ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Keysight / Agilent 16542A* at our website: ***Click HERE***

Programmer's Guide

Publication number 16542-90902
First Edition September 1992

©Copyright Hewlett-Packard Company 1992
All Rights Reserved

HP 16542A 2-Mbyte, 100-MHz State/Timing Logic Analyzer

In This Guide

This HP 16542A programmer's guide is organized in 14 chapters. Chapter 1 contains, general information about programming the HP 16542A analyzer, frequently used mainframe system commands, and the HP 16542A command tree. If you are unfamiliar with programming a logic analyzer, you should read this chapter.

Chapters 2 through 11 explain the commands in the command set for the logic analyzer. These chapters are organized in subsystems with each subsystem representing a front-panel menu.

Chapter 12 contains information on the SYSTem:DATA and SYSTem:SETup commands for this module. These commands transfer data and setup configuration to and from the logic analyzer.

Chapter 13 contains C programming examples using the HP-IB communication's interface.

Chapter 14 contains the front-panel menu to subsystem command mapping. This chapter is useful when you known which front-panel menu field you want to change but cannot remember the command required to change it.

Contents

1 Programming the HP 16542A

Programming the HP 16542A Analyzer 1–2
Mainframe Commands 1–4
CARDcage 1–4
MENU 1–5
SELect 1–6
STARt 1–6
STOP 1–7
RMODe 1–7
SYSTem:ERRor 1–7
SYSTem:PRINt 1–8
MMEMory Subsystem 1–8
INTermodule Subsystem 1–8
Command Set Organization 1–9
Module Status Reporting 1–12
MESE<N> 1–13
MESR<N> 1–14

2 CONFig Subsystem

CONFig 2–3
CMEMory 2–3
NAME 2–4
SYSWidth 2–5
TYPE 2–7

3 FORMat Subsystem

FORMat 3–4
CLOCk 3–5
CTHReshold 3–6
LABel 3–7
MASTer 3–9
REMove 3–10
SETHold 3–10

SLAVe 3–11
THReshold 3–13

4 TRIGger Subsystem

TRIGger 4–5
FIND 4–5
MLENgth 4–6
NUMRec 4–13
POSTstore 4–14
RECord 4–15
RLENgth 4–16
STORe 4–17
TERM 4–18

5 LIST Subsystem

LIST 5–5
COLumn 5–6
DATA 5–7
LINE 5–8
MMODe 5–9
OPATtern 5–10
OSEarch 5–11
OSTate 5–12
RECord 5–12
REMove 5–13
SEARch 5–13
XPATtern 5–14
XSEarch 5–15
XSTate 5–16

6 WLISt Subsystem

WLISt 6–8
COLumn 6–8
DATA 6–9
LINE 6–10
MMODe 6–10
OCONdition 6–11

OPATtern 6–12
OSEarch 6–13
OSTate 6–14
OTIMe 6–15
REMove 6–15
RUNTil 6–16
SEARch 6–17
TAVerage 6–18
TMAXimum 6–18
TMINimum 6–19
VRUNs 6–19
XCONdition 6–20
XOTime 6–20
XPATtern 6–21
XSEarch 6–22
XSTate 6–23
XTIMe 6–24

7 SWAVeform Subsystem

SWAVeform 7–5
ACCumulate 7–6
CENTer 7–6
DELay 7–7
INSert 7–8
MMODe 7–8
OPATtern 7–9
OSEarch 7–10
OSTate 7–11
RANGe 7–12
REMove 7–12
RUNTil 7–13
SEARch 7–14
XPATtern 7–14
XSEarch 7–15
XSTate 7–16

8 TWAVeform Subsystem

TWAVeform 8–7
ACCumulate 8–7

CENTer 8–8
DELay 8–8
INSert 8–9
MINus 8–10
MMODe 8–10
OCONdition 8–11
OPATtern 8–12
OSEarch 8–13
OTIMe 8–14
OVERlay 8–15
PLUS 8–16
RANGe 8–16
REMove 8–17
RUNTil 8–18
SEARch 8–19
SPERiod 8–19
TAVerage 8–20
TMAXimum 8–20
TMINimum 8–20
VRUNs 8–21
XCONdition 8–21
XOTime 8–22
XPATtern 8–22
XSEarch 8–24
XTIMe 8–25

