## CHAPTER TWO: *Control by GPIB*

**In this chapter, see how**

- ***To address your Waverunner scope for GPIB***
- ***To configure GPIB software***
- ***To enable remote or local control***
- ***To make transfers of data***
- ***To make service requests***
- ***To poll Waverunner***
- ***To drive hardcopy devices***

# Talk, Listen, or Control

You can remotely control your Waverunner oscilloscope, using the General Purpose Interface Bus (GPIB). GPIB is similar to a standard computer bus. But while the computer interconnects circuit cards by means of a backplane bus, the GPIB interconnects independent devices (oscilloscopes and computers, for example) by means of a cable bus. GPIB also carries both program and interface messages.

**Program messages**, often called device dependent messages, contain programming instructions, measurement results, and oscilloscope status and waveform data.

**Interface messages** manage the bus itself. They perform functions such as initialization, addressing and "unaddressing" of devices, and the setting of remote and local modes.

### TALK, LISTEN, OR CONTROL

On the one hand, devices connected by GPIB to your Waverunner oscilloscope can be listeners, talkers, or controllers. A talker sends program messages to one or more listeners, while a controller manages the flow of information on the bus by sending interface messages to the devices. The host computer must be able to play all three roles. For details of how the controller configures the GPIB for specific functions, refer to the GPIB interface manufacturer's manual.

On the other hand, the Waverunner can be a talker or listener, but NOT a controller.

### INTERFACE

Waverunner interface capabilities include the following IEEE 488.1 definitions:

| | | | |
|---|---|---|---|
| **AH1** | Complete Acceptor Handshake | **DC1** | Complete Device Clear Function |
| **SH1** | Complete Source Handshake | **DT1** | Complete Device Trigger |
| **L4** | Partial Listener Function | **PP1** | Parallel Polling: remote configurable |
| **T5** | Complete Talker Function | **C0** | No Controller Functions |
| **SR1** | Complete Service Request Function | **E2** | Tri-state Drivers |
| **RL1** | Complete Remote/Local Function | | |

### ADDRESS

Every device on the GPIB has an address. To address Waverunner, set the remote control port to **GPIB** by means of the scope's front panel UTILITIES button and on-screen menus. If you select "RS-232" in the same way, the oscilloscope will execute over the GPIB solely "talk-only" operations, such as driving a printer. Setting Waverunner to "RS-232" enables the oscilloscope to be controlled through the RS-232-C port. See Chapter 12 of the *Operator's Manual* for how to do this.

If you address Waverunner to talk, it will remain in that state until it receives a universal untalk command (UNT), its own listen address (MLA), or another oscilloscope's talk address.

If you address Waverunner to listen, it will remain configured to listen until a universal unlisten command (UNL), or its own talker address (MTA), is received.

### GPIB SIGNALS

The GPIB bus system consists of 16 signal lines and eight ground or shield lines. The signal lines are divided into three groups:

**Data Lines:** These eight lines, usually called DI01 through DI08, carry both program and interface messages. Most of the messages use the 7-bit ASCII code, in which case DI08 is unused.

**Handshake Lines:** These three lines control the transfer of message bytes between devices. The process is called a three-wire interlocked handshake, and it guarantees that the message bytes on the data lines are sent and received without transmission error.

**Interface Management Lines:** These five lines manage the flow of information across the interface:

**ATN (ATteNtion):** The controller drives the ATN line true when it uses the data lines to send interface messages such as talk and listen addresses or a device clear (DCL) message. When ATN is false, the bus is in data mode for the transfer of program messages from talkers to listeners.

**IFC (InterFace Clear):** The controller sets the IFC line true to initialize the bus.

**REN (Remote ENable):** The controller uses this line to place devices in remote or local program mode.

**SRQ (Service ReQuest):** Any device can drive the SRQ line true to asynchronously request service from the controller. This is the equivalent of a single interrupt line on a computer bus.

**EOI (End Or Identify):** This line has two purposes: The talker uses it to mark the end of a message string. The controller uses it to tell devices to identify their response in a parallel poll (discussed later in this section).

### I/O BUFFERS

The oscilloscope has 256-byte input and output buffers. An incoming program message is not decoded before a message terminator has been received. However, if the input buffer becomes full (because the program message is longer than the buffer), the oscilloscope starts analyzing the message. In this case data transmission is temporarily halted, and the controller may generate a timeout if the limit was set too low.

