

## **CHAPTER TWO: *Control by GPIB***

In this chapter, see how to

- *Address your WaveMaster scope for GPIB*
- *Configure GPIB software*
- *Enable remote and local control*
- *Make transfers of data*
- *Make service requests*
- *Poll WaveMaster*
- *Set timing and synchronization*



## Talk, Listen, or Control

You can control your WaveMaster DSO remotely, 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, 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.

On the one hand, devices connected by GPIB to your WaveMaster DSO 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 WaveMaster DSO can be a talker or listener, but NOT a controller.

Much of the material in this chapter is general to all GPIB systems, but where detailed instructions and program fragments are provided in this manual, they are based on National Instruments hardware and software, and on some form of BASIC language. Where INCLUDES are mentioned, this points to the need to couple the programming language to the GPIB by including some drivers. The National Instruments manuals explain this. Variables ending with % are integers, and variables ending with \$ are strings, in accordance with the practice in some BASIC languages. The entire system is of course compatible with any hardware and software based on IEEE-488.2, and any programming language can be used if it can be linked to GPIB.

INTERFACE

WaveMaster DSO 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 the WaveMaster DSO, set the remote control port to **GPIB** by means of the scope's front panel UTILITIES button and on-screen menus.

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

It is wise to use the general Unlisten and Untalk commands before setting up the talker and listener states, to avoid conflicts.

The following characters are used in GPIB to control talking and listening –

ASCII 63 = ?	General Unlisten
ASCII 95 = _	General Untalk
ASCII 32 = Space	Base Listen Address
ASCII 64 = @	Base Talk Address

To make an actual talk address and listen address we have to add the GPIB address to the ASCII values of the base characters, to give the ASCII value of the new character. So a string of these commands looks like a random set of characters. Using named variables makes programs easier to understand. For example, if we have a DSO at GPIB address 4, and a PC at address 4, we construct the command strings as follows, for use later in the program.

```
UnListen$ = Chr$ (63) : UnTalk$ = Chr$(95)
BaseListen% = 32 : BaseTalk% = 64 : DSOAddress% = 4
DSOListen$ = Chr$ (BaseListen% + DSOAddress%)
DSOTalk$ = Chr$ (BaseTalk% + DSOAddress%)
```

If the PC is at address 0, we can also write

```
PCTalk$ = Chr$ (BaseTalk%) : PCListen$ = Chr$(BaseListen%)
```

Finally –

```
DSOListenPCTalk$ = UnListen$ + UnTalk$ + PCTalk$ + DSOListen$
DSOTalkPCListen$ = UnListen$ + UnTalk$ + PCListen$ + DSOTalk$
```

These last two strings, once defined, can be used in programs for sending to the DSO.

### **GPIB SIGNALS**

The GPIB 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 DIO1 through DIO8, carry both program and interface messages. Most of the messages use the 7-bit ASCII code, in which case DIO8 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).

## PART ONE: ABOUT REMOTE CONTROL

---

### 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 WaveMaster DSO operation.

### DEVICE CLEAR

In response to a universal Device Clear (DCL) or a Selected Device Clear message (SDC), the WaveMaster DSO 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 the WaveMaster DSO to arm the trigger system, and is functionally identical to the \*TRG command.

### REMOTE ENABLE

WaveMaster DSOs do not lock out any local controls when placed in the remote state, or in RWLS. It always accepts remote as well as local control inputs (unless you turn off remote control capability in Utilities → Remote.)

### INTERFACE CLEAR

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

**NOTE:** To illustrate the GPIB programming concepts, a number of examples are included here, written in a similar way to BASIC. It is assumed that the controller is IBM-PC compatible, 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.

**This procedure refers to the installation and configuration of a GPIB card under the DOS operating system. More recent operating systems (Windows 95, 98, ME, NT, 2000, XP, etc) generally use 'Plug 'n' Play' GPIB drivers, which are configured using an icon in the control panel.**

### CONFIGURE THE GPIB DRIVER SOFTWARE

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.

Connect the WaveMaster DSO to the computer with a GPIB interface cable.

Set the GPIB address to the required value. The program examples assume a setting of 4.