9 CHARt Subsystem

CHARt 9–4
ACCumulate 9–5
CENTer 9–5
HAXis 9–6
VAXis 9–8

10 COMPare Subsystem

COMPare 10–5
CLEar 10–6
CMASk 10–6
COPY 10–7
DATA 10–7

FIND 10-9
LINE 10-10
MENU 10-11
RANGe 10-11
RECord 10-12
RUNTil 10-13
SET 10-14

11 SYMBol Subsystem

SYMBol 11-4
BASE 11-5
PATTern 11-5
RANGe 11-6
REMove 11-7
WIDTh 11-7

12 DATA and SETup Commands

SYSTem:DATA 12-4
Data <section_header> Description 12-6
Data <section_data> Description 12-6
<acquisition_data> Description 12-10
SYSTem:SETup 12-14
Setup <section_header> Description 12-15

13 Examples

C Programming Examples 13-2
Example Programs' Header Files 13-3
Imaging Example 13-7
Telecommunications Example 13-9
ASIC Testing Example 13-12
Lotus Spreadsheet Example 13-15
HP16542A Functions 13-17
HP-IB Interface Functions 13-45
HP-UX HP-IB Interface Functions 13-46
Hewlett-Packard HP-IB Functions for the PC 13-54
National Instrument's GPIB Functions for the PC 13-60

14 Menus and Commands

Configuration 14-2
Chart 14-3
Chart 14-4
Chart 14-5
Compare 14-6
Compare 14-7
Compare 14-8
Format 14-9
State Listing 14-10
State Listing 14-11
State Listing 14-12
State Listing 14-13
Timing Listing 14-14
Timing Listing 14-15
Timing Listing 14-16
Timing Listing 14-17
State Waveform 14-18
State Waveform 14-19
State Waveform 14-20
State Waveform 14-21
Symbol 14-22
Symbol 14-23
Timing Waveform 14-24
Timing Waveform 14-25
Timing Waveform 14-26
Timing Waveform 14-27
Timing Waveform 14-28
Timing Waveform 14-29
Trigger 14-30
Trigger 14-31

Index

1

Programming the HP 16542A

Introduction

This manual, combined with the *HP 16500A Programmer's Guide* manual, provides you with the information needed to program the HP 16542A 2 MByte, 100 MHz State and Timing Analyzer module. Each module has its own manual to supplement the mainframe manual because not all mainframes are configured with the same modules.

Programming the HP 16542A Analyzer

This chapter introduces you to the basic command structure used to program the state analyzer. Also included is an example program that sets up the state analyzer for a measurement. The example program is explained line by line to help illustrate the basic program elements and to give you a comfortable feeling of the programming basics.

Selecting the Module

Before you can program the HP 16542A analyzer, you must first "select" it. This directs your commands to the HP 16542A analyzer.

To select the HP 16542A analyzer module, use the system command :SELect followed by the numeric reference for the slot location of the HP 16542A analyzer (1...10 refers to slot A...J respectively). For example, if the HP 16542A master card is in slot C, then the command to select slot 3 would be as follows:

```
:SELect 3
```

For more information on the select command, refer to the *HP 16500A Programming* manual.

Programming the State Analyzer

A typical HP 16542A analyzer program performs the following:

- selects the appropriate module.
- specifies the analyzer type.
- assigns pods.
- assigns labels.
- sets pod thresholds.
- specifies a trigger condition.
- sets up the display.
- specifies acquisition type.
- starts the acquisition.

The following example program sets up the logic analyzer to make a simple state measurement.

Example

```
10  OUTPUT XXX;":SELECT 3"
20  OUTPUT XXX;":CONFIG:NAME 'STATETEST'"
30  OUTPUT XXX;":CONFIG:TYPE STATE"
40  OUTPUT XXX;":FORMAT:LABEL 'DATA',POS, #B11111111"
50  OUTPUT XXX;":FORMAT:THRESHOLD1 ECL"
60  OUTPUT XXX;":MENU 3,5"
70  OUTPUT XXX;":RMODE SINGLE"
80  OUTPUT XXX;":START"
90  END
```

The three X's (XXX) after the "OUTPUT" statements in the previous example refer to the device address required for programming over either HP-IB or RS-232C. Refer to your controller manual and programming language reference manual for information on initializing the interface.

