker, the Controller sends the proper primary address (TAD or LAD) and then follows it with the correct secondary address, called SAD. Not many systems will need this capability so it will not be discussed in detail here.

Note that you can have some devices on the loop capable of accepting secondary addressing while others will only implement the primary addressing mode or perhaps not accept auto-addressing at all. Devices that do not implement a particular level of auto-addressing simply pass on those related messages to the next device on the loop without changing the message in any way.

ASYNCHRONOUS OPERATIONS

All of the operations described in this and the preceding chapters have been synchronous in the sense that every operation is conducted singularly—there are never two competing or conflicting operations happening on the loop. You have seen that there can be orderly terminations of an ongoing operation such as a data transfer to allow for servicing of another device's more immediate needs. But these terminations or interruptions were conducted within the synchronized flow of ongoing operations by such techniques as setting service request bits or inserting Not Ready for Data (NRD) messages within the flow.

These techniques for altering ongoing operations on the loop all adhere to one rule: there is never more than one message in transit around the loop at any given time. There may be cases, however, when the techniques described so far may be inadequate. For example, if the device sending data messages around the loop is extremely slow, and there is a device on the loop which requires rapid response to its service-requests, then an alternate technique might be needed.

Once again, the designers of HP-IL have provided a solution to this potentially vexing quandary. They will allow multiple messages to travel around the loop, but only under special circumstances. The mode of operation used during these special circumstances is defined as "asynchronous" operation.

Now that the subject has been brought up, note that implementation of this asynchronous mode should not be considered unless absolutely required by your system, and then only with great caution, since it can greatly complicate the logic required within system devices. Additionally, this mode of operation can endanger what is otherwise an extremely data-safe system. It becomes much more difficult to

ensure the integrity of data during asynchronous operations. As with secondary addressing, very few systems will have need of this feature, so it will not be detailed.

CHAPTER 5

Some Typical Loop Sequences

Chapters 3 and 4 described several messages that might be observed during loop operations. This chapter will discuss and illustrate a number of loop sequences as a way of pulling together the information presented about HP-IL thus far. Since we have already described nearly all of the messages that are used on the loop, the nomenclature should be familiar to you. If the function or use of a particular message is confusing to you, refer back to Chapter 3 or 4 where it was first presented, and to the Glossary, which describes all of the HP-IL messages.

Obviously, we cannot describe all possible message sequences that might occur on the loop. Furthermore, the sequences that are described are not the only way, nor necessarily the best way, that a particular task can be accomplished. Instead, the sequences presented here are intended to give you a better feel for how the messages are used and how devices on the loop interact. No doubt, you will be able to come up with more efficient and creative solutions to these tasks as you become more familiar with HP-IL.

A POWER-ON SEQUENCE

In almost any system you might design, you will need to establish initial operating conditions when power is first applied to the system. Typical tasks that must be accomplished upon first powering up an HP-IL system include establishing which device is the Controller, determining when all other devices on the loop are ready for operation, and initializing devices that are on the loop.

Here is an example of a power-on sequence:

IFC
IFC The System Controller initially sends the Interface Clear (IFC) message out to the loop at regular, slow intervals. The Controller continues sending the IFC message until it receives the IFC message back from its transit around the loop. When the IFC message is returned to the Controller, it indicates that all devices on the loop have been powered-up, are properly receiving messages, and are in the process of initializing themselves.

RFC After the IFC message has returned, the System Controller sends out the Ready For Command (RFC) message. This message is not passed on by devices on the loop until they have actually finished their initialization operations and are ready to receive the next command from the Controller. Note that the RFC command would only be sent once, while the IFC command message would be sent continuously until it was returned to the Controller.

Once the RFC message has returned to the Controller, other initialization procedures that might be device-specific can be accomplished using standard loop protocol.

INITIALIZING A SYSTEM WITHOUT CONTROLLER

HP-IL allows you to have a system that has no Controller. This is the ultimate "friendly" system where the devices on the loop know exactly what to expect from, and what they must provide to, other devices on the loop. Design of such "friendly" devices may make sense in special applications; a printer simply logging readings from a voltmeter, for example.

Let's look at how initialization of such a system might differ from the one just described.

If there is no Controller on the loop, then there must be one device that initially starts out as the Talker. In this situation, the Talker—while it needn't perform the tasks that a loop or System Controller does—must initiate activity on the loop.

DAB1 First, the Talker must send its first data byte out on the loop. This device must continue sending this same data byte out onto the loop at slow, regular intervals until it is returned. When the first data byte is returned, it indicates that the other devices on the loop are powered-up and are properly receiving and retransmitting messages.

This Data Byte (DAB) message is thus the equivalent of both the IFC and RFC messages in a more typical system.

DAB2
DAB3 Once the Talker on this loop gets back its first Data Byte message, it proceeds to send

DABn the rest of its data string at its normal speed. When the last Data Byte message has been sent and the transmission has been completed, the Talker can begin sending the next message string.

In this case, no EOT message is required because devices in such a friendly loop know that there is only a Talker and Listeners on the loop. Loop protocol and discipline is therefore greatly simplified. Transmission of these messages can continue indefinitely in this relatively undisciplined manner until or unless system demands require that devices be capable of responding to messages beyond those in the repertoire of simple Talkers and Listeners.

A DATA-TRANSFER SEQUENCE

Transfers of data around the loop will typically be the most common task that is performed in an HP-IL system. Once system roles and discipline have been established, you want to start moving information. Here is one possible sequence that might be used to move some data around the loop:

UNL First, if necessary, the Controller will send out the Unlisten (UNL) message to inhibit any other devices that might have been addressed as Listeners (and that should not receive this data transmission.)

RFC The UNL message is followed, as are all command messages, by the Ready for Command message.

LAD1 The Controller then will send out the appropriate Listen

RFC Address (LAD) messages to enable a device in the role of

LAD2 Listener on the loop. The Controller can set up more

RFC than one device as listener. Each device must be enabled by a separate LAD message, however, and each LAD message must be followed by an RFC (Ready For Command) message.

After the appropriate devices on the loop have been enabled as Listeners, the Controller should send out the message to the device that is to put data out onto the loop—the Talker.

TAD The Controller sends out the Talk Address message to en-

RFC able one device on the loop to send some data out onto the loop directed toward the Listener(s).

SDA The Controller, when it receives back the RFC mes-

DAB1 sage, will send out the Send Data (SDA) message. When

DAB2 the device that has been addressed as Talker receives

DAB3 the SDA message, it replaces that message with its own first byte of data (DAB1).

DABn After the Talker gets its last data byte back from its transit

ETO around the loop, it must send

the End of Transmission (ETO) message out to indicate that all of the data that it sent out, returned, and error-checked correctly.

When the Controller receives the ETO message back at the end of the data transmission by the Talker, it replaces the ETO message with the next loop operation.

Notice that completion of this data transmission has not, of itself, changed the roles of any of the devices on the loop. The device that was sending data out to Listeners on the loop is still the Talker and receivers of the data are still addressed as the Listeners.

If any of the roles of the loop devices are to change, the Controller must send out some messages reassigning roles after it receives the ETO message. (Remember, the Controller does not retransmit the End of Transmission message—regardless of whether it is an ETO or ETE message. It replaces this message with its next loop operation message.)

INTERRUPTION OF DATA TRANSMISSION

This subject was mentioned in Chapter 4 during the discussion on the need for the Loop Controller to be able to jump in if a slow-moving device were monopolizing the loop, when another device had more urgent need for access to the loop. Although HP-IL has intentionally been designed for use in relatively low-speed systems, there is no reason why provisions cannot be provided to maximize the performance that can be obtained within these constraints.

Interruption of data transmissions is one way of squeezing as much performance out of the loop as you possibly can (so far as simple data throughput is concerned). Interruption of a data transmission that is in progress between a loop Talker and one or more loop Listeners requires the intervention of the Loop Controller. Here's one way of doing it:

DAB5 Let's assume that the Talker has just sent its fifth data byte out onto the loop. At this point, the Controller decides that it has more important tasks to perform and hangs onto the Talker's fifth data byte (DAB5).

NRD The Controller replaces the fifth data byte (DAB5) with the Not Ready for Data (NRD) message. When the active Talker on the loop receives this message back, instead of the data byte that it sent out, it curtails its data transmission and passes on the NRD message to the Controller. This response places no small amount of responsibility on the Talker. It must recognize that the NRD message indicates that it must suspend transmission of its data. When the Controller gets back the NRD, it then passes on the data byte which was delayed.

DAB5 Not only that, the Talker must remember that when it finally

ETO does get its "lost" data byte back, it now must insert an ETO message onto the loop.

CMD When the Controller receives the ETO message from the Talker, it can then begin the operation for which it interrupted the data transmission by sending out its next command message (CMD).

If the Controller subsequently sends out the Send Data (SDA) message after having readdressed the Talker that was active when the previous data transmission was interrupted, the interrupted data transmission resumes from the point where it was interrupted. Thus, in our example here, if the Talker is still addressed as the active Talker when it receives the SDA message again, it should respond by placing its next data byte (DAB6) out onto the loop.

A SERIAL-POLLING OPERATION

Serial polling is used by the Loop Controller to discover the identity of a device (or devices) on the loop that have signalled they need attention by setting the service-request bit in a message traveling around the loop. As discussed earlier, a device on the loop can indicate its need for attention by setting the service-request bit in any Data Byte (DAB) message that is traveling around the loop—regardless of who initiated the transmission of that Data Byte message.

When the Controller sees a message on the loop that has the service-request bit set, it may choose to respond immediately (perhaps by using the NRD sequence described previously to interrupt the current loop transmission) or it may simply wait patiently until the present loop operation terminates normally. In either case, when the Controller gains control of the loop, it might use a sequence like this to find out who requested service and why:

UNL First, the Controller should send out the Unlisten command to prevent all the Listeners on the loop from automatically receiving the status bytes that are going to be soon sent out on the loop from polled devices. (This is, of course, followed by the RFC message.)

RFC

TAD1 The Controller addresses the first device on the loop to talk. (Of course, the Controller may know that only certain devices on the loop are capable of requesting service. If this is the case, there is no need for the Controller to poll every device on the loop—it can simply check those devices that are capable of requesting service.)

RFC

SST The Controller transmits the Send Status (SST) message which the addressed Talker replaces on the loop with its status byte or bytes (DABs).

DAB

ETO After the Talker has received back its last status byte, it must send out the ETO message. The Controller will rec-

TAD2

RFC ognize this as the end of the status-oriented transmission and replace ETO with a Talk Address message (TAD2 in our example) directed to the next device on the loop that is capable of requesting service.

SST This same sequence of addressing devices as Talkers
DAB and then requesting that they send status will be repeated
ETO for all of the devices on the loop. The Controller must examine the status bytes as they are returned from each of the polled devices to determine which device(s) actually require service.

If a particular device simply returns the SST message to the Controller instead of a data byte, the Controller knows that the particular device does not implement the serial poll function (and therefore cannot be the device requesting service). It's easy to see that serial poll is very similar to data transmission. The main difference is that the Controller receives the "data" (status bytes) without actually being addressed as a Listener.

PARALLEL POLLING AND CONFIGURATION

As explained in the preceding paragraphs, serially polling all devices on the loop, or even just those devices on the loop capable of responding to the serial poll, can require quite a bit of time. If your system requires a more time-efficient method of polling, you can implement the parallel-polling function defined by the HP-IL specification.