### USE IEEE 488.1 STANDARD MESSAGES

The IEEE 488.1 standard specifies not only the mechanical and electrical aspects of the GPIB, but also the low-level transfer protocol. For instance, it defines how a controller addresses devices, turns them into talkers or listeners, resets them or puts them in the remote state. Such interface messages are executed with the interface management lines of the GPIB, usually with ATN true.

All these messages except GET are executed immediately upon receipt.

The command list in Part Two of this manual does not contain a command for clearing the input or output buffers, nor for setting the oscilloscope to the remote state.

***NOTE: In addition to the IEEE 488.1 interface message standards, the IEEE 488.2 standard specifies certain standardized program messages, i.e., command headers. They are identified with a leading asterisk * and are listed in the System Commands section.***

This is because such commands are already specified as IEEE 488.1 standard messages. Refer to the GPIB interface manual of the host controller as well as to its support programs, which should contain special calls for the execution of these messages.

The following description covers those IEEE 488.1 standard messages that go beyond mere reconfiguration of the bus and that have an effect on Waverunner operation.

### DEVICE CLEAR

In response to a universal Device CLear (DCL) or a Selected Device Clear message (SDC), Waverunner clears the input or output buffers, cancels the interpretation of the current command (if any) and clears pending commands. However, status registers and status-enable registers are *not* cleared. Although DCL will have an immediate effect, it can take several seconds to execute if the oscilloscope is busy.

### GROUP EXECUTE TRIGGER

The Group Execute Trigger message (GET) causes Waverunner to arm the trigger system, and is functionally identical to the *TRG command.

### REMOTE ENABLE

This interface message is executed when the controller holds the Remote ENable control line (REN) true, allowing you to configure the oscilloscope as a listener. All the front panel controls except the menu buttons are disabled. The menu indications on the right-hand side of the screen no longer appear, since menus cannot now be operated manually. Instead, the text REMOTE ENABLE appears at the top of the menu field to indicate that the oscilloscope is set in the remote mode. Whenever the controller returns the REN line to false, all oscilloscopes on the bus return to GO TO LOCAL.

When you press the GO TO LOCAL menu button, the scope returns to front panel control, unless you have placed the oscilloscope in Local LOckout (LLO) mode (see below).

The Go To Local message (GTL) causes the oscilloscope to return to local mode. All front panel controls become active and the normal menus reappear. Thereafter, whenever the oscilloscope is addressed as a listener it will be immediately reset to the remote state, except when the LLO command has been sent.

When you activate Local Lockout the scope can only be returned to its local state by returning the LLO to false. Whenever you return the oscilloscope to the remote state the local lockout mode will immediately become effective again.

The Local LOckout message (LLO) causes the GO TO LOCAL menu to disappear. You can send this message in local or remote mode. But it only becomes effective once you have set the oscilloscope in remote mode.

### INTERFACE CLEAR

The InterFace Clear message (IFC) initializes the GPIB but has no effect on the operation of the Waverunner.

***NOTE: To illustrate the GPIB programming concepts a number of examples written in BASICA are included here. It is assumed that the controller is IBM-PC compatible, running under DOS, and that it is equipped with a National Instruments GPIB interface card. Nevertheless, GPIB programming with other languages such as C or Pascal is quite similar. If you're using another type of computer or GPIB interface, refer to the interface manual for installation procedures and subroutine calls.***

### CONFIGURE THE GPIB DRIVER SOFTWARE

1. Verify that the GPIB interface is properly installed in the computer. If it is not, follow the interface manufacturer's installation instructions. In the case of the National Instruments interface, it is possible to modify the base I/O address of the board, the DMA channel number, and the interrupt line setting using switches and jumpers. In the program examples below, default positions are assumed.
2. Connect Waverunner to the computer with a GPIB interface cable.
3. Set the GPIB address to the required value. The program examples assume a setting of **4**.

The host computer requires an interface driver that handles the transactions between the operator's programs and the interface board.

In the case of the National Instruments interface, the installation procedure will:

a. Copy the GPIB handler GPIB.COM into the boot directory.

b. Modify the DOS system configuration file CONFIG.SYS to declare the presence of the GPIB handler.

c. Create a sub-directory called GPIB-PC, and install in GPIB-PC a number of files and programs useful for testing and reconfiguring the system, and for writing user programs.