You may have noticed that not all fields in the analyzer menus were set with the sample program. If menu field settings from power-up or a previous program setting are correct, you do not have to program those fields.

Program Comments

Line 10 selects the logic analyzer in slot C.

Line 20 sets analyzer's name to "STATETEST".

Line 30 specifies an External clock (state) measurement.

Line 40 assigns a label called DATA, with positive polarity, and the eight least significant channels of the first pod to the label.

Line 50 sets the threshold of the first pod to ECL.

Line 60 changes the on-screen display to the Listing menu.

Line 70 specifies the Single run mode.

Line 80 starts data acquisition.

For more information on the specific HP 16542A analyzer commands, refer to chapters 2 through 11 of this manual.

Mainframe Commands

These commands are part of the HP 16500A mainframe system and are mentioned here only for reference. For complete information on these commands, refer to the *HP 16500A Programmer's Guide* manual.

CARDcage

Query

```
CARDcage?
```

The CARDcage query returns a string of integers which identifies the modules that are installed in the mainframe. The returned string is in two parts. The first five two-digit numbers identify the card type. The identification number for the HP 16542A master is 42. A "-1" in the first part of the string indicates no card is installed in the slot.

The five single-digit numbers in the second part of the string indicate which slots have cards installed, which card has the controlling software for the module, and where the master card is located.

Example

```
12,11,-1,43,42,2,2,0,5,5
```

A returned string of 12,11,-1,43,42,2,2,0,5,5 means that an oscilloscope (16530A) time base card (ID number 11) is loaded in slot B and the oscilloscope (16531A) acquisition card (ID number 12) is loaded in slot A. The next slot (C) is empty (-1). Slot D contains the HP 16542A analyzer module's expander board (ID number 43). Slot E contains the HP 16542A analyzer module's master board (ID number 42).

The next group of numbers (2,2,0,5,5) indicate that a 2 card module is installed in slots A and B with the master card in slot B. The "0" indicates an empty slot or the module software is not recognized or not loaded. The last two numbers show a two card module in slots D and E with the master in slot E.

MENU

Command

```
MENU <X>[,<menu>]
```

The MENU command selects a new displayed menu. The first parameter (X) specifies the desired module. The optional second parameter specifies the desired menu in the module (defaults to 0 if not specified).

Query

```
MENU?
```

The query returns the currently selected (and displayed) menu.

Menu identification for the HP 16542A analyzer is listed below:

X,0 - Configuration menu
X,1 - Format menu
X,2 - Trigger menu
X,3 - Timing Waveform menu
X,4 - State Waveform menu
X,5 - Listing menu
X,6 - Unused
X,7 - Compare menu
X,8 - Chart menu
X,9 - Unused
X,10 - Calibration

An error is returned when selecting menu X,7 if the compare memory is disabled.

An error is returned when selecting menu X,3 (Timing Waveform) and the analyzer is in state mode or when selecting menu X,4 (State Waveform) and the analyzer is in timing mode.

An error is returned when selecting menu X,6 or X,9.

The menu X,5 (Listing) is used for either state of timing.

SELect

Command

`SELect <module_number>`

The SELect command selects which module or intermodule will have parser control. SELect 0 selects the intermodule, SELect 1 through 10 selects modules A through J, respectively. Values -1 and -2 select software options 1 and 2.

Query

SELect?

The SELect query returns the currently selected module.

STARt

Command

`STARt`

The STARt command starts the specified module or intermodule. If the specified module is configured for intermodule, the STARt command starts all modules configured for intermodule.

STOP

Command

```
STOP
```

The STOP command stops the specified module or intermodule. If the specified module is configured for intermodule, the STOP command stops all modules configured for intermodule.

> STARt and STOP are Overlapped Commands. Overlapped Commands allow execution of subsequent commands while the logic analyzer operations initiated by the Overlapped Command are still in progress. For more information see *OPC and *WAI commands in Chapter 5 of the HP 16500A Programmer's Guide manual.