Of course, the gains that you obtain by implementing the parallel-polling capability in devices on the loop are not free. They require that devices on the loop have additional logic or intelligence (also known as $$$).

Let's take a look at how the parallel polling works so that you can get a feeling for how it might complicate the logic needed by a device on the loop.

UNL First of all, the Controller may want to send out the
RFC Unlisten (UNL) command. This prevents unwanted devices from responding to the subsequent parallel-polling
PPU sequence. Alternately, or perhaps additionally, the Controller may send out the Parallel Poll Unconfigure (PPU) message. This would reset any parallel-polling configuration that was previously set up on the loop.

Now, the Controller can begin to configure the loop for parallel polling. This consists of individually addressing each of the devices on the loop that the Controller wants to be able to respond to a parallel poll.

LAD1 First, the Controller addresses the desired device
RFC (Device 1 in this example) as a Listener so that the device can subsequently receive the

Parallel Poll Enable (PPE) command.

PPE13
RFC

Then, the Controller sends the Parallel Poll Enable command to the addressed Listener. The PPE message includes information indicating the bit that device should use in subsequent poll responses and the polarity of the response (1 or 0). In this example, the addressed Listener (Device 1) is expected to use Bit 3 for poll response and should set that bit to a "1" if it requires service.

UNL
RFC

Now the Controller must unaddress Device 1 as a Listener so that it can individually send polling information to another device on the loop.

This sequence (LAD, PPE, UNL) must be repeated for each unit on the loop that is to be enabled for parallel polling. After all the desired devices have been enabled, the loop can be considered as "configured" and normal loop operations can continue. While this procedure may seem lengthy, you should keep in mind that it probably will not need to be performed frequently. Furthermore, once this has been accomplished, the actual polling operation is greatly simplified.

IDY

The Controller executes the parallel poll by sending out the Identify (IDY) message. When this message is received by devices that have had their parallel-poll capability enabled, they set the appropriate bit in the IDY message according to the polarity programmed during configuration. They then pass the IDY message on around the loop. Devices that do not have the parallel-poll capability, or that have not been enabled, simply pass the IDY message on without altering it in any way.

When the Controller gets the IDY message back from its journey around the loop, it needs only to examine the state of the data bits of the message to determine which device or devices on the loop require service. If only one device is assigned to each data bit, then up to eight devices can be rapidly polled by the Controller.

ASSIGNING AUTO-ADDRESSES

The automatic assignment of addresses is a very simple task as far as the Controller is concerned. It does, however, assume the presence of additional logic on each of the devices that are to be capable of responding to the auto-address messages.

AAU
RFC

First, the Controller may need to send the Auto-Address Unconfigure (AAU) message out to the loop to reset any previous addresses. This also prepares all devices on the loop for reception of their new addresses. When the Controller receives the RFC message back from the loop, it

can then proceed with address assignment.

AAD1 The Controller sends the first auto-address message out to the loop and the first device to receive the message accepts that address (1) as its address. That device then increments the address contained in the message and sends the message on to the next device on the loop. That next device repeats the procedure. It accepts address 2 as its own, increments the address and retransmits the auto-address message.

AAD2

AAD3

AADn This process continues until the auto-address message returns to the Controller. When the Controller gets back the AAD message, it can examine the message to determine how many devices have been assigned addresses. The AAD message will contain the highest address, plus 1, that was assigned.

If the AAD message returns to the Controller containing an address of 31, it may mean that there are too many devices on the loop. To check this, the Controller could send out the AAD1 message again. Since devices should not respond to another auto-address assignment until the AAU message has been sent out, this second attempt to assign addresses should be ignored and the AAD1 message should return unchanged. If the AAD1 message has been incremented when it returns to the Controller, it indicates that there are too many devices connected to the loop. Some sort of corrective action must therefore be initiated, since the loop will not work if there are too many devices connected.

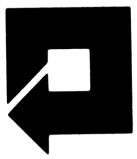
PASSING LOOP CONTROL TO ANOTHER CONTROLLER

As previously mentioned, there can be only one active Controller on the loop at any one time. Additionally, one device must be designated as the System Controller and must be responsible for such things as initializing the loop on power-up. There can be more than one device capable of being Loop Controller, however, and control of the loop can be transferred from device to device. The procedure is quite simple.

TADn The current active Controller first addresses the device that is going to become the Controller as a Talker using the Talk Address (TAD) message. If the desired device were already the active Talker, this step would not be needed.

TCT The current Controller then sends out the Take Control (TCT) message. The device addressed as Talker does not pass this message on around the loop to the Controller that sourced it. Instead, the device immediately assumes the role of Controller and replaces the TCT message with its first operation.

When the previous Controller receives back a message other than the TCT message, it knows that another device is now the active Loop Controller. If the previous Controller should receive back the TCT message, it indicates that the addressed Talker could not or would not assume the role of Controller. The previous Controller must, therefore, resume control of the loop and take whatever actions are necessary.



CHAPTER 6

The Interface Functions

Several times the last few chapters have referred to the "interface functions" defined by the HP-IL specification. However, these functions have not been defined or described to explain their significance. Now it's time to dig deeper into the HP-IL specification to see how these interface functions define all of the loop-related characteristics of a device.

Let's begin by listing all of the interface functions. They are grouped into three categories: primary functions, device control functions, and address/status/service-request functions.

Primary interface functions:

- Receiver
- Acceptor Handshake
- Driver
- Listener
- Source Handshake
- Talker
- Controller.

Device-Control interface functions:

- Power-Down
- Device Clear
- Device Trigger
- Remote Local.

The Address/Status/Service-Request interface functions:

- Parallel Poll
- Service-Request
- Automatic Address
- Auto-Extended Address
- Auto-Multiple Address
- Device Dependent.

The names of some of these interface functions are, no doubt, familiar to you.

Talkers, Listeners, and Controllers have been discussed throughout the preceding chapters along with service-requests and auto-addressing. However, these phrases usually were used in the context of "device roles" or "device capabilities." The complete definition of a device's role and its capabilities is provided by determining which of the 17 interface functions are implemented by that device.

Some of the functions must be implemented by every device that is going to operate on the loop. Other functions, such as Talker or Listener, obviously must be implemented if the device is to assume one of those primary roles. Additionally, within most of the interface functions, there are subsets of the function that a device can implement, thus further varying the personality of a particular device.

Essentially, then, you decide on what loop-related capabilities a device must have and then design the device by implementing the interface functions that define the needed capabilities. This process might be compared to deciding what equipment you want on a car. Some of the equipment, such as wheels, engine, and brakes, is required if you are going to operate the car on the road. However, even among those required capabilites, you may have choices such as engine type (6-cylinder, 8-cylinder), tires (radial or bias-ply), and so on. Other equipment may be completely optional, such as the music system, but you may have choices even within these categories (8-track or cassette).

Your decisions about the functions or options you want to implement will also involve tradeoffs that are similar to those you might make when selecting and specifying a new car: performance and creature-comforts vs. cost. You must pay for each of the interface functions you choose to implement by adding more intelligence or logic to the device.

The best way of gaining an understanding of the interface functions is probably to just dive into a description of them. We'll begin with the primary ones that will be implemented most often.

Remember, the descriptions that follow will often refer to one interface function communicating with another interface function. This concept may seem confusing at first, since all communications discussed thus far have been between separate devices on the loop. However, this simply implies that the capabilities and options that you can implement on a particular device must usually interact with one another.

If you recall the analogy of the automobile, you could say that the automatic transmission option interacts with the engine function to control speed of the engine and vehicle. The transmission does not operate in a vacuum. It makes no sense to try to think of the transmission traveling down the freeway (around the loop) on its own. Similarly, many of the interface functions are interdependent. Each performs a precisely defined function within a device/system, so you can economically select only those options that you want. However, all of the options must work closely together to ensure that the device can operate on the loop.

THE PRIMARY INTERFACE FUNCTIONS

These are the interface functions that are either required for all devices operating on the loop, or that are intimately related with the three primary loop roles of Talker, Listener, and Controller.

Receiver (R)

All devices on the loop must implement the Receiver interface function, since every device must be able to receive messages from the loop. The Receiver interface function has responsibility for performing some decoding of incoming messages to determine if this device has to deal with the message in any way, or if it can simply be immediately retransmitted.

The Receiver function does not perform the retransmission of the message itself. Instead, it informs the Driver interface function (which will be described later) that the current message should be retransmitted.

The Receiver function interacts closely with the Driver function, since it must wait until the Driver function has finished sending out any previous message before it tells the Driver that there is another message ready for retransmission.

Since the majority of the messages that any device receives will be intended for other devices on the loop, the Receiver function will spend most of its time checking incoming messages for destination, and letting the Driver function know that it must send another message out onto the loop.

Since messages are received bit-serial on the loop, and since the three most significant bits (the first received) tell you the type of message you are receiving, you can usually tell if a particular message might be for your device even before the rest of the bits are received. For example, if your device begins receiving a command message and it has not just sourced such a message, it knows that this message should be immediately retransmitted. (Remember, all command messages will be followed by an RFC message to determine if the previously issued command has been completed.)

Note that there are no subsets of the Receiver interface function. All devices must implement all of the capabilities of this primary function.

Acceptor Handshake (AH)

All devices on the loop must implement this function. The AH interface function allows a device to receive messages from the Receiver function (another required function) which are directed to this particular device. The AH function also must determine whether the message it has received needs to be retransmitted after it has been decoded. Note that this decoding goes beyond that performed by the Receiver function. The Receiver determined whether this device had to involve itself with an incoming message and, if not, informed the other functions that the message could be immediately retransmitted.

The Acceptor Handshake function, however, must further decode the message to determine when the message has been fully decoded (beyond the first three bits)

and whether it's the type of message that needs to be retransmitted. For example, if this device sourced the message, and it was a data message, it will not need to be retransmitted.

The Acceptor Handshake function has no subsets. All devices must fully implement all of the capabilities.

Driver (D)

All devices must have the Driver interface function, since it provides the device with the capability of transmitting messages on the loop. Regardless of a device's primary role on the loop, it must be able to pass messages on around the loop. This is the responsibility of the Driver function.

Messages that the Driver function must handle can come from three different places: messages coming in that must be immediately retransmitted, messages that must be retransmitted after this device has taken a look at them, and messages that this device might have generated itself.

Since the Driver function is such a primary function, there are no subsets defined nor allowed.

Listener (L)

The Listener interface function need only be implemented by devices that will assume the role of Listener on the loop. Notice here the differentiation between Acceptor Handshake logic, which is required to allow a device to receive a message (and, likely retransmit it without any other action), and Listener which implies a different and larger set of responsibilities. This precise division of labor between the functions can be implemented on a device to give you maximum flexibility and (hopefully) economy. It does, however, make the description of loop-device characteristics and capabilities more complicated.

The Listener function is only active when this device has been addressed as a Listener (it received its LAD message). When a device has been designated as a Listener on the loop, this logic must receive messages from the Acceptor Handshake logic that are intended for the device itself. Thus, you can view the Acceptor Handshake function as an interface between the loop and the Listener function, and the Listener function as the interface to the device-specific functions.

There are two parts to this function: one allows use of a one-frame address and the other permits use of an extended, two-frame address. Refer to the discussion of auto-extended addressing later in this chapter and in Chapter 4 for details.

Source Handshake (SH)

This function must be implemented by all devices that need to originate messages—that is, devices that operate as a Talker or Controller on the loop. Since most devices will usually be required to operate as Talkers some time (if only to send status information), the Source Handshake function usually will be required.