## PART ONE: ABOUT REMOTE CONTROL

---

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:

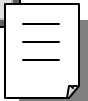
- Copy the GPIB handler GPIB.COM into the boot directory.
- Modify the DOS system configuration file CONFIG.SYS to declare the presence of the GPIB handler.
- Create a sub-directory called GPIB-PC, and install in GPIB-PC a number of files and programs useful for testing and reconfiguring the system and 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 the WaveMaster DSO's remote commands.

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

**NOTE:** *In the program examples in this section, it is assumed that the National Instruments (NI) GPIB driver GPIB.COM is in its default state, i.e., that you have 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. Later boards from National Instruments, and boards from other vendors, will require their own software, though NI has achieved good compatibility with its earlier systems, and older software will often work with newer boards.*



### 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 the WaveMaster DSO and displays it on the terminal:

```
GPIB:      This line holds the INCLUDE for the GPIB routines

Find:      DEV$ = "DEV4"                '   Because the DSO has been set at
          address 4.

          CALL IBFIND (DEV$, SCOPE%) '   Find the DSO: label it "SCOPE%".

Send:      CMD$ = "TDIV?"              '   Make a query string about the
          time base speed.

          CALL IBWRT (SCOPE%, CMD$)    '   Send the string to the DSO.

Read:      CALL IBRD (SCOPE%, RD$)     '   Read the response from the DSO.

          PRINT RD$                   '   Print the response string.

          END
```

### Explanation

**GPIB:** This line or lines must hold the link between the programming language and the National Instruments GPIB functions and drivers.

**Find:** 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.

**Send:** 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.

**Read:** Read the response of the oscilloscope and place it into the character string RD\$.

## PART ONE: ABOUT REMOTE CONTROL

---

When running this sample program, the WaveMaster DSO will automatically be set to the remote state when IBWRT is executed, and will remain in that state. Here is a slightly modified version of the sample program that checks if any error occurred during GPIB operation:

```
GPIB:      ` This line should hold the INCLUDE for the GPIB routines
Address:   DEV$ = "DEV4"
Find:      CALL IBFIND (DEV$, SCOPE%)      ` Find the DSO.
Send:      CMD$ = "TDIV?"      ` Time base query
          CALL IBWRT (SCOPE%, CMD$)      ` Send the string to the DSO.
ErrorS:    IF IBSTA% < 0 THEN PRINT "WRITE ERROR =" ; IBERR% : END
Read:      CALL IBRD (SCOPE%, RD$)      ` Try to read a string from the
          DSO.
ErrorR:    IF IBSTA% < 0 THEN PRINT "READ ERROR =" ; IBERR% : END
          PRINT RD$

190      END
```

The GPIB status word IBSTA%, the GPIB error variable IBERR% and the count variable IBCNT% are defined by the GPIB handler and are updated with every GPIB function call. IBSTA% is negative if there is an error, while IBERR% shows what type of error has occurred. IBCNT% is the number of bytes transferred. Refer to the National Instruments manual for details. The sample program above would report an error if the GPIB address of the oscilloscope was set to a value other than 4. When you are sending remote commands to the DSO, the IBSTA% and IBERR% don't necessarily indicate that the scope accepted the string, but merely that the string was correctly transmitted to the the DSO to interpret. To ensure that commands were valid, and weren't rejected by the DSO, use the Remote Control Assistant.

### 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 the WaveMaster DSO to arm the trigger system.

National Instruments supplies 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 SStatus 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 WaveMaster DSO 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. The WaveMaster DSO 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 “new signal acquired.” This event is 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$)
```

## 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 that 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$)
MASK% = 1  ' New Signal Bit has value 1
DO
    CMD$ = "INR?"
    CALL IBWRT (SCOPE%, CMD$)
    CALL IBRD (SCOPE%, RD$)
    NEWSIG% = VAL (RD$) AND MASK%
LOOP UNTIL NEWSIG% = MASK%
```

### 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 when several oscilloscopes are connected. 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 DIO2 takes place.