RMODe

Command

```
RMODe <mode>
```

The RMODe command specifies the run mode (SINGle or REPetitive) for a module or intermodule. If the selected module is configured for intermodule, the intermodule run mode is set by this command.

Query

```
RMODe?
```

The RMODe query returns the current run mode setting.

SYSTem:ERRor

Query

```
SYSTem:ERRor?
```

The SYSTem:ERRor query returns the oldest error in the error queue. In order to return all the errors in the error queue, a simple FOR/NEXT loop can be written to query the queue until all errors are returned. Once all errors are returned, the query returns zeros.

SYSTem:PRINt

Command

```
SYSTem:PRINt {SCReen|ALL}
```

The SYSTem:PRINt command initiates a print of the screen or listing buffer over the current printer communication interface.

Query

```
SYSTem:PRINt? {SCReen|ALL}
```

The SYSTem:PRINt query sends the screen or listing buffer data over the current controller communication interface.

MMEMory Subsystem

The MMEMory Subsystem provides access to both internal disk drives for loading and storing configurations.

INTermodule Subsystem

The INTermodule Subsystem commands are used to specify intermodule arming between multiple modules.

Command Set Organization

The command set for the HP 16542A is divided into subsystem commands. Each subsystem contains the commands used in its corresponding menus as listed below.

- Configuration menu (CONFig subsystem).
- Format menu (FORMat subsystem).
- Trigger menu (TRIGger subsystem).
- State Listing menu (LIST subsystem).
- Timing Listing menu (WLISt subsystem).
- State Waveform menu (SWAVeform subsystem).
- Timing Waveform menu (TWAVeform subsystem).
- Chart menu (CHARt subsystem).
- Compare menu (COMPare subsystem).
- Symbols field in Format menu (SYMBol subsystem).

The command tree in table 1-1 shows all subsystems for the HP 16542A State Analyzer module. The (x) by the SELect command at the top of the tree represents the slot number where the HP 16542A analyzer module is installed. The number may range from 1 through 10, representing slots A through J, respectively.

You will find in the remaining chapters, a full description of the individual subsystems, syntax diagrams and the commands in alphabetical order. The commands are shown in longform and shortform using upper and lower-case letters. For example, LABel indicates that the longform of the command is LABel and the shortform is LAB. Each of the commands contain a description of the command and its arguments, the command syntax, and a programming example.