The Source Handshake function coordinates the transfer of messages from the device itself to the Driver interface function. This function does not actually generate messages, it merely receives messages

from the device and then passes them on to the Driver function.

There are no subsets of the Source Handshake function. If a device requires this capability, it must fully implement the function.

Talker (T)

This function is, obviously, one of the primary functions. If a device implements this function, it has a voice on the loop and assumes more than a few responsibilities. There are two main parts to this function: the basic function responds to a one-byte address, while the Extended Talker function responds to a two-byte address.

If a device has the Talker function, that capability is not activated until the device receives its Talk Address (TAD) message. Once it has been addressed as the Talker, the Talker interface function is enabled to send messages that its device generates out onto the loop. The Talker function must wait until a Send Data (SDA) message is received before it can actually send out messages (via the Driver function).

The Talker interface function also is responsible for responding to serial polls by sending out one or more bytes of device status information.

The End of Transmisson (EOT) message needed after data and status transmissions also must be generated by the Talker function.

Controller (C)

This is obviously a major function and, just as obviously, is not a function that must be implemented by all devices on the loop. Remember, there is only one Controller at any one time on the loop.

If a device implements the Controller interface function, it can send the Command (CMD), Ready (RDY), and Identify (IDY) messages to other devices on the loop. It must also be able to conduct parallel-polling operations, check for service-requests and detect transmission errors in device-dependent transmissions.

As you saw in Chapter 5, there can be more than one device on the loop capable of operating as the Loop Controller. This responsibility is passed between devices using the Take Control (TCT) message. However, there can be only one device designated as System Controller. The System Controller usually does its chores at system power-up and is the only device on the loop that can source the Interface Clear (IFC) message used to initialize devices on the loop.

You can derive most of the characteristics of the Controller interface function by looking at the loop sequences described in the preceding chapters. Whenever there is a particular sequence of messages required, it is usually the Controller function that is responsible for scheduling and implementing that sequence.

The Controller interface function gets messages from the Receiver and Acceptor Handshake functions and sends messages out via the Driver and Source Handshake functions. Thus, you might visualize the Controller function as operating in somewhat of an ivory tower, once removed from the pedestrian activities going on around

the loop or within the device. The Controller function receives information from the device and the loop and generates commands based on this information and on the system operating criteria you have programmed into the Controller function logic.

There are several optional subsets of capabilities that can be implemented in a device that has the Controller interface function:

- C0 (device has no Controller capability).
- C1 (device has basic Controller capability and can send messages and detect transmission errors).
- C2 (device is System Controller and can send the IFC message).
- C3 (device can respond to service-request messages that travel around the loop).
- C4 (device can transfer and receive control of the loop).
- C5 (device can configure devices for and execute parallel polling).
- C6 (device can handle asynchronous loop operations).
- C7 (device can assign automatic addresses).

Note that these Controller function options or subsets are not hierarchical. All controllers must have the C1 capability, but you can implement whichever of the other capabilities you want. Thus, you can have a device that can recognize service-requests and that can transfer control. Such a device would be identified as C1,3,4.

This discussion should illustrate that not all of the operations and sequences described in the preceding chapters are possible in all systems. For example, if the Controller in your loop does not implement the parallel-poll capability (C5), that operation cannot be performed on the loop. Therefore, if you were designing a device to operate on this particular loop, it would not make sense to include the Parallel Poll interface function (which will be described later) on that device. Once again, this demonstrates how the precision of the HP-IL specification leads to both flexibility and economy of design. You can select the exact capabilities you want for your device and know that so long as you include the minimum required functions, your device will be able to operate on the loop.

DEVICE-CONTROL INTERFACE FUNCTIONS

These interface functions are, as the name assigned to them indicates, more closely associated with how actual devices attached to the loop operate, rather than with how the loop itself performs. The names of these functions will probably be the most unfamiliar of those described in this chapter, since they determine how an HP-IL interface device you design will interface to the loop.

Power Down

This function gives a device the capability to be placed in a power-down or low-power mode of operation via a command received over the loop. It also allows the device to be powered up again under loop control. The

Power-Down function is optional and there are no subsets of the function. A device either has the capability or it doesn't.

Devices that do implement this capability must place themselves in a low-power state or completely powered-down condition when they receive the LPD (Loop Power-Down) message from the loop. It is up to the device designer to determine the details of the powered-down condition. The loop doesn't care whether the device is completely powered-down or placed in a low-power state.

The loop does require, however, that a device that implements this function and responds to the LPD message, be able to power up when it receives a message from the loop. Typically, the Controller will "wake up" or reactivate devices that have been powered-down by repeatedly sending a command out until the command eventually returns to the Controller. Devices that are powered-down are expected to monitor the loop for any incoming pulse. When the leading edge of any message is detected, the device must reactivate itself and begin to pass messages on to the next device on the loop.

Devices that do not implement the Power-Down function simply pass the LPD message on when they receive it. Notice that the LPD message is not addressed to any particular device on the loop. All devices that have the capability will power down when they receive the LPD message. (Actually, they must first pass the LPD and RFC messages on to the next device on the loop before they power down.) Also note that any activity on the loop after the LPD message will automatically wake up all the powered-down devices.

This capability would be quite useful in remote, unattended operations. If a particular device is battery-powered or consumes a lot of power or both, it may make sense to turn it off whenever it is not needed and there is a lull in loop activity.

Device Clear

This function lets the Controller initialize devices on the loop by sending them a "clear" message. This function is optional, although most likely you would want to implement the function on most devices. While devices typically perform some type of initialization when they are first powered-up, it is usually a good idea if the Controller has some positive way of establishing device initialization.

The HP-IL specification does not define the actions that a device must take when it receives the Device Clear message: the designer has complete freedom in deciding what the device should do to clear itself. This is quite logical since the actions that a device would take during initialization vary widely. A cassette and a CRT obviously would have little in common as far as initialization activities.

There are two versions in which the Device Clear function can be implemented. Actually, there are three versions if you consider the version where the function is not implemented at all. If a device does not implement this function, HP-IL nomenclature defines it as DC0.

If a device responds to the Device Clear (DCL) message, it is defined as having the DC1 capability. The DCL message will reset *all* devices on the loop that implement this DC1 capability.

If a device responds to the Selected Device Clear (SDC) message, it is defined as having the DC2 capability. The SDC message is addressed to a particular device so that the Loop Controller can reset or clear a single device. This capability can be quite useful if a particular device is producing errors or requires frequent human intervention (such as a printer might).

Note that if a device is designated as having DC2 capability, it means it can respond to both the DCL and SDC messages. You are not allowed to implement only the SDC response capability.

Device Trigger

This function lets the Controller initiate the basic operation of a device by sending it a message (GET—Group Execute Trigger) on the loop. This function is optional and there are no subsets of the function. A device either has it or it doesn't.

The Device Trigger function will typically be used to synchronize the operation of a device with real-time operations or to synchronize its operation with that of other devices on the loop. For example, if you had several temperature/pressure measuring devices on the loop, you could trigger them all to perform their basic measuring function at (approximately) the same time by sending the GET message around the loop.

If you implement the GET function in a device, that device must also have the Listener function implemented. Further, the device must be addressed as an active Listener on the loop before it can respond to the GET message. This approach lets the Controller address either a single device for the triggering operation or address several devices as Listeners so they can be triggered at the same time.

Again, note that the action the device itself takes when it is triggered is completely up to the device designer. The loop could care less about the details of what the device is doing when it is triggered.

Remote Local (RL)

This function lets the Controller direct a device to accept information from either the loop (remote) or from its own front panel or switches (local). This capability is useful for devices that must have settings and controls set up manually at various points in their operation. For devices that are programmable, this function allows them to be programmed under loop control or local control. As a simple example of this, you could establish a voltmeter's voltage range setting by sending the proper data messages to it while in remote mode, or you could press the correct range switch on the front panel in local mode.

The Remote Local function is optional. A device need not implement the function if it doesn't need it.

If a device does implement this function, it must also be capable of being a Listener, since it must be addressed as an active Listener before it can activate remote mode or respond to the Go To Local (GTL) message

that is used with this function.

Devices that implement this function must power up in the local mode. They will remain in the local mode until they have received the REN message and are addressed as a Listener. At that point, they go to the remote state and can be programmed by the Controller.

The device will remain in the remote state (even if it is subsequently unaddressed as a Listener) until it receives the Go To Local (GTL) message. The GTL message is not addressed to any specific device—it is directed to all devices that are currently addressed as Listeners. Thus, the Controller can select which device(s) should Go To Local by first selecting the device(s) as Listeners.

The Remote Local function is optional. If a device does not implement this function, it is designated as RL0. If it implements the basic Remote Local function just described, it is designated as RL1.

There is an extended version of this function, designated as RL2, that has a "lockout" capability. In this version, the Controller can send a device the Local Lockout (LLO) message that will prevent an operator from inadvertently changing an instrument's control settings at some critical time. Once a device is locked out, it remains locked out until it receives a Go To Local (GTL) message.

THE ADDRESS AND OTHER INTERFACE FUNCTIONS

These interface functions should sound a little more familiar than the device-related functions just described. The nomenclature and messages discussed here have appeared in the earlier chapters. Generally, these functions deal with the abilities of a device to respond to commands that are addressed to a specific device, the way that a device responds to a request for status information, and the ways in which a device can request service from the Controller.

Admittedly, this is sort of a catch-all category created to cover these functions. However, there are only a few functions left and it is easier to clump them together here.

Parallel Poll

This function enables a device to respond to a parallel poll by the Controller by returning one bit of status information within the IDY polling message. The Parallel Poll function is optional. Devices do not have to implement this capability. There are no subsets of this function. A device either has it, or it doesn't. If a device is going to implement this function, however, it must also be capable of being a Listener.

If a device implements this function, it must be able to respond to several loop messages: Parallel Poll Enable (PPE), Parallel Poll Disable (PPD), Parallel Poll Unconfigure (PPU), and Identify (IDY).

Again, refer to Chapter 5 for a more thorough discussion of parallel-polling. However, the parallel-polling operation will be summarized here.

The Controller defines the bit number and polarity that a device should use to respond to a parallel poll by sending out

the PPE message. Individual devices can be disabled to prevent their response to a poll by sending them the PPD message. All devices on the loop can have their parallel-polling capability disabled by sending out the PPU message.

The actual polling operation is performed by sending the IDY message around the loop. Devices that have the parallel-polling capability and that are currently enabled, respond by setting a bit in the IDY message data byte.

Service-Request

This function lets a device request service from the Controller by setting the service-request bit in a Data or End (DOE) message or in an Identify (IDY) message. When the Controller gets a message back with the service-request bit set, it knows that some device on the loop needs service, but it doesn't know which one. The Controller then must perform a polling operation to discover which device or devices require service.

The Service-Request function is optional. If it is implemented, the device must also have the Talker interface function implemented so that the device can send status bytes to the Controller as a response to polling operations.

For a thorough discussion of service-requests and polling operations, refer to Chapter 5, which includes illustrations of typical loop sequences.

There is one subset of the Service-Request function. If the device does not implement the Service-Request function at all, it is designated as SR0. If it implements the basic Service-Request function just described, it is designated as SR1. The subset of the Service-Request function, designated SR2, allows a device to asynchronously request service by generating its own IDY message. We discussed asynchronous loop operations briefly in Chapter 4 and then advised against implementing that capability unless you absolutely must. That advice will be repeated here and you can refer to the HP-IL specification for the details on the capability.