The following files in the sub-directory GPIB-PC are particularly useful:

**IBIC.EXE** allows interactive control of the GPIB by means of functions entered at the keyboard. Use of this program is highly recommended to anyone unfamiliar with GPIB programming or with Waverunner's remote commands.

**DECL.BAS** is a declaration file that contains code to be included at the beginning of any BASICA application program. Simple application programs can be quickly written by appending the operator's instructions to DECL.BAS and executing the complete file.

**IBCONF.EXE** is an interactive program that allows inspection or modification of the current settings of the GPIB handler. To run IBCONF.EXE, refer to the National Instruments manual.

***NOTE: In the program examples in this section, it is assumed that the National Instruments GPIB driver GPIB.COM is in its default state, i.e., that the user has not modified it with IBCONF.EXE. This means that the interface board can be referred to by the symbolic name 'GPIB0' and that devices on the GPIB bus with addresses between 1 and 16 can be called by the symbolic names 'DEV1' to 'DEV16'. If you have a National Instruments PC2 interface card rather than PC2A, you must run IBCONF to declare the presence of this card rather than the default PC2A.***

### MAKE SIMPLE TRANSFERS

For a large number of remote control operations it is sufficient to use just three different subroutines (IBFIND, IBRD and IBWRT) provided by National Instruments. The following complete program reads the timebase setting of Waverunner and displays it on the terminal:

```
1–99    <DECL.BAS>
100     DEV$="DEV4"
110     CALL IBFIND(DEV$,SCOPE%)
120     CMD$="TDIV?"
130     CALL IBWRT(SCOPE%,CMD$)
140     CALL IBRD(SCOPE%,RD$)
150     PRINT RD$
160     END
```

Lines 1–99 are a copy of the file DECL.BAS supplied by National Instruments. The first six lines are required for the initialization of the GPIB handler. The other lines are declarations which may be useful for larger programs, but are not really required code. The sample program above only uses the strings `CMD$` and `RD$`, which are declared in DECL.BAS as arrays of 255 characters.

Lines 100 and 110 open the device `DEV4` and associate with it the descriptor `SCOPE%`. All I/O calls after that will refer to `SCOPE%`. The default configuration of the GPIB handler recognizes `DEV4` and associates with it a device with the GPIB address 4.

Lines 120 and 130 prepare the command string `TDIV?` and transfer it to the oscilloscope. The command instructs the oscilloscope to respond with the current setting of the timebase.

Lines 140 and 150 read the response of the oscilloscope and place it into the character string RD$.

Line 170 displays the response on the terminal.

> **NOTE:** DECL.BAS requires access to the file BIB.M during the GPIB initialization. BIB.M is one of the files supplied by National Instruments, and it must exist in the directory currently in use.

The first two lines of DECL.BAS both contain a string XXXXX, which must be replaced by the number of bytes that determine the maximum workspace for BASICA (computed by subtracting the size of BIB.M from the space currently available in BASICA). For example, if the size of BIB.M is 1200 bytes, and when BASICA is loaded it reports "60200 bytes free," you should replace "XXXXX" by the value 59 000 or less.

When running this sample program, Waverunner will automatically be set to the remote state when IBWRT is executed, and will remain in that state. Pressing the LOCAL menu button will return Waverunner to local mode if the GPIB handler was modified to inhibit Local LOckout (LLO). Here is a slightly modified version of the sample program that checks if any error occurred during GPIB operation:

```
1-99     <DECL.BAS>
100      DEV$="DEV4"
110      CALL IBFIND(DEV$,SCOPE%)
120      CMD$="TDIV?"
130      CALL IBWRT(SCOPE%,CMD$)
140      IF ISTA% < 0 THEN GOTO 200
150      CALL IBRD(SCOPE%,RD$)
160      IF ISTA% < 0 THEN GOTO 250
170      PRINT RD$
180      IBLOC(SCOPE%)
190      END
200      PRINT "WRITE ERROR =";IBERR%
210      END
250      PRINT "READ ERROR =";IBERR%
260      END
```

The GPIB status word ISTA%, the GPIB error variable IBERR% and the count variable IBCNT% are defined by the GPIB handler and are updated with every GPIB function call. Refer to the National Instruments manual for details. The sample program above would report if the GPIB address of the oscilloscope was set to a value other then 4. Line 180 resets the oscilloscope to local with a call to the GPIB routine IBLOC.