**Stage 1**

1. Enable the INE and PRE registers
2. Configure the controller for parallel poll
3. Instruct the WaveMaster DSO to respond on data line 2 (DIO2) with these commands:

```
CMD1$ = DSOListenPCTalk$           ` As defined earlier
CALL IBCMD (BRD0%, CMD1$)
CMD$ = "INE 1;*PRE 1"
CALL IBWRT (BRD0%, CMD$)
PPE$ = Chr$ (&H5)                  ` GPIB Parallel Poll Enable
MSA9$ = Chr$ (&H69)                 ` GPIB Secondary Address 9
CMD4$ = PPE$ + MSA9$ + UnListen$
CALL IBCMD (BRD0%, CMD4$)
```

**Stage 2**

4. Parallel poll the oscilloscope until DIO2 is set with these commands:

```
Do
CALL IBRPP (BRD0%, PPR%)
Loop Until (PPR% AND &H2) = 2
```

**Stage 3**

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

```
PPU$ = Chr$ (&H15)                 ` GPIB Parallel Poll
Unconfigure
CALL IBCMD (BRD0%, PPU$)
CALL IBCMD (BRD0%, CMD1$)           ` As defined earlier
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 the WaveMaster DSO (device) are located at addresses 0 and 4, respectively.

PART ONE: ABOUT REMOTE CONTROL

The listener and talker addresses for the controller and the WaveMaster DSO are:

LOGIC DEVICE	LISTENER ADDRESS	TALKER ADDRESS
External Controller	32 (ASCII<space>)	64 (ASCII @)
WaveMaster DSO	32 + 4 = 36 (ASCII \$)	64 + 4 = 68 (ASCII D)

PERFORM AN \*IST POLL

You can also read the state of the Individual SStatus bit (IST) returned in parallel polling by sending the \*IST? query. To enable this poll mode, you must initialize the WaveMaster DSO 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$)
DO
    CMD$ = "*IST?"
    CALL IBWRT (SCOPE%, CMD$)
    CALL IBRD (SCOPE%, RD$)
LOOP UNTIL VAL (RD$) = 1
```

## Timing and Synchronization

Depending on how your remote program is written, it may be affected by timing changes between different DSO series, even between Waverunner DSOs and WavePro DSOs. In WaveMaster DSOs, these effects may be even more pronounced than in previous scopes, for several reasons. Firstly, WaveMaster DSOs are faster than our earlier scopes. Secondly, WaveMaster DSOs support faster interfaces. That is, the standard network interface is 100BaseT instead of 10BaseT. Secondly, and more significantly, for the most part, our earlier scope series processed remote commands sequentially. That is, they would not start executing any command until execution of the previous one had finished. This meant that many operations were automatically synchronous by default, and remote control programs which did not use status bytes or \*OPC?, may have worked "by luck". That is not the case in WaveMaster DSOs. Since they use multitasking, you must be much more diligent in programming.

Most timing and synchronization problems are related to changing acquisitions, or the completion of analysis after an acquisition occurs. For example, if you change the offset of channel 1 while the scope is in Auto trigger mode, and then you use the PAVA? query to read a parameter computed on channel 1, in the older scopes, you would almost always get the results after the data has been acquired with the new offset. However, in WaveMaster DSOs, the processing is overlapped with the next acquisition and, as a consequence, the PAVA? result may have come from the acquisition prior to the offset change.

There are several ways of ensuring that your program gives the correct results when controlling the scope remotely. To simplify the synchronization issue, in most cases you can put the scope into single trigger mode. Then you can use either the status registers available in the scope, or the \*OPC? query and the WAIT command to detect completion when the acquisition and any processing are done.

Note that when you arm the scope by sending the TRMD SINGLE command, the scope will automatically perform any necessary calibrations before actually starting to acquire data. These calibrations may take several seconds, so if you query the status immediately after sending TRMD SINGLE, you need to have the GPIB (or remote) timeout set to be at least 10 seconds to prevent a timeout before getting the correct results.