Automatic Address

This function allows a device to be assigned an address via a command from the loop (as opposed to having the address hard-wired or manually set). The Auto-Address function is optional. If all devices on the loop implement this function, however, the user is completely relieved of the necessity of manual address configuration, and so it is strongly recommended.

If a device does not implement the Auto-Address function, it simply ignores associated messages (AAU and AAD) and passes them on to the next loop device. If this function is implemented, the device accepts the address contained in an AAD message as its address, then increments the address and passes it on to the next device on the loop. (Refer to Chapter 4 for a thorough discussion of Auto-Address operations.)

There are no subsets of the Auto-Address interface function. However, there are two other interface functions that provide variations of the auto-addressing capability. If a device is going to have some

auto-addressing capability, it only has to implement one of these three auto-addressing functions. Here are the two variations of this capability.

Auto-Extended (AE) And Auto-Multiple (AM)

The basic auto-addressing function described in the preceding paragraphs allows as many as 31 devices to be addressed on the loop. The Auto-Extended and Auto-Multiple functions permit up to 961 devices to be addressed.

The operation of the Auto-Extended function is basically as we described for the auto-addressing function. However, instead of the address being completely contained in the Auto-Address (AAD) message, there is an additional byte of address information passed to the device. The first message sent is called the Auto-Extended Secondary (AES) message. Each device that has implemented this function will store the address contained in this message, increment it, and pass the message on to the next loop device. This operation is the same as the one described for the standard auto-addressing mode. However, when the address contained in the AES message has been incremented to its maximum (31), subsequent devices will no longer respond to the message but will just pass it on around to the Controller.

The Controller will then send the Auto-Extended Primary (AEP) message out. This time, however, only those devices that received legal addresses in the AES message will accept this primary address as the most significant part of their address. Furthermore, they will not increment the address contained in the AEP message, but simply pass it on around to the Controller. This group of devices is now configured and should not respond to any further AES or AEP messages until they receive the Auto-Address Unconfigure (AAU) message.

The Controller can now send out another AES message to the next group of devices (which did not receive a legal address in the AES message on the previous round). This will again be followed by the AEP message. This sequence will be repeated until AES is returned to the Controller with a legal address still contained in the message.

The Auto-Multiple address function is similar to the two functions just described but is intended for use with devices that have multiple functional capabilities implemented within the device itself. The Auto-Multiple function allows each function within the device to be assigned a separate loop address. Here is the sequence that would be used with the Auto-Multiple function.

The Controller would first send out the Auto-Multiple Primary (AMP) message. Each device implementing the multiple function would accept this address as its own, increment the address and pass it on around the loop. Next, the Controller would issue the Zero Extended Secondary (ZES) message. When the first multiple device receives the ZES message, it begins assigning addresses to each of its internal functions beginning with address zero. As it assigns these internal addresses, it incre-

ments the address contained in the ZES message. When all internal addresses have been assigned, the device passes the ZES messsage back around to the Controller, which can then tell how many function addresses are assigned to that device. The Controller then sends the ZES message to the next device on the loop until all multiple devices have responded.

Device Dependent (DD)

This function lets the Controller send command messages to devices where the meaning of the message depends entirely upon the device receiving it. A device must first be addressed as a Talker or Listener before it can respond to a Device Dependent message and the Controller uses different messages for Talkers and Listeners: DDL (Device Dependent Listener) and DDT (Device Dependent Talker).

The Device Dependent interface function is optional but, since it provides such a simple way to initiate and control device activity, it will probably be implemented in most devices.

The least significant five bits of the DDL and DDT messages contain the Device Dependent command information. Thus, the device designer has up to 32 separate commands that can be defined for a particular device. The HP-IL specification places no limits on the uses of the Device Dependent commands. The effect of a command on a device is left completely up to the designer.

For example, in a tape cassette you might use Device Dependent commands to perform such tasks as rewind and advance tape to next record. A printer might use Device Dependent commands to initiate such activities as advance paper to top of form, or to set up a double-space mode of operation.

Note that since there can only be one active Talker on the loop at any one time, there can never be any confusion about which device a DDT message is directed to. There can be multiple active Listeners, however. Therefore, you must ensure that there is not more than one active Listener which has implemented the Device Dependent function unless you want the DDL message to be accepted by all of the active Listeners.

CHAPTER 7

What You Have Not Been Told

The approach in this book has been to use each successive chapter to lead you to a greater level of detail about HP-IL. The intent has never been, however, to replace the actual HP-IL specification. Instead, the basic concepts and capabilities of HP-IL have been presented so that you might evaluate its suitability for your application and gain some appreciation for the amount of work involved in implementing HP-IL at various levels.

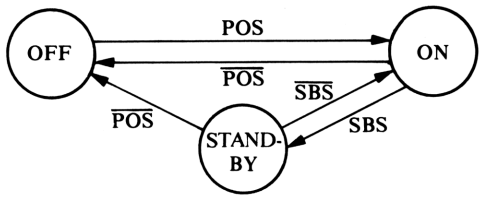
We do not, therefore, claim that this book is comprehensive. In fact, there are some details that have been intentionally left out, since they will be needed only by those who must implement the HP-IL specification at the most detailed level.

This chapter will give you a glimpse of some of the details that have been avoided thus far. The purpose is twofold: first, to give even the casual reader a look at the total picture of HP-IL and an appreciation of the thoroughness of the specification; second, to give those who must eventually deal with HP-IL at this most detailed level a head start, or at least a running start, at deciphering the "state diagrams," which fully define the interface functions of HP-IL.

SIMPLE STATE DIAGRAMS

State diagrams are often used, especially in the design phase, to describe all the legal states in which a device or system can operate, and to define the conditions that must exist to allow or cause a transition from one state to another.

The following is an illustration of a simple state diagram.



The device depicted in this illustration has three legal operating states: OFF, ON, and STANDBY. Thus, the legal states are shown as circles with their names (or mnemonics) within the circles.

Arrows connecting the state circles show the legal paths for going from one state to another. The conditions that allow transitions from one state to another are defined by the letters alongside the connecting arrows. In the preceding illustration, POS stands for Power On Switch and SBS stands for StandBy Switch. Also note that in the nomenclature used here, POS with no line above the expression indicates that the condition is "true" (the switch is on) while an expression with a line over it means that the condition is "false" (for example, power is off or *not* on).

Thus, for this device to go from the OFF state to the ON state, the POS term must be "true" (the Power On Switch must be on). The transition from the ON state to the STANDBY state can occur when the StandBy switch (SBS) is true.

Note that this state diagram allows a transition from the STANDBY state to the OFF state if the Power On Switch is "false" (power is turned off). A transition from OFF to STANDBY is not permitted, however.

The state diagrams for HP-IL, as you might expect, are more complicated than the preceding illustration. Nonetheless, the same simple principles that have just been discussed apply, no matter how complicated the diagrams might appear.

LOCAL MESSAGES AND PSEUDO-MESSAGES

One more hurdle must be cleared before you can proceed to the actual state diagrams for HP-IL. Most of the terms used as conditions for making transitions between states will be familiar. They will consist of messages and/or interface function expressions that have been discussed in preceding chapters. Occasionally, however, state transition conditions will include expressions (always shown in lower case) that are defined as local messages or pseudo-messages.

HP-IL defines remote messages as the messages that travel around the loop itself—these are the messages that were explained in preceding chapters. The HP-IL specification also loosely defines local and pseudo-messages. These are messages or signals exchanged between the interface functions of a device and the completely device-dependent functions. The simplest examples of local messages are the pof (power off) and pon (power on) messages. These two local messages are used in the state diagrams of the Power-Down interface function (described in Chapter 6) and simply indicate that the interface function and the device electronics must have some way of indicating to each other the state of

the power switch of the device.

There are numerous local and pseudo-messages defined by the HP-IL specification. However, you don't need to know all of them unless you are going to work with the state diagrams. Therefore, rather than defining all of them, we will simply explain the few that appear in the state diagrams used here.

DRIVER STATE DIAGRAM

Since all devices on the loop must have the Driver interface function, it is one of those to be discussed. Figure 7-1 shows the state diagram for this interface function. There are four defined states:

- DACS - Driver Transmit from Acceptor State
- DIDS - Driver Idle State
- DSCS - Driver Transmit from Source State
- DTRS - Driver Transfer State

At this point, you may want to refer to Chapter 6 for a discussion of the Driver interface function. Briefly, this interface function is responsible for transmitting all messages (the remote messages) out onto the loop, regardless of whether the message is being initiated by this device or was simply being passed along the loop.

One more bit of nomenclature must be explained at this point. The four-letter mnemonics appearing in this diagram (for example, PONS), and enclosed in brackets represent a linkage from or to another of the interface states, in another interface function.

The Driver Idle State (DIDS) is entered from the Power On State (PONS) which is a separate interface function (the PD function). In this idle state, the Driver function is just waiting for something to happen and is not transmitting any messages. This is the state that the Driver function goes to when power is first applied.

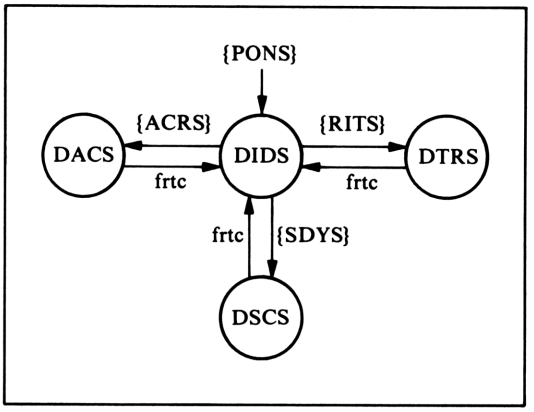


FIGURE 7-1. Driver interface function state diagram

Three different paths can be taken from the idle state. Each of them basically represents another interface function (in this case, three different sources of messages to be transmitted) that the Driver interface function must service. The other three functions "serviced" by the driver are the Source Handshake (SH), Acceptor Handshake (AH), and Receiver (R) interface functions.

The RITS (Receiver Immediate Transfer State) expression causes a transition from the idle state to the DTRS state and indicates there is a message that must be retransmitted immediately and not delayed while the device decodes it.

The ACRS (Acceptor Ready State) expression causes a transition from the idle state to the DACS state and indicates that a message has been received and accepted and can now be retransmitted.

The SDYS (Source Delay State) expression causes a transition from the idle state to the DSCS state and indicates that the driver should transmit a message which is being sourced by this device itself.

Note that all three of the expressions discussed here (RITS, ACRS, and SDYS) are themselves states within other interface functions and are defined in the state diagrams for those functions. You should now have some idea of the complexity of the HP-IL specification (and of designing at this level). There is a tremendous amount of interaction that must occur between all of the interface functions. This approach of dividing total system and device capabilities along strictly and compactly defined functional lines can lead to great flexibility and economy. However, it can also be quite complicated.

The common term to all three transition paths back to the idle state is "frtc" (Frame Transmission Complete.) The frtc expression is a pseudo-message generated by the device's encoder circuitry to signal the driver function that it is done transmitting an entire frame and that a return to the idle state can be made. Thus, the function returns to wait for another request for transmission of a message.

LISTENER STATE DIAGRAM

Figure 7-2 shows the state diagram for the Listener interface function. This diagram is more complicated than that for the Driver and includes many more terms in the expressions that cause transitions between states. Let's begin with the Listener Idle State (LIDS).