### USE ADDITIONAL DRIVER CALLS

- **IBLOC** is used to execute the IEEE 488.1 standard message Go To Local (GTL), i.e. it returns the oscilloscope to the local state. The programming example above illustrates its use.
- **IBCLR** executes the IEEE 488.1 standard message Selected Device Clear (SDC).
- **IBRDF** and **IBWRTF**, respectively, allow data to be read from GPIB to a file, and written from a file to GPIB. Transferring data directly to or from a storage device does not limit the size of the data block, but may be slower than transferring to the computer memory.
- **IBRDI and IBWRTI** allow data to be read from GPIB to an integer array, and written from integer array to GPIB. Since the integer array allows storage of up to 64 kilobytes (in BASIC), IBRDI and IBWRTI should be used for the transfer of large data blocks to the computer memory, rather than IBRD or IBWRT, which are limited to 256 bytes by the BASIC string length. Note that IBRDI and IBWRTI only exist for BASIC, since for more modern programming languages, such as C, the functions called IBRD and IBWRT are far less limited in data block size.
- **IBTMO** can be used to change the timeout value during program execution. The default value of the GPIB driver is 10 seconds — for example, if the oscilloscope does not respond to an IBRD call, IBRD will return with an error after the specified time.
- **IBTRG** executes the IEEE 488.1 standard message Group Execute Trigger (GET), which causes Waverunner to arm the trigger system.

National Instruments supply a number of additional function calls. In particular, it is possible to use the so-called board level calls, which allow a very detailed control of the GPIB.

***NOTE: The SRQ bit is latched until the controller reads the STatus Byte Register (STB). The action of reading the STB with the command *STB? clears the register contents except the MAV bit (bit 4) until a new event occurs. Service requesting can be disabled by clearing the SRE register with the *SRE 0 command.***

### MAKE SERVICE REQUESTS

When a Waverunner is used in a remote application, events often occur asynchronously, i.e., at times that are unpredictable for the host computer. The most common example of this is a trigger wait after the oscilloscope is armed: the controller must wait until the acquisition is finished before it can read the acquired waveform. The simplest way of checking if a certain event has occurred is by either continuously or periodically reading the status bit associated with it until the required transition is detected. Continuous status bit polling is described in more detail below. For a complete explanation of status bits refer to Chapter 5.

Perhaps a more efficient way of detecting events occurring in the oscilloscope is the use of the Service Request (SRQ). This GPIB interrupt line can be used to interrupt program execution in the controller. The controller can then execute other programs while waiting for the oscilloscope. Unfortunately, not all interface manufacturers support the programming of interrupt service routines. In particular, National Instruments supports only the SRQ bit within the ISTA% status word. This requires you to continuously or periodically

check this word, either explicitly or with the function call IBWAIT. In the absence of real interrupt service routines the use of SRQ may not be very advantageous.

In the default state, after power-on, the Service ReQuest is disabled. You enable SRQ by setting the Service Request Enable register with the command "*SRE" and by specifying which event should generate an SRQ. Waverunner will interrupt the controller as soon as the selected event(s) occur by asserting the SRQ interface line. If several devices are connected to the GPIB, you may be required to identify which oscilloscope caused the interrupt by serial polling the various devices.

**Example:** To assert SRQ in response to the events "new signal acquired" or "return-to-local" (pressing the soft key/menu button for GO TO LOCAL).These events are tracked by the INR register, which is reflected in the SRE register as the INB summary bit in position 0. Since bit position 0 has the value 1, the command `*SRE 1` enables the generation of SRQ whenever the INB summary bit is set.