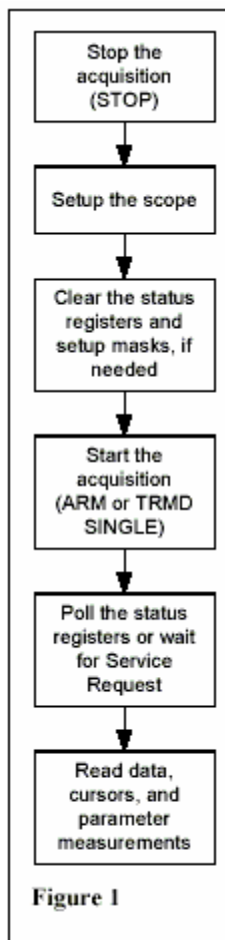
Calibrations are performed if your program changes some control settings (e.g., volts/div, number of active channels, etc.) or if the temperature of the scope has changed significantly. You can disable the calibrations by sending the AUTO\_CALIBRATE OFF command. However, the scope performance may be degraded if the temperature changes and it does not get a chance to self calibrate. A calibration of the WaveMaster DSO can be "forced" by issuing a \*CAL? command. This technique allows you to control the timing of calibrations so that they will not interfere with the acquisition of important data.

One case when you may need to use "normal" or "auto" trigger mode is the accumulation of many acquisitions for functions such as averaging or histogramming. In this case, it is best to stop the acquisitions, set up the scope, and then set the trigger mode to NORMAL to acquire the data. (A possible alternative is to use sequence mode. It is faster, but does require that you know how many acquisitions to accumulate. That number can be specified and captured in sequence mode).

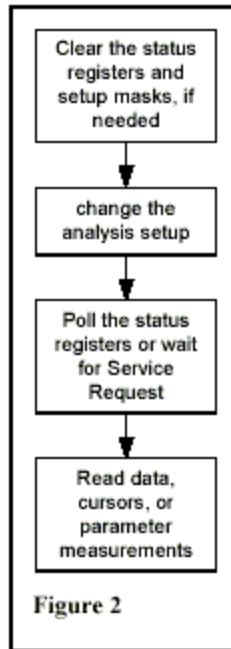
### STATUS REGISTERS

Status registers store a record of events and conditions that occur inside the DSO. Some of the events recorded are: New data has been acquired; Processing has completed; Hardcopy has completed; An error has occurred; etc. The programmer can use the registers to sense the condition of the instrument by polling them until the desired status bit has been set. A status register can be polled by querying its associated remote command (e.g., \*STB, INR?, \*ESR, etc.). Alternatively, with GPIB, the scope can request service from the controller by using the mask registers to select the events of interest.

The following diagram (Figure 1) shows the steps necessary to acquire data using the status registers for synchronization.



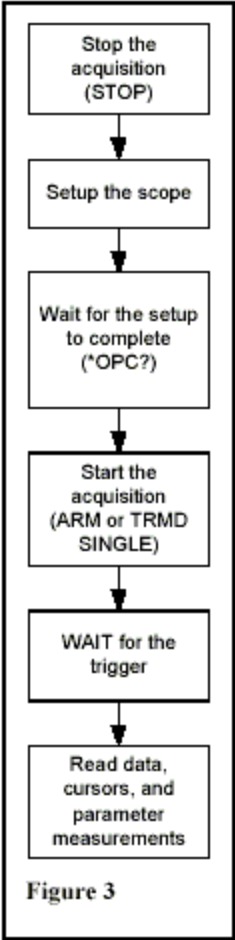
If the data have already been acquired and you want to do further analysis (math, parameters, cursors) on it, you can proceed as shown in Figure 2:



For more details on Status registers, see Chapter 5.

### **SYNCHRONIZING WITH \*OPC? AND WAIT**

The \*OPC? query returns a 1 when the previous commands have finished. Therefore, you can use this query with the WAIT command to synchronize the scope with your controller, using the steps shown in Figure 3 below. The WAIT command waits for the acquisition to complete, but it does not wait for the processing. The WAIT command allows you to specify an optional timeout so that if the scope does not trigger, your program will not hang. However, if you use the timeout it is strongly advised to subsequently check the status registers to ensure that the scope actually triggered and that any processing has completed.



§ § §

BLANK PAGE