As you might expect, this function also powers up (note the link from the PONS state) in the idle state. The transition to LADS (Listener Addressed State) is fairly straightforward. It requires that this device

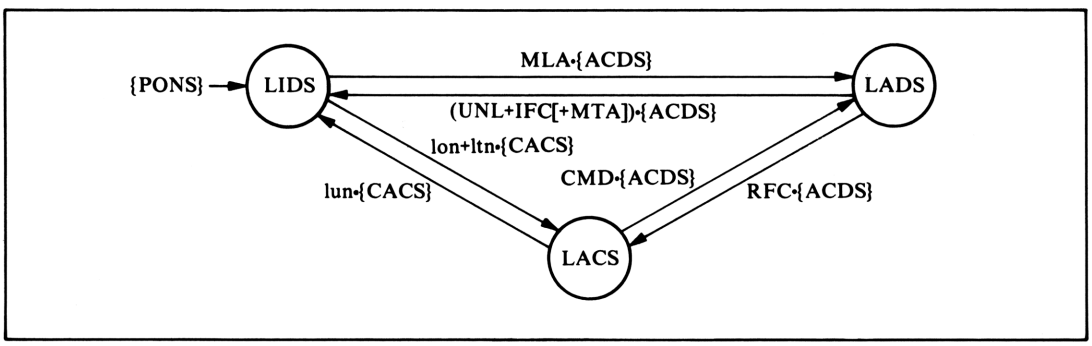


FIGURE 7-2. Listener interface function state diagram

has received its listen address (MLA = My Listen Address) from the loop and that the Acceptor Handshake (AH) function is in the Acceptor Data State (ACDS), indicating that the MLA message has been fully decoded and is valid for this device. Note that the dot joining the MLA and ACDS expressions means AND; that is, both expressions must be true for the entire expression to be valid.

The transition from LADS to the Listener Active State (LACS), is also straightforward. When the device is already in the Listener Addressed State (LADS) and the Ready For Command (RFC) message is received, then the function can proceed to assume the role of an active Listener on the loop. Once again, you will note that the other qualifying term for transition from LADS to LACS is Acceptor Data State (ACDS), which indicates that the RFC message has been completely decoded by the Acceptor Handshake function.

When the function is in the Listener Active State (LACS), it remains there, receiving messages over the loop until it receives a command (CMD) message.

If a CMD message is received, the device is still addressed as a Listener and returns to the LADS (Listener Addressed State). From there it can once again return to LACS if RFC is received, or it can go back to the idle state (LIDS). The expression that can cause a return to the idle state is (UNL+IFC[+MTA])•{ACDS}.

While this expression may look intimidating, it is not actually that complicated. It says if the Unlisten (UNL) message is received OR, (the plus sign means OR), if the Interface Clear (IFC) message is received, the function returns to the idle state (LIDS). Of course, the now familiar ACDS term also must be true.

There is one term in the expression not yet discussed, [+MTA]. You will notice that this term is enclosed in square brackets instead of parentheses or curly brackets. In the conventions used in the HP-IL state diagrams, the square brackets are used to indicate optional terms that may or may not be implemented by the function. In this case, it indicates that optionally the transition back to the idle state can also occur if a device addressed as a Listener is then addressed as a Talker. MTA is the My Talk Address message indicating that this device has been addressed as a Talker.

The other two transitions in the Listener state diagram are mainly of concern to a device that is also the active Controller.

ACCEPTOR HANDSHAKE STATE DIAGRAM

The Acceptor Handshake (AH) interface function, like the Driver interface function described earlier, must be implemented by all devices. This function receives messages from the Receiver interface function which are intended for this device. When the message is valid for interpretation, the AH function indicates this fact to the other functions. (You will recall the ever-present Acceptor Data State—ACDS—term encountered in the description of the Listener state diagram.)

The AH interface function also determines whether the message needs to be retransmitted after interpretation (this is

called "repeat") or whether it can be discarded (which is called "norepeat").

Figure 7-3 shows the state diagram for the Acceptor Handshake interface function. The figure shown does not look much more complicated than that for the Listener. However, its appearance has been simplified by using the terms "repeat" and "norepeat" in the figure. The actual expressions required to make up these two simplified terms are listed below the state diagram drawing and are quite involved. You can ignore those complete expressions for a moment and look at the simplified form of the state diagram.

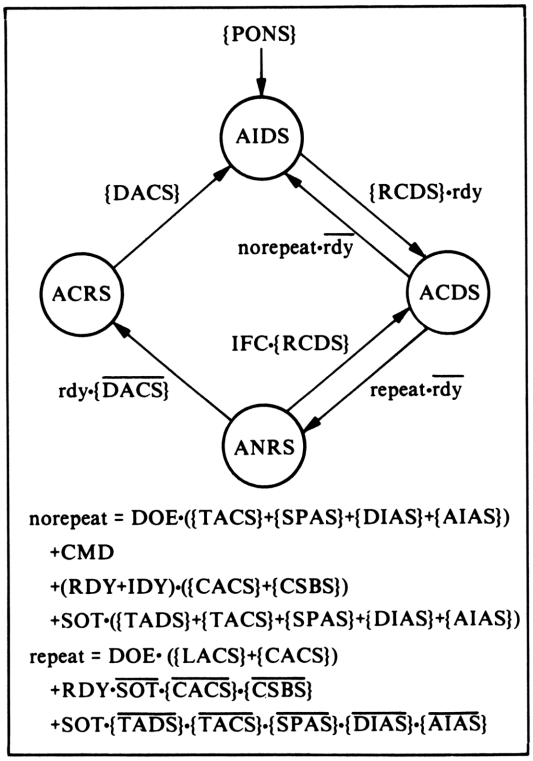


FIGURE 7-3. Acceptor Handshake interface function state diagram

Once again, the function starts out in the idle state (AIDS) when power is first applied. From there, the only legal transition is the Acceptor Data State (ACDS). This transition occurs when the local ready (rdy) message is true and the Receiver interface function indicates that a message for this device is available by entering its Receiver Data State (RCDS). The local rdy message is from the device and indicates to the Acceptor Handshake function that the device is ready to receive another byte of data.

After the device has accepted the message for interpretation, it sets the local rdy message false. The AH function will then return to the idle state (AIDS) if the message does not have to be retransmitted (norepeat), or to the Acceptor Not Ready State (ANRS) if the message must be retransmitted (repeat).

In the Acceptor Not Ready State (ANRS), the function is waiting for the device to indicate that it is ready for the next byte of data from the loop. The device indicates this by returning the local rdy message to the true state. This typically happens after the device has finished any actions necessitated by the current message. When the device is ready, the AH function can make the transition to the Acceptor Ready State (ACRS) where retransmission of the message can begin.

You will notice that in addition to the local rdy message, there is another term involved in causing the transition from ANRS to ACRS. The rdy message must be

accompanied by DACS being false. If you refer back to our earlier discussion of the Driver interface function, you will see that DACS (Driver Transmit from Acceptor State) is true when the Driver is actually sending out a message. Therefore, the AH function cannot proceed to ACRS where the Driver must begin sending out another message until the preceding message has been transmitted by the Driver.

When the function is in the Acceptor Ready State (ACRS) it is indicating to the Driver function that retransmission of the current message frame should begin. When the Driver function indicates that it has begun retransmitting that message (by setting DACS true), the AH function can make the transition back to the idle state (AIDS).

Return for a moment to the Acceptor Not Ready State (ANRS). We followed the transition path from that state to ACRS but, if you look at Figure 7-3 again, the function can also return to Acceptor Data State (ACDS) from ANRS. This transition path will be taken if an Interface Clear (IFC) message is received by the Receiver (RCDS) while in the ANRS state.

Now turn your attention to the lengthy expressions beneath the state diagram in Figure 7-3 that were simplified to norepeat and repeat. We are not going to explain all of the terms shown in these expressions, since most of them refer to states within other interface functions. This would require a discusssion of each of the state diagrams for the other interface functions involved. Instead, we will attempt to summarize the effect of each of these expressions to give you an idea of what factors are involved in allowing the Acceptor Handshake function to determine whether a message must be retransmitted.

Basically, there are three different types of messages that do not need to be retransmitted:

- DOE (Data or End) messages that were sourced by this device (the device is the Talker)
- CMD (Command), RDY (Ready), or IDY (Identify) messages that were sourced by this device (the device is the Controller)
- SOT (Start Of Transmission) messages intended for this device (once again, the device is the Talker).

All of the terms which follow these message mnemonics represent various states within other interface functions that interact with the Acceptor Handshake function.

Conversely, there are three categories of messages that must be retransmitted by the AH function.

- DOE messages that were not sourced by this device
- RDY messages that were not sourced by this device
- SOT messages that are not directed to this device.

Even without discussing all of the other terms involved in the simplified norepeat and repeat expressions, you can see that the interaction between functions can become quite complicated, as Figure 7-4 shows.

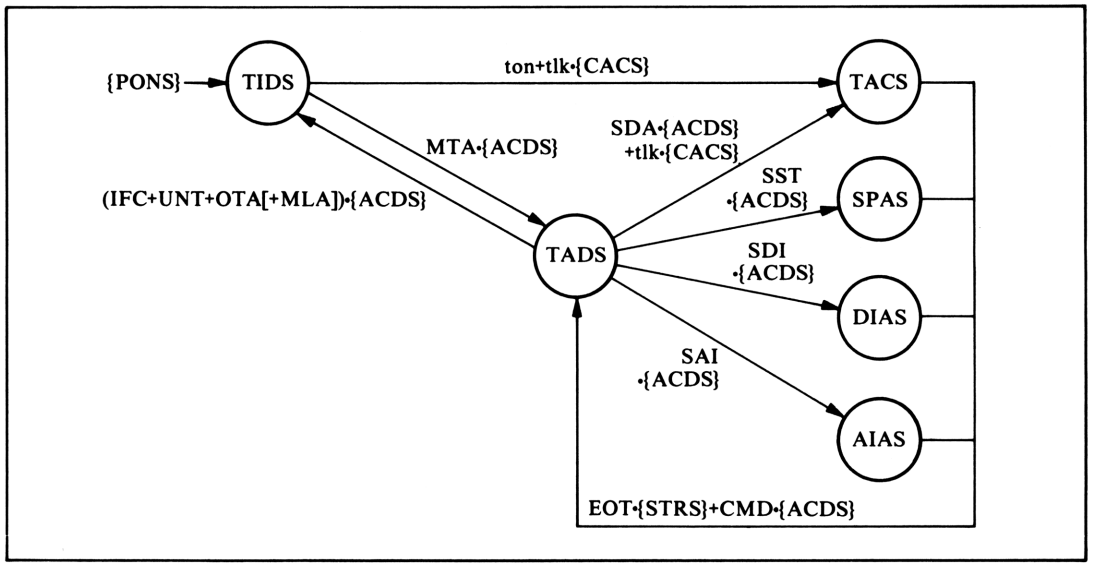


FIGURE 7-4. Talker interface function state diagram

TALKER STATE DIAGRAM

Figure 7-4 shows the state diagram for the Talker interface function. As you can see, it is more complicated than those discussed thus far. Remember, however that what makes this diagram appear more complicated is the fact that it has more states in which it can operate. This does not mean, though, that it is more difficult to understand or, more importantly, to implement than some of the simpler appearing diagrams. In fact, as a general rule of thumb, it is the number of terms in the transition expressions—especially when many of the terms are from other interface functions—that determine how complicated it may be to implement all the capabilities of a given function.