In addition, the events of the INR register that may be summarized in the INB bit must be specified. The event "new signal acquired" corresponds to INE bit 0 (value 1) while the event "return-to-local" is assigned to INE bit 2 (value 4). The total sum is 1 + 4 = 5. Thus the command `INE 5` is needed:

```
CMD$="INE 5;*SRE 1"
CALL IBWRT(SCOPE%,CMD$)
```

**Example:** To assert SRQ when soft key 4 (fourth menu button from top of screen) is pressed. The event "soft key 4 pressed" is tracked by the URR register. Since the URR register is not directly reflected in STB but only in the ESR register (URR, bit position 6), the ESE enable register must be set first with the command `*ESE 64` to allow the URQ setting to be reported in STB. An SRQ request will now be generated provided that the ESB summary bit (bit position 5) in the SRE enable register is set (`*SRE 32`):

```
CMD$="*ESE 64;*SRE 32"
CALL IBWRT(SCOPE%,CMD$)
```

***NOTE: The term "soft-key," used here in reference to remote operations, is synonymous with "menu button," used in the accompanying Operator's Manual to mean front panel operations. Both terms refer to the column of seven buttons running parallel to the screen on the Waverunner front panel and the menu functions they control.***

# Take Instrument Polls

You can regularly monitor state transitions within the oscilloscope by polling selected internal status registers. There are four basic polling methods you can use to detect the occurrence of a given event: continuous, serial, parallel, and `*IST`. By far the simplest of these is continuous polling. The others are appropriate only when interrupt-service routines (servicing the SRQ line) are supported, or multiple devices on GPIB require constant monitoring. To emphasize the differences between the methods, described below, the same example (determining whether a new acquisition has taken place) is used in each case.

### DO CONTINUOUS POLLING

A status register is continuously monitored until a transition is observed. This is the most straightforward method for detecting state changes, but may not be practical in certain situations, especially with multiple device configurations.

In the following example, the event "new signal acquired" is observed by continuously polling the INternal state change Register (INR) until the corresponding bit (in this case bit 0, i.e., value 1) is non-zero, indicating a new waveform has been acquired. Reading INR clears this at the same time, so that there is no need for an additional clearing action after a non-zero value has been detected. The command `CHDR OFF` instructs the oscilloscope to omit any command headers when responding to a query, simplifying the decoding of the response. The oscilloscope will then send "1" instead of "INR 1":

```
CMD$="CHDR OFF"
CALL IBWRT(SCOPE%,CMD$)
MASK% = 1'New Signal Bit has value 1'
LOOP% = 1
WHILE LOOP%
CMD$="INR?"
CALL IBWRT(SCOPE%,CMD$)
CALL IBRD(SCOPE%,RD$)
NEWSIG% = VAL(RD$) AND MASK%
IF NEWSIG% = MASK% THEN LOOP% = 0
WEND
```

### TAKE A SERIAL POLL

Serial polling takes place once the SRQ interrupt line has been asserted, and is only advantageous when you are using several oscilloscopes at once. The controller finds which oscilloscope has generated the interrupt by inspecting the SRQ bit in the STB register of each. Because the service request is based on an interrupt

mechanism, serial polling offers a reasonable compromise in terms of servicing speed in multiple-device configurations.

In the following example, the command `INE 1` enables the event "new signal acquired" to be reported in the INR to the INB bit of the status byte STB. The command `*SRE 1` enables the INB of the status byte to generate an SRQ whenever it is set. The function call IBWAIT instructs the computer to wait until one of three conditions occurs: &H8000 in the mask (MASK%) corresponds to a GPIB error, &H4000 to a timeout error, and &H0800 to the detection of RQS (ReQuest for Service) generated by the SRQ bit.

Whenever IBWAIT detects RQS it automatically performs a serial poll to find out which oscilloscope generated the interrupt. It will only exit if there was a timeout or if the oscilloscope (`SCOPE%`) generated SRQ. The additional function call `IBRSP` fetches the value of the status byte, which may be further interpreted. For this to work properly the value of "Disable Auto Serial Polling" must be set to "off" in the GPIB handler (use `IBCONF.EXE` to check):

```
CMD$="*CLS; INE 1; *SRE 1"
CALL IBWRT(SCOPE%,CMD$)
MASK% = &HC800
CALL IBWAIT(SCOPE%,MASK%)
IF (IBSTA% AND &HC000) <> 0 THEN PRINT "GPIB or Timeout Error" : STOP
CALL IBRSP(SCOPE%,SPR%)
PRINT "Status Byte =.", SPR%
```

Board-level function calls can deal simultaneously with several oscilloscopes attached to the same interface board. Refer to the National Instruments manual.