Table 1-1 **HP 16542A Command Tree**

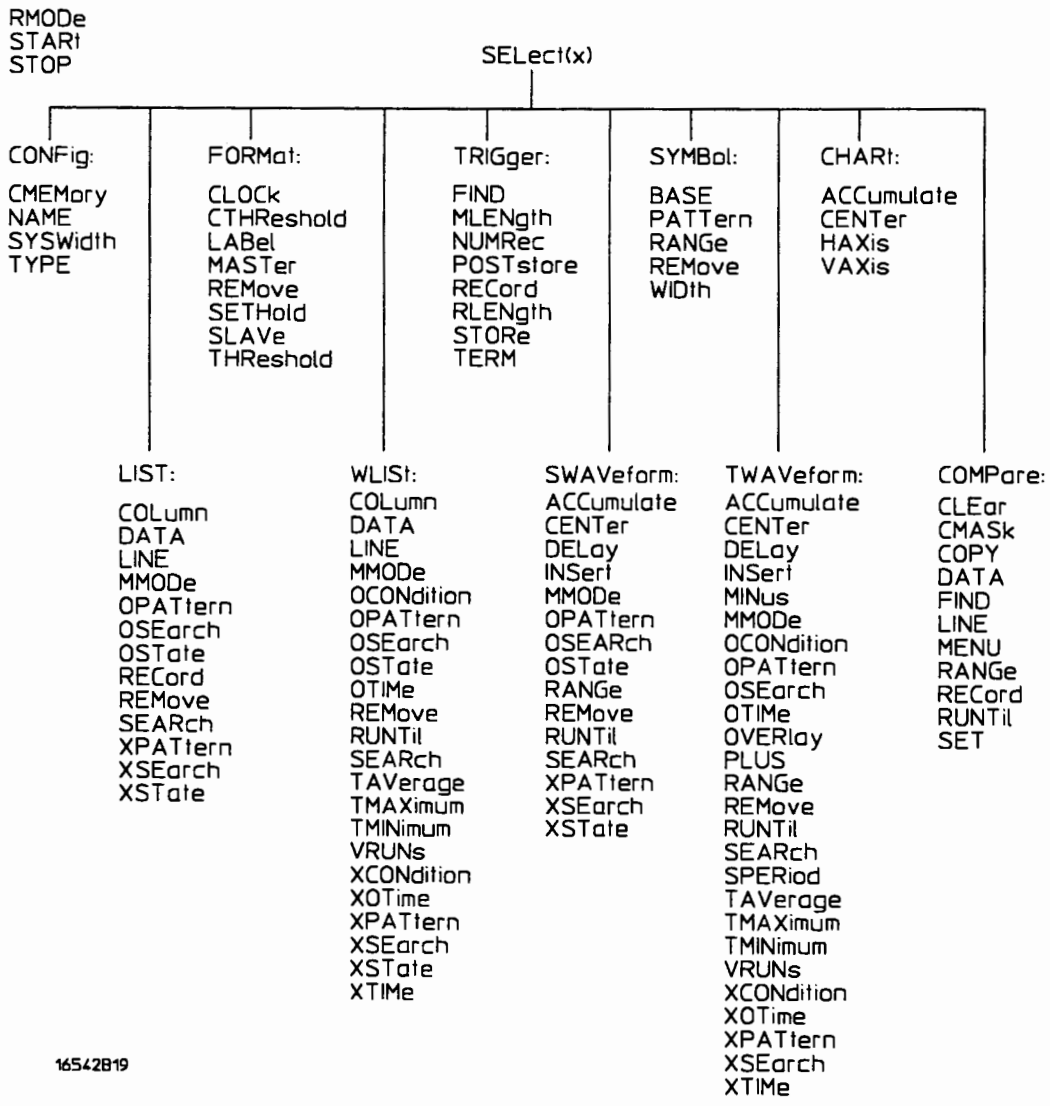


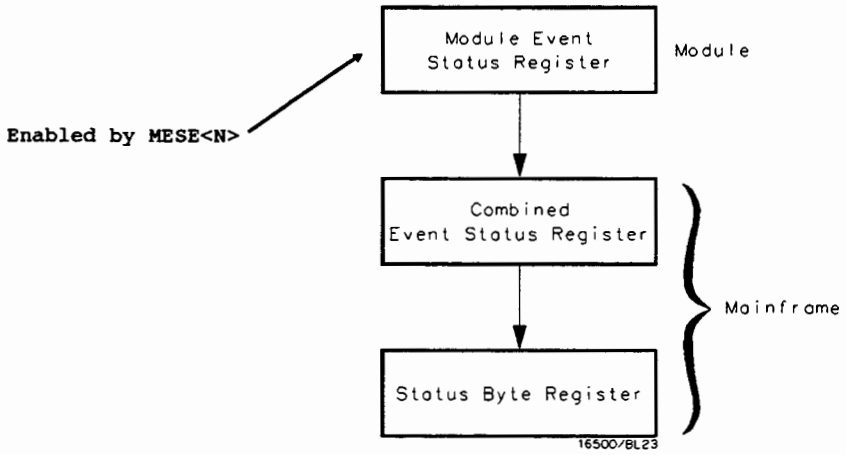
Table 1-2 Alphabetical Command-to-Subsystem Directory