The Talker interface function state diagram will not be described in any detail, since you should have some understanding of how these things work by now. To help you get started on this one, here are the names of the six states illustrated for the Talker interface function in Figure 7-4:

- TIDS – Talker Idle State
- TADS – Talker Addressed State
- TACS – Talker Active State
- SPAS – Serial Poll Active State
- DIAS – Device Identify Active State
- AIAS – Accessory Identify Active State.

Since many of the terms in the transition expressions for this state diagram consist of remote messages, you should be able to refer to the Message Glossary in the back of this book to figure out quite a bit about what is going on in this diagram.

APPENDIX A

The HP-IL Instruction Set

Now that you have a fair understanding of HP-IL concepts, you may want to look over the entire set of messages that can be sent over the Loop interface. This collection of messages is called the HP-IL Instruction Set.

At first glance, the list of messages can be rather forbidding, as is often the case when you go to learn a new instruction set. But if you look at the way these messages are organized, you will find that they break down into four basic classes: Data or End, Command, Ready, and Identify (see Table A-1).

The following discussion is presented to help you understand the HP-IL Instruction Set. We begin by reviewing the general structure of an HP-IL message and then explain, in basic terms, the way in which HP-IL messages are grouped or organized.

Each HP-IL message in Table A-2 is listed alphabetically by its three-letter mnemonic. Each instruction or message is also defined in the Glossary at the end of the book.

THE HP-IL MESSAGE STRUCTURE

An HP-IL message frame can be thought of as a packet of information, 11 bits long.

TABLE A-1. The Coding of Bits C2, C1, C0

C2	C1	C0	Class	Mnemonic
0	END	SRQ	Data or End	DOE
1	1	SRQ	Identify	IDY
1	0	0	Command	CMD
1	0	1	Ready	RDY

TABLE A-2. HP-IL Message Table

Name	Message Coding	Class	Message Function	Subgroup
AAD	101 100 AAAAA	Ready	Auto Address 0-30	AAG
AAG	101 100 XXXXX	Ready	Auto Address Group	—
AAU	100 1001 1010	Command	Auto Address Unconfigure	UCG
ACG*	100 X000 XXXX	Command	Addressed Command Group	—
AEP	101 101 AAAAA	Ready	Auto Extended Primary	AAG
AES	101 110 AAAAA	Ready	Auto Extended Secondary	AAG
AMP	101 111 AAAAA	Ready	Auto Multiple Primary	AAG
ARG	101 01XX XXXX	Ready	Addressed Ready Group	—
CMD	100 XXXX XXXX	Command	Command Class Message	—
DAB	00X XXXX XXXX	Data or End	Data Byte	—
DCL	100 0001 0100	Command	Device Clear	UCG
DDL	100 101X XXXX	Command	Device Dependent Listener	ACG
DDT	100 110X XXXX	Command	Device Dependent Talker	ACG
DOE	0XX XXXX XXXX	Data or End	Data or End Class	—
EAR	100 0001 1000	Command	Enable Asynchronous Requests	UCG
END	01X XXXX XXXX	Data or End	End Byte	—
EOT	101 0100 000X	Ready	End of Transmission	ARG
ETE	101 0100 0001	Ready	End of Transmission — Error	ARG
ETO	101 0100 0000	Ready	End of Transmission — OK	ARG
GET	100 0000 1000	Command	Group Execute Trigger	ACG
GTL	100 0000 0001	Command	Go to Local	ACG
IAA	101 100 11111	Ready	Illegal Auto Address	AAG
IDY	11X XXXX XXXX	Identify	Identify	—
IEP	101 101 11111	Ready	Illegal Extended Primary	AAG
IES	101 110 11111	Ready	Illegal Extended Secondary	AAG
IFC	100 1001 0000	Command	Interface Clear	UCG
IMP	101 111 11111	Ready	Illegal Multiple Primary	AAG
LAD	100 001 AAAAA	Command	Listen Address (0-30)	LAG
LAG	100 001X XXXX	Command	Listen Address Group	—
LLO	100 0001 0001	Command	Local Lockout	UCG
LPD	100 1001 1011	Command	Loop Power Down	UCG
MLA	100 001 AAAAA	Command	My Listen Address	LAG

AAAAA = a 5-bit Address
XXXXX = Don't care bits
S = sense: 0 = set if SRQ; 1 = set if SRQ
BBB = bit : 000 = D0 to 111 = D7

* Also includes DDL & DDT commands

TABLE A-2. HP-IL Message Table (continued)

Name	Message Coding	Class	Message Function	Subgroup
MSA	100 011 AAAAA	Command	My Secondary Address	SAG
MTA	100 010 AAAAA	Command	My Talk Address	TAG
NAA	101 100 AAAAA	Ready	Next Auto Address	AAG
NES	101 110 AAAAA	Ready	Next Extended Secondary	AAG
NMP	101 111 AAAAA	Ready	Next Multiple Primary	AAG
NRD	101 0100 0010	Ready	Not Ready for Data	ARG
NRE	100 1001 0011	Command	Not Remote Enable	UCG
NUL	100 0000 0000	Command	Null Command	ACG
OSA	100 011 AAAAA	Command	Other Secondary Address	SAG
OTA	100 010 AAAAA	Command	Other Talk Address	TAG
PPD	100 0000 0101	Command	Parallel Poll Disable	ACG
PPE	100 1000 SBBB	Command	Parallel Poll Enable	ACG
PPU	100 0001 0101	Command	Parallel Poll Unconfigure	UCG
RDY	101 XXXX XXXX	Ready	Ready Class	–
REN	100 1001 0010	Command	Remote Enable	UCG
RFC	101 0000 0000	Ready	Ready for Command	–
SAD	100 011 AAAAA	Command	Secondary Address	SAG
SAG	100 011X XXXX	Command	Secondary Address Group	–
SAI	101 0110 0011	Ready	Send Accessory ID	ARG
SDA	101 0110 0000	Ready	Send Data	ARG
SDC	100 0000 0100	Command	Selected Device Clear	ACG
SDI	101 0110 0010	Ready	Send Device ID	ARG
SOT	101 0110 0XXX	Ready	Start of Transmission	ARG
SRQ**	0X1 XXXX XXXX	DOE or IDY	Service Request	–
SST	101 0110 0001	Ready	Send Status	ARG
TAD	100 010 AAAAA	Command	Talk Address	TAG
TAG	100 010X XXXX	Command	Talk Address Group	–
TCT	101 0110 0100	Ready	Take Control	ARG
UCG	100 X001 XXXX	Command	Universal Command Group	–
UNL	100 0011 1111	Command	Unlisten	LAG
UNT	100 0101 1111	Command	Untalk	TAG
ZES	101 110 00000	Ready	Zero Extended Secondary	AAG

AAAAA = a 5-bit Address
XXXXX = Don't care bits
S = sense: 0 = set if SRQ; 1 = set if SRQ
BBB = bit : 000 = D0 to 111 = D7

** Also 111 XXXX XXXX (IDY with SRQ)

The information contained within this packet is structured as shown in Figure A-1.

This 11-bit message is what travels over the loop itself from Controller to device, or from Talker to Listener. Figure A-2 depicts a typical message frame traveling down the loop. Each message is sent with the most significant bit of the frame first. Since messages are being sent serially over the loop interface, the first bits to arrive at a particular device will be bits C2, C1, and C0, in that order. So, each device on the loop will have a chance to decode these bits and prepare itself for the proper handling of the message.

If the message contains a command that pertains to a particular device, execution is usually deferred until after a local copy has been made at the device. This speeds up the overall throughput of the loop and allows devices to be concurrently executing commands at the local level.

Bits C2, C1, and C0 are used to classify the type of message which is contained in bits D7 through D0.

Bit C2

Bit C2 serves two purposes. Since it is the first bit sent in a message, it must act as the sync (synchronization) or start bit. It indicates to a device that this is the start of a message. This synchronization is necessary, since messages could arrive at a device at any time. (A message is an asynchronous event.) Bit C2 is also used to indicate to a device whether a message contains a data byte.

When bit C2 is 0, it indicates to a device that the current message contains eight bits of data (D7-D0). When bit C2 is a 1, it signifies to a device on the loop that this message contains one of the three remaining classes of messages: Command, Ready, or Identify.

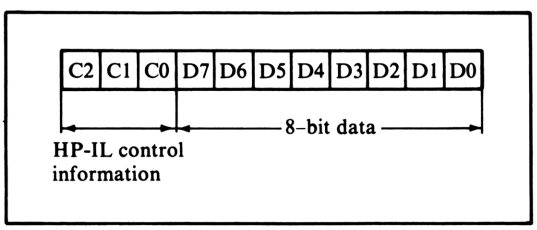


FIGURE A-1. An HP-IL message frame

Bit C1

If a message frame contains a data class message, bit C1 is used to indicate a logical end of record condition. When data bytes are being transferred from Talker to Listener via the interface loop, it is often necessary to tell the Listener that this byte is the last in a logical group. For example, ASCII files are often sent with a Carriage Return and Linefeed to indicate the end of a line of text. In this case the logical group is a line of text. If a file of text were being transferred over the loop, to a printer for example, bit C1 could be used to indicate the end of line condition. In non-data frames, this bit indicates to a device whether the message is a Command or Ready frame (C1 = 0), or an Identify frame (C1 = 1).

Bit C0

Bit C0 is used by a device to indicate to the Controller that it needs servicing as soon as

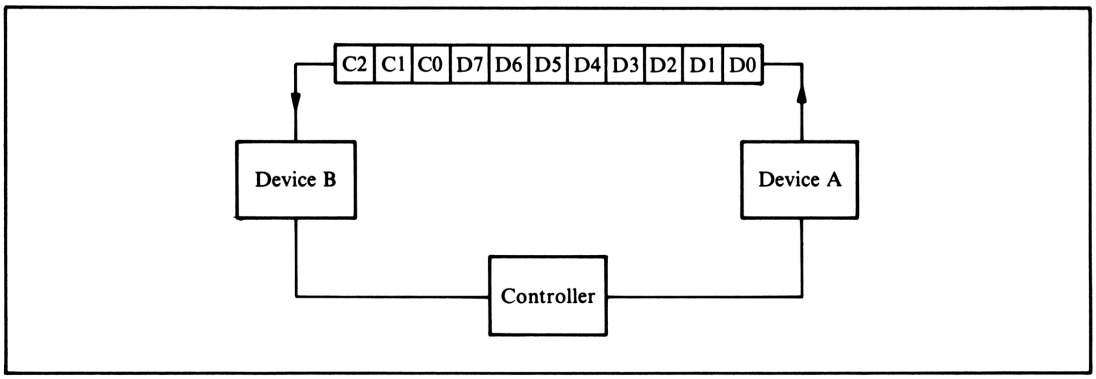


FIGURE A-2. A message on the loop

possible. This method of signaling a Controller is known as a service-request.

Since messages addressed to various devices on the loop are constantly passing through a device's interface, it must have some means of quickly alerting the active Controller. An input device may need to indicate that it needs servicing in order to prevent an overflow condition, or a printer may need to tell a Talker (via a Controller) that its buffer is full and it can no longer accept any new information. Note that C0 is the service-request bit only for Data and Identify frames. If C2 and C1 are 1 and 0 respectively, then C0 indicates whether the message is a Command (C0 = 0) or a Ready frame (C0 = 1).