***NOTE: After the serial poll is completed, the RQS bit in the STB status register is cleared. Note that the other STB register bits remain set until they are cleared by means of a "*CLS" command or the oscilloscope is reset. If these bits are not cleared, they cannot generate another interrupt.***

**DO A PARALLEL POLL**

Like serial polling, this is only useful with several oscilloscopes. The controller simultaneously reads the Individual STatus bit (IST) of all oscilloscopes to determine which one needs service. This method allows up to eight different oscilloscopes to be polled at the same time.

When a parallel poll is initiated, each oscilloscope returns a status bit over one of the DIO data lines. Devices may respond either individually, using a separate DIO line, or collectively on a single data line. Data-line assignments are made by the controller using a Parallel Poll Configure (PPC) sequence.

In the following example, the command `INE 1` enables the event "new signal acquired" in the INR to be reported to the INB bit of the status byte STB. The PaRallel poll Enable register (PRE) determines which events will be summarized in the IST status bit. The command `*PRE 1` enables the INB bit to set the IST bit

whenever it is itself set. Once parallel polling has been established, the parallel-poll status is examined until a change on data bus line DI02 takes place.

**Stage 1**

1. Enable the INE and PRE registers
2. Configure the controller for parallel poll
3. Instruct Waverunner to respond on data line 2 (DI02) with these commands:

```
CMD1$="?_@$"
CALL IBCMD(BRD0%,CMD1$)
CMD$="INE 1;*PRE 1"
CALL IBWRT(BRD0%,CMD$)
CMD4$=CHR$(&H5)+CHR$(&H69)+"?"
CALL IBCMD(BRD0%,CMD4$)
```

**Stage 2**

4. Parallel poll the oscilloscope until DI02 is set with these commands:

```
LOOP% = 1
WHILE LOOP%
CALL IBRPP(BRD0%,PPR%)
IF (PPR% AND &H2) = 2 THEN LOOP% = 0
WEND
```

**Stage 3**

5. Disable parallel polling (hex 15) and clear the parallel poll register with these commands:

```
CMD5$=CHR$(&H15)
CALL IBCMD(BRD0%,CMD5$)
CALL IBCMD(BRD0%,CMD1$)
CMD$="*PRE 0"CALL IBWRT(BRD0%,CMD$):
```

In the above example, board-level GPIB function calls are used. It is assumed that the controller (board) and Waverunner (device) are respectively located at addresses 0 and 4.

The listener and talker addresses for the controller and Waverunner are:

| LOGIC DEVICE | LISTENER ADDRESS | TALKER ADDRESS |
|---|---|---|
| External Controller | 32 ASCII<space>) | 64 (ASCII @) |
| Waverunner | 32+4=36 (ASCII $) | 64+4=68 (ASCII D) |

### PERFORM AN *IST POLL

You can also read the state of the Individual STatus bit (IST) returned in parallel polling by sending the `*IST?` query. To enable this poll mode, you must intialize Waverunner as for parallel polling by writing into the PRE register. Since `*IST` emulates parallel polling, apply this method wherever parallel polling is not supported by the controller. In the following example, the command `INE 1` enables the event "new signal acquired" in the INR to be reported to the INB bit of the status byte STB. The command `*PRE 1` enables the INB bit to set the IST bit whenever it is set. The command `CHDR OFF` suppresses the command header in the oscilloscope's response, simplifying the interpretation. The status of the IST bit is then continuously monitored until set by the oscilloscope:

```
CMD$="CHDR OFF; INE 1; *PRE 1"
CALL IBWRT(SCOPE%,CMD$)
LOOP% = 1
WHILE LOOP%
CMD$="*IST?"
CALL IBWRT(SCOPE%,CMD$)
CALL IBRD(SCOPE%,RD$)
IF VAL(RD$) = 1 THEN LOOP% = 0
WEND
```

***NOTE: The characters "?" and "_" appearing in the command strings stand for unlisten and untalk respectively. They are used to set the devices to a "known" state. To shorten the size of the program examples, device talking and listening initialization instructions have been grouped into character chains. They are:*** `CMD1$ = "?_@$" Unlisten, Untalk, PC talker, DSO listener.` ***The remote message code for executing a parallel response in binary form is 01101PPP, where PPP specifies the data line. Because data line 2 is selected, the identification code is 001, which results in the code 01101001 (binary) or &H69 (hex). See Table 38 of the IEEE 488-1978 Standard for further details.***

# Drive Hard-copy Devices on the GPIB

You can interface your Waverunner oscilloscope with a wide range of hard-copy devices, such as printers and plotters, and copy the screen contents to them. List the devices supported using the command `HARDCOPY_SETUP`.