Command	Where used
ACCumulate	CHARt, SWAVeform, TWAVeform
BASE	SYMBol
CENTer	CHARt, SWAVeform, TWAVeform
CLEar	COMPare
CLOCk	FORMat
CMASk	COMPare
CMEMory	CONFig
COLumn	LIST, WLISt
COPY	COMPare
CTHReshold	FORMat
DATA	COMPare, LIST, WLISt
DELay	SWAVeform, TWAVeform
FIND	COMPare, TRIGger
HAXis	CHARt
INSert	SWAVeform, TWAVeform
LABel	FORMat
LINE	COMPare, LIST, WLISt
MASTer	FORMat
MENU	COMPare
MINus	TWAVeform
MLENgth	TRIGger
MMODe	LIST, SWAVeform, TWAVeform, WLISt
NAME	CONFig
NUMRec	TRIGger
OCONdition	TWAVeform, WLISt
OPATtern	LIST, SWAVeform, TWAVeform, WLISt
OSEarch	LIST, SWAVeform, TWAVeform, WLISt
OSTate	LIST, SWAVeform, WLISt
OTIMe	TWAVeform, WLISt
OVERlay	TWAVeform
PATTern	SYMBol
PLUS	TWAVeform
POSTstore	TRIGger
RANGe	COMPare, SWAVeform, SYMBol, TWAVeform
RECord	COMPare, LIST, TRIGger
REMove	FORMat, LIST, SWAVeform, SYMBol, TWAVeform, WLISt
RLENgth	TRIGger
RUNTil	COMPare, SWAVeform, TWAVeform, WLISt
SEARch	LIST, SWAVeform, TWAVeform, WLISt
SET	COMPare
SETHold	FORMat
SLAVe	FORMat
SPERiod	TWAVeform
STORe	TRIGger
SYSWidth	CONFig
TAVerage	TWAVeform, WLISt
TERM	TRIGger
THReshold	FORMat
TMAXimum	TWAVeform, WLISt
TMINimum	TWAVeform, WLISt
TYPE	CONFig
VAXis	CHARt
VRUNs	TWAVeform, WLISt
WIDTh	SYMBol
XCONdition	TWAVeform, WLISt
XOTime	TWAVeform, WLISt
XPATtern	LIST, SWAVeform, TWAVeform, WLISt
XSEarch	LIST, SWAVeform, TWAVeform, WLISt
XSTate	LIST, SWAVeform, WLISt
XTIMe	TWAVeform, WLISt

Module Status Reporting

Each module reports its status to the Module Event Status Register (MESR<N>), which in turn reports to the Combined Event Status Register (CESR) in the HP 16500A mainframe. The Module Event Status Register is enabled by the Module Event Status Enable Register (MESE<N>).

Figure 1-1



The MESE<N> and MESR<N> instructions are not used in conjunction with the SELect command, so they are not listed in the command tree of the HP 16542A.

On the following pages are descriptions of the MESE<N> and MESR<N> instructions. They provide the module specific information needed to enable and interpret the contents of the registers.

MESE<N>

Command

```
:MESE<N> <enable_mask>
```

The MESE<N> command sets the Module Event Status Enable register bits. The MESE register contains a mask value for the bits enabled in the MESR register. A one in the MESE will enable the corresponding bit in the MESR, and a zero will disable the bit.

The first parameter <N> specifies the module (1...10 refers to the module in slot A...J). The second parameter specifies the enable value.

Refer to table 1-3 for information about the Module Event Status Enable register bits, bit weights, and what each bit masks for the module. Complete information for status reporting is in chapter, "Mainframe Commands" in the *HP 16500A Programming Reference* manual.

`<N>` `{1|2|3|4|5|6|7|8|9|10}` number of slot in which the module resides

`<enable_mask>` An integer from 0 to 255

Example

```
OUTPUT XXX;":MESE5 1"
```

Query

```
:MESE<N>?
```

The MESE query returns the current setting.

Returned Format

```
[:MESE<N>]<enable_mask><NL>
```

`<N>` `{1|2|3|4|5}` number of slot in which the module resides

`<enable_mask>` An integer from 0 to 255

Example

```
OUTPUT XXX;":MESE5?"
```

Table 1-3 **Module Event Status Enable Register**

Bit	Weight	A "1" Enables
7	128	Not Used
6	64	Not Used
5	32	Not Used
4	16	Not Used
3	8	Not Used
2	4	Trigger found
1	2	RNT-Run until satisfied
0	1	MC-Measurement complete

The Module Event Status Enable Register contains a mask value for the bits to be enabled in the Module Event Status Register (MESR). A one in the MESE enables the corresponding bit in the MESR, a zero disables the bit.

MESR<N>

Query

`:MESR<N>?`

The MESR<N> query returns the contents of the Module Event Status register.

> Reading the register clears the Module Event Status Register.

Table 1-4 shows each bit in the Module Event Status Register and their bit weights for this module. When you read the MESR, the value returned is the total bit weights of all bits that are set at the time the register is read.