Generally, bits C2, C1, and C0 are used to categorize the type of message that is being sent via the Loop. Bits C2, C1, and C0 define the HP-IL message hierarchy, a simple way of categorizing messages which will now be described.

THE HP-IL MESSAGE HIERARCHY

The structure of an HP-IL message and the various categories and types of messages have been briefly discussed. Messages are categorized in an organization or hierarchy.

This hierarchy is useful for two reasons. First, it can help you to quickly associate a message with its basic function; and second, it defines a logical and convenient way of implementing HP-IL functions in hardware or software.

HP-IL messages are (for organizational purposes) divided into four basic classes or major types: the Command class (CMD), the Data or End class (DOE), the Ready class (RDY), and the Identify (IDY) class.

These categories or classes are based upon the coding of bits C2, C1, and C0 of the message frame (Table A-1). Figure A-3 shows the overall organization of HP-IL

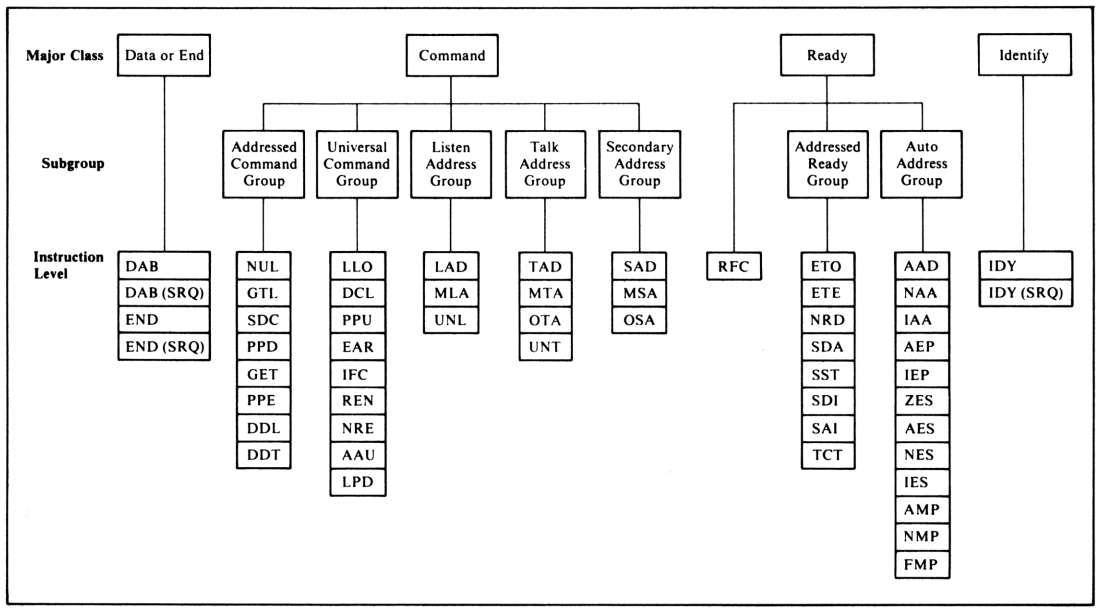


FIGURE A-3. HP-IL message hierarchy

messages, known as the HP-IL message hierarchy.

Command Class

The Command (CMD) class includes all messages sent from an active Controller to other devices on the loop. These messages contain commands that are used to control either interface or device functions. Instructions in the Command class may pertain to a single device or to all devices residing on the loop.

A device receiving a Command class message will immediately retransmit the message to the next device on the loop and will save its own copy of the command before beginning execution. The active Controller must follow the Command class message with a Ready For Command (RFC) message (after the Command returns) to verify that the device has completed execution of the command.

Instructions in the Command class include

- Those commands which are directed at a specific device (the Addressed Command Group);
- Those commands which are directed at all devices on the Loop (Universal Command Group); and
- Those commands which assign Listen and Talk responsibilities to devices on the loop (LAG, TAG, and SAG).

Data or End (DOE) Class

Messages in this class include all data sent from an active Talker to an active Listener. The last byte of data in a record is called the

End byte. The Talker may indicate this end of record condition by setting bit C1 in the data message. Note that the End byte is distinct from the End of Transmission message (ETO or ETE). The End byte does not cause the termination of transmission from a Talker to a Listener. Normally C1 is used to indicate a logical end condition. This may be an end of record, or, in the case of ASCII files, an end of line.

Ready Class

Ready class messages provide a means of controlling loop operations. There are three basic groups of messages within the Ready class: the RFC (Ready For Command) message, the group of messages known as the Addressed Ready Group, and the group of Ready frames called the Auto-Address Group.

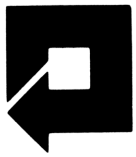
The RFC (Ready For Command) message allows the active Controller to determine when devices have completed the execution of a prior command.

Addressed Ready Group commands normally are used to control the sending of data from Talker to Listener. These instructions specify when a device should start sending data (SOT group) and indicate to the Controller when the transfer is complete (EOT group) or when a Controller must interrupt a data transmission (NRD message).

Ready class messages also include a group of commands known as the Auto-Address Group (AAG). Auto-Address messages simply allow a Controller to bring up or configure a loop without operator intervention. In this mode devices are effectively assigned an address through the use of messages in the AAG group.

Identify (IDY) Class

The class of messages known as the Identify (IDY) class is used by the Controller to determine if a device on the loop needs servicing. The process of identifying which device needs attention is called polling. The HP-IL command structure allows for two methods of polling: serial and parallel.



Terms and Remote Messages

The following is a listing of terms and remote messages used in the HP-IL. Messages are listed alphabetically by three-letter mnemonic. Following the definition of each message, the binary representation is given in which "X" equals a bit of any value and "A" equals any address bit.

AAD Auto-Address. This message allows a Controller to assign addresses to a maximum of 31 devices on the loop. The AAD message is issued by the Controller to the first device on the loop. The first device takes the number AAAAA as its address, then increments AAAAA and passes the message along to the next device on the loop.

(101 100 AAAAA)

AAG Auto-Address Group. This group of messages is issued by the Controller to assign addresses to devices that are on the loop.

(101 100 XXXXX)

AAU Auto-Address Unconfigure. This command is issued by the Controller to cause all devices capable of being auto-addressed to relinquish their present address assignments. Once a device has received an AAU command, it may be readdressed using the AAD command.

(100 1001 1010)

ACG Addressed Command Group. This group of commands is sent from a Controller to a device which has been previously addressed as a Talker or Listener. Commands in this group are used to control specific devices on the loop.

(100 X000 XXXX)
or (100 101X XXXX)
or (100 110X XXXX)

AEP Auto-Extended Primary. This message is issued by the Controller to assign a single extended primary address (AAAAA) to a group of devices that are

capable of extended-addressing and which have previously been assigned extended secondary addresses with the AES message.

(101 101 AAAAA)

AES Auto-Extended Secondary. This message is issued by the Controller to assign a group of secondary addresses to devices capable of supporting extended-addressing. A device receiving an AES message will take AAAAA as its secondary address, increment AAAAA and send it on to the next device on the loop.

(101 110 AAAAA)

AMP Auto-Multiple Primary. This message is issued by the Controller to assign primary addresses to devices capable of multiple addressing. The Controller issues the AMP message and each device in turn accepts the primary address AAAAA, increments it and passes it on to the next device on the loop.

(101 111 AAAAA)

ARG Addressed Ready Group. This group of messages is primarily issued by the Controller to initiate the sending of data or status from a Talker, or to interrupt the flow of data from a Talker to a Listener. The ARG messages also include two which are issued by a Talker to notify a Controller that it is through transmitting data.

(101 01XX XXXX)

CMD Command class. The class of messages known as Commands are sent from Controllers to devices on the loop. Commands are used to control either device or interface functions.

(100 XXXX XXXX)

Controller A Controller is a device which has been assigned the function of maintaining order among all devices on the loop. There is only a single active Controller at any one time, however; control may pass between any number of Controllers on the loop.

DAB Data Byte. The Data Byte is the basic unit of data sent from a Talker to a Listener over the HP-IL bus. A Data Byte message contains data in bits D7-D0 of the message frame. The most significant bit of data (D7) is sent first. Data is normally encoded in ASCII for compatibility between different types of devices.

(00X XXXXXXXX)

DCL Device Clear. This command is issued by the Controller to cause all devices to be reset to their initial clear states. Devices need not be addressed to respond to the DCL command.

(100 00010100)

DDL Device Dependent Listener. This command is issued by the Controller to devices addressed as Listeners. A total of 32 possible DDL commands is encoded into bits D4 thru D0 of the message frame. The effect of each of the 32 DDL commands depends upon the device. DDL commands will vary from device to device.

(100 101X XXXX)

DDT Device Dependent Talker. This command is issued by the Controller to

devices addressed as Talkers. A total of 32 possible DDT commands is encoded in bits D4 thru D0 of the message frame. The effect of each of the 32 DDT commands is dependent on the device.
(100 110X XXXX)

Device A device is any physical unit. Devices and Controllers communicate with each other by sending messages over a common loop interface.

DOE Data Or End. The Data or End class of messages contain eight bits of data (bits D7-D0) usually sent from a Talker to a Listener over the loop. The End message is identical to a Data message with the exception that bit C1 is set to indicate an end of record condition.
(0XX XXXXXXXX)

EAR Enable Asynchronous Requests. This command is issued by the active Controller to allow all devices so equipped to originate their own IDY messages to indicate a request for service. Once the EAR command is in effect, any universal command other than EAR or LPD issued by a Controller will disable the ability of a device to source an asynchronous request.
(100 0001 1000)

END End Byte. The End Byte is a data byte with bit C1 set to indicate an end of record condition. The End Byte does not cause transmission from a Talker to a Listener to terminate, it merely signals a logical end condition.
(01X XXXX XXXX)

ETE End of Transmission Error. This message is issued by a Talker to notify a Controller that it is finished sending data and that an error was detected in the transmission.
(101 0100 0001)

ETO End of Transmission OK. This message is issued by a Talker to notify a Controller that it is through sending its data and that all data sent was correctly received.
(101 0100 0000)

Extended Addressing Extended Addressing allows for up to 961 devices to be addressed on the loop. The first 31 devices are assigned extended secondary addresses with the AES message. The same devices are then assigned a common primary address with the AEP message. This process continues until all devices have been assigned addresses.