With a hard-copy device connected to the GPIB, you can use either of two basic configurations. When only Waverunner and a hardcopy device such as a printer are connected, you must configure the oscilloscope as talker-only, and the hardcopy device as listener-only, to ensure proper data transfer. However, when an external controller is connected to the GPIB, you must use this controller to supervise the data transfers. You can then use a variety of schemes to transfer Waverunner screen contents.

Configure Waverunner as talker-only with its front panel controls. The hardcopy device manufacturer usually specifies an address that forces the oscilloscope into listening mode, and you can select this as well as the other necessary settings using the same menus. See Chapter 6, "Document Your Work" of the *Operator's Manual*.

Use the following schemes for driving hard copy devices by remote control using GPIB.

### READ DATA BY CONTROLLER

The controller reads the data into internal memory, then sends them to the printer. You can arrange this with simple high-level GPIB function calls. The controller stores the full set of printer instructions and afterwards sends them to the graphics device. This method is the most straightforward way to transfer screen contents, but requires a large amount of buffer storage:

```
CMD$ = "SCDP"
CALL IBWRT(SCOPE%,CMD$)
FILE$="PRINT.DAT"
CALL IBRDF(SCOPE%,FILE$)
CALL IBWRTF(PRINTER%,FILE$)
```

### SEND DATA TO BOTH

Waverunner sends data to both controller and printer. The oscilloscope puts the printer instructions onto the bus. The data is directly put out and saved in scratch memory in the controller. The contents of the scratch file can be deleted later:

**Stage 1:** Controller talker, Waverunner listener.

1. Issue the screen dump command

   ```
   CMD1$="? @$": CALL IBCMD(BRD0%,CMD1$)
   CMD$="SCDP": CALL IBWRT(BRD0%,CMD$)
   ```

**Stage 2:** Waverunner talker, controller and printer listeners.

2. Print data while storing data in scratch file SCRATCH.DAT with the commands

```
CMD2$="? D%": CALL IBCMD(BRD0%,CMD2$)
FILE$="SCRATCH.DAT": CALL IBRDF(BRD0%,FILE$)
```

### TALK DIRECTLY TO PRINTER

a. The controller goes into a standby state.

b. Waverunner becomes a talker and sends data directly to the printer.

c. The controller goes into standby and resumes GPIB operations once the data have been printed, i.e., when an EOI is detected:

**Stage 1:** Controller talker, Waverunner listener.

1. Issue the screen dump command

```
CMD1$="?_@$": CALL IBCMD(BRD0%,CMD1$)
CMD$="SCDP": CALL IBWRT(BRD0%,CMD$)
```

**Stage 2:** Waverunner talker, printer or plotter listener.

2. Put controller in standby:

```
CMD2$="?_D%": CALL IBCMD(BRD0%,CMD2$)
V%=1: CALL IBGTS(BRD0%,V%)
```

In the second and third schemes presented above, board-level GPIB function calls are used. It is assumed that the controller (board), Waverunner and the printer are respectively located at addresses 0, 4, and 5.

The listener and talker addresses for the controller, Waverunner, and printer are as follows:

| LOGIC DEVICE | LISTENER ADDRESS | TALKER ADDRESS |
|---|---|---|
| Controller | 32 (ASCII<space>) | 64 (ASCII @) |
| Waverunner | 32+4=36 (ASCII $) | 64+4=68 (ASCII D) |
| Printer | 32+5=37 (ASCII %) | 64+5=69 (ASCII E) |

The characters "?" and "_" appearing in the command strings stand for unlisten and untalk respectively. They are used to set the devices to a "known" state.

To shorten the size of the program examples, device talking and listening initialization instructions have been grouped into character chains. They are:

`CMD1$ = "?_@$"` Unlisten, Untalk, Controller talker, Waverunner listener

`CMD2$ = "?_ D"` Unlisten, Untalk, Controller listener, Waverunner talker