The parameter 1...10 refers to the module in slot A...J, respectively.

Returned Format

`[MESR<N>] <status><NL>`

`<N>` `{1|2|3|4|5|6|7|8|9|10}` number of slot in which the module resides

`<status>` An integer from 0 to 255

Example

```
OUTPUT XXX;":MESR5?"
```

Table 1-4 **Module Event Status Register**

Bit	Weight	Condition
7	128	Not Used
6	64	Not Used
5	32	Not Used
4	16	Not Used
3	8	Not Used
2	4	1 = Real trigger found 0 = Real trigger not found
1	2	1 = Run until satisfied 0 = Run until not satisfied
0	1	1 = Measurement complete 0 = Measurement not complete

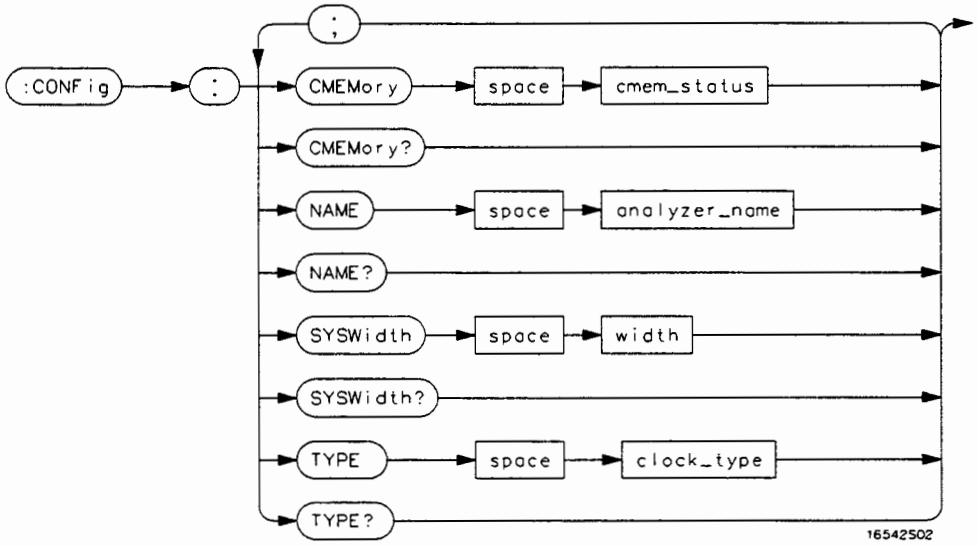
2

CONFig Subsystem

The CONFig subsystem contains the commands available for the Configuration menu. These commands are listed below:

- CMEMory
- NAME
- SYSWidth
- TYPE

Figure 2-1



CONFig Subsystem Syntax Diagram

Table 2-1 **CONFig Parameter Values**

Parameter	Value
analyzer_name	string of up to 10 alphanumeric characters
clock_type	Any one of these keywords: `TIMing or STATe`
cmem_status	Any one of these keywords: ENABled or DISabled
width	Any one of these integers: 8, 16, 24, 32, 40, 48, 64, 80

Table 2-2 **CONFig Power On Default Values**

Command	Default
NAME	MACHINE 1
TYPE	STATe
CMEMory	DISabled
SYSWidth	16 times the number of cards in the analyzer

CONFig

Selector

```
:CONFig
```

The CONFig selector is used as a part of a compound header to access the settings in the Configuration menu. The CONFig selector must precede any Configuration subsystem commands.

Example

```
OUTPUT XXX;":CONFig:NAME 'DRAMTEST'"
```

CMEMory

Command

```
:CONFig:CMEMory <cmem_status>
```

The CMEMory command specifies whether the compare memory is enabled or disabled.

`<cmem_status>` `{ENABled|DISabled}`

Example `OUTPUT XXX;":CONFig:CMEMory ENABled`

Query `:CONFig:CMEMory?`

The CMEMory query returns the current compare memory status for the analyzer.

Returned Format `[:CONFig:CMEMory] <cmem_status><NL>`

`<cmem_status>` `{ENABled|DISabled}`

Example `OUTPUT XXX;":CONFig:CMEMory?"`

NAME

Command `:CONFig:NAME <analyzer_name