GET Group Execute Trigger. This command is issued by the Controller to addressed Listeners on the loop. Receipt of the GET command by the device will cause a device function to begin operating. Functions which may be controlled in this manner are dependent upon the device.
(100 0000 1000)

GTL Go To Local. This command is used to tell all devices addressed as Listeners to accept commands from their own front panel (local) controls rather than accepting device-dependent commands via the loop interface.
(100 0000 0001)

IAA Illegal Auto-Address. This message is a form of the AAD message used

to indicate to the Controller that the number of devices on the loop is equal to or greater than 31 (the maximum allowed with simple addressing). It is simply defined as an AAD message with an address field AAAAA, equal to 31.
(101 1001 1111)

IDY Identify. This class of message is issued by the Controller to determine if a device on the loop needs servicing. Devices needing service may alert the Controller by setting bit C0 of the message frame. Additionally, if parallel polling is being used, devices may set bits D0-D7 to indicate specific service-requests.
(11X XXXX XXXX)

IEP Illegal Extended Primary. This is a form of the AEP message with the address field AAAAA equal to 31 (an illegal condition). Devices will not respond to an IEP message.
(101 1011 1111)

IES Illegal Extended Secondary. An AES message which has been incremented so that the address field AAAAA equals 31 is called the IES message. The IES message indicates to the Controller that there are more devices remaining to be addressed.
(101 1101 1111)

IFC Interface Clear. The IFC command is issued by the System Controller to all devices and Controllers on the loop. It causes all active devices to return to an idle state, but does not destroy any device addressing information previously set.
(100 1001 0000)

IMP Illegal Multiple Primary. This is a form of the AMP message with the address field equal to 31. It indicates to the Controller that the number of devices having multiple address capabilities on the loop equals or exceeds 31 (the maximum number allowed).
(101 1111 1111)

LAD Listen Address. This command is issued by a Controller to cause a device to become an active Listener on the loop. Active Listeners are allowed to receive data from the active Talker. When a device receives an LAD message with AAAAA equal to its own assigned Listen Address, it becomes an active Listener on the loop. If the device is configured for extended or multiple addressing, the LAD address field contains a primary address only. In this case the device does not become active until receiving the proper SAD command.
(100 001 AAAAA)

LAG Listen Address Group. This group of commands is used by a Controller to cause devices to become active Listeners on the interface loop.
(100 001 XXXXX)

Listener A Listener is a device which has been assigned the ability to receive data being sent over the interface loop. Listeners are enabled via the Listen Address Group (LAG) commands.

LLO Local Lockout. This command is used by the Controller to prevent a device's front panel controls from inadvertently being changed by the operator. Once an LLO command has been

accepted by a device, that device will no longer accept commands from its front panel controls.
(100 0001 0001)

LPD Loop Power-Down. This command, issued by the Controller, causes all devices on the loop, which have the capability, to go to a powered-down state. Once a device has been placed in such a state it may be reawakened by any message traveling over the interface loop. A device placed in a powered-down state consumes very little power.
(100 10011011)

Message A message is the information that is contained in a message frame. If the message is a command, bits D7-D0 may be used as part of the command. If the message contains data, bits D7-D0 contain the 8-bit data sent from Talker to Listener.

Message Frame A message frame is the logical object which travels over the loop interface from Controller to device or from Talker to Listener. A message frame is 11 bits in length. The first three bits (C2, C1, C0) determine the message class and type of message.

MLA My Listen Address. When a device receives an LAD message with the address bits AAAAA equal to its own previously assigned Listen Address, it becomes an active Listener. Active Listeners are qualified to receive data sent by an active Talker.
(100 001 AAAAA)

MSA My Secondary Address. When a device requiring a two-byte extended or multiple address receives an SAD message with the address bits AAAAA equal to its own secondary address, it becomes active.
(100 011 AAAAA)

MTA My Talk Address. When a device receives a TAD message with the address bits AAAAA equal to its own previously assigned Talk Address, it becomes an active Talker. Active Talkers are qualified to send data over the loop to any number of active Listeners.
(100 010 AAAAA)

Multiple Address Multiple addresses allow a single physical device to be allocated up to 31 unique addresses consisting of a single primary address and up to 31 secondary addresses. Multiple addressed devices are assigned addresses with the AMP and ZES messages.

NAA Next Auto-Address. An AAD message whose address has been incremented by a device is referred to as an NAA message.
(101 100 AAAAA)

NES Next Extended Secondary. An AES message whose address field AAAAA has been incremented by a device is known as the NES message.
(101 110 AAAAA)

NMP Next Multiple Primary. This message is an AMP message whose address field AAAAA has been incre-

mented by a device capable of multiple addressing.
(101 111 AAAAA)

NRD Not Ready for Data. During the transmission of data from Talker to Listener, the Controller may interrupt by replacing a data byte with the NRD message. The Talker, upon receipt of the NRD message should terminate transmission. The final byte of data is released by the Controller upon receipt of the NRD message on the loop.
(101 0100 0010)

NRE Not Remote Enable. This command is issued by the Controller to all devices on the loop. It enables a device which was previously set to accept commands via the loop interface to now begin responding to its local front panel controls. Once an NRE command is in effect, devices will no longer respond to commands received via the loop interface.
(100 1001 0011)

NUL Null Command. The Null command does not cause any operation to occur at a device (no-op). It may, however, be used in instances where loop devices must be awakened from a powered-down condition without further affecting the device. The Null command may also be used in instances where it is necessary to have a message circulated on the loop without affecting any specific device.
(100 0000 0000)

OSA Other Secondary Address. A device receiving an SAD command in which the address bits AAAAA do not match its own secondary address cannot become active. Talkers receiving OSAs will become inactive.
(100 011 AAAAA)

OTA Other Talk Address. A Talker receiving a TAD command in which the address bits AAAAA do not match its own previously assigned Talk Address will become inactive, since only one Talker may be active at any one time.
(100 010 AAAAA)

Parallel Polling Parallel polling allows a Controller to quickly identify a device needing service. The Controller sources an IDY message. Devices set preassigned bits in the IDY message to indicate service-request conditions.

Polling Polling is a method which allows the Controller to identify a device needing service. There are two methods of polling used in the HP-IL sytems: serial and parallel.

PPD Parallel Poll Disable. This command issued by the Controller causes devices which are currently addressed as Listeners to cease further parallel-poll operations.
(100 0000 0101)

PPE Parallel Poll Enable. This command is issued by a Controller to configure a device addressed as a Listener for parallel-polling operation. Bits D2-D0 are used to assign a service-request

bit to a device (000 = D0, 111 = D7). The sense of bit D3 will determine whether the device must set the parallel-poll bit when it needs service (D3 = 1) or when it does not need service (D3 = 0).
(100 1000 SBBB)

PPU Parallel Poll Unconfigure. This command issued by a Controller prevents all devices from responding to further parallel-poll operations. Devices do not have to be addressed as Listeners in order for the PPU command to apply.
(100 0001 0101)

Primary Address A primary address is the most significant byte of a two-byte address. Primary addresses are used in addressing devices which have been assigned extended addresses or have multiple address capabilities.

RDY Ready. This class of messages is normally used to control loop operations. Ready class messages include the RFC command, the Addressed Ready Group commands and the Auto-Address Group commands.
(101 XXXX XXXX)

REN Remote Enable. This is a command issued by the Controller to all devices on the loop. It enables devices to begin accepting commands via the loop interface when properly addressed. Once the REN command is in effect and the device has received the MLA message, front panel controls will have no effect on the device.
(100 1001 0010)

RFC Ready For Command. This Ready class message is issued by the active Controller to verify that a previously issued command has been executed by a device or devices on the loop. Each command issued by the Controller must be followed by an RFC message.
(101 0000 0000)

SAD Secondary Address. This command is issued by a Controller to enable those devices which require a two-byte address. A device which has received a primary address command will become active upon receipt of a SAD command in which AAAAA equals a previously assigned secondary address.
(100 011 AAAAA)

SAG Secondary Address Group. This group of commands is issued by a Controller to assign secondary addresses to those devices using two-byte, extended or multiple addressing.
(100 011X XXXX)

SAI Send Accessory ID. This message is issued by a Controller to cause the active Talker to begin sending its accessory ID. The accessory ID is usually a single byte in which the high order four bits specify the device class (for example, printer, mass storage, etc.) and the lower four bits indicate a specific device.
(101 0110 0011)

SDA Send Data. This message is issued by the Controller to allow a Talker to begin sending data.
(101 0110 0000)

SDC Selected Device Clear. When this command is issued by the Loop Controller, all devices which are addressed as Listeners will be reset to their initial clear states.
(100 0000 0100)

SDI Send Device ID. This message is issued by the Controller to cause an active Talker to begin sending its device identification. The device identification is usually an ASCII string consisting of a two-letter manufacturer's code, a five-character model number, model revision, and any additional information included by the manufacturer of the device.
(101 0110 0010)

Secondary Address A secondary address is the least significant byte of a two-byte address. Secondary addresses are used to address devices that have been assigned extended addresses or have multiple address capabilities.

Serial Polling Serial polling is the normal method polling used by a Controller to identify devices needing service. The Controller sources an IDY message. Bit C0 of the message is set by any device requesting service. In serial polling, the Controller must address each device individually to determine the source of the request.

Simple Address A simple address is a single byte identifier in the address field of a message frame. Simple addresses may specify up to 31 unique devices.

SRQ Service-Request. A service-request is an action initiated by a device to alert a Controller to the fact that it needs servicing. The service-request bit, C0 of the message frame, is turned on by the device to indicate the SRQ condition. This may occur in either a DOE or IDY class of message.
(0X1 XXXX XXXX)
or (111 XXXX XXXX)

SST Send Status. This message is issued by the Controller to tell a Talker to begin sending its status byte(s). Status information may indicate that a device needs service or may contain device-dependent information.
(101 0110 0001)

TAD Talk Address. This command is issued by a Controller to cause one device on the loop to become the active Talker. When a device receives the TAD command it checks to see if the address AAAAA matches its own assigned Talk Address. If so, it becomes the active Talker on the loop. If the device is configured for extended or multiple addressing, the address field is taken as a primary address. The device will not become active until receiving the proper SAD command.
(100 010 AAAAA)

TAG Talk Address Group. This group of commands is used by a Controller to cause a device to become the active Talker on the loop.
(100 010 XXXXX)

Talker A Talker is a device which has been assigned the ability to source data

on the interface loop. Talkers are enabled through the Talk Address Group (TAG) commands.

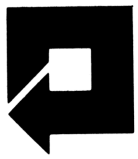
TCT Take Control. This message is issued by the active Controller to cause control to pass to another Controller on the loop.
(101 0110 0100)

UCG Universal Command Group. This group of commands is sent from a Controller to all devices on the loop. UCG commands affect all devices, not only those which have been addressed.
(100 X001 XXXX)

UNL Unlisten. This command is issued by a Controller to cause all currently addressed Listeners to go inactive.
(100 0011 1111)

UNT Untalk. This command is issued by a Controller to cause the active Talker to go inactive.
(100 0101 1111)

ZES Zero Extended Secondary. This message is issued by the Controller to assign secondary addresses to those devices capable of multiple addressing. Following the assignment of primary addresses with the AMP message the Controller issues ZES to each multiple addressed device in turn. A device receiving a ZES message will increment the low-order five bits of the ZES message to indicate to the Controller the number of addresses reserved for the device.
(101 1100 0000)



INDEX

Gerry Kane
Steve Harper
David Ushijima

THE HP-IL SYSTEM: An Introductory Guide to the Hewlett-Packard Interface Loop

Here's a definitive guide to the new Hewlett-Packard Interface Loop (HP-IL), an eight-bit serial interface designed for small low-cost, battery-operable systems. The HP-IL makes a new generation of portable systems for controlling instruments and peripherals possible.

The HP-IL links programmable calculators such as the HP-41C/CV, and the new series 80 personal computers, to low-power, low cost peripheral devices and test equipment. For those who require portability in the field it's an indispensable tool in data collection, reduction of duplicate paperwork, briefcase computation and bench instrumentation.

Co-author Steve Harper is on the Research and Development staff at Hewlett-Packard's Corvallis Division in Corvallis, Oregon. He assists Gerry Kane and David Ushijima in providing an in-depth description of the HP-IL and its functions. Contents include instructions on how to interface both HP-IL and non-HP-IL devices to the Loop. Also included are summaries of all HP-IL remote messages and message sequences.

ISBN 0-931988-77-2

5955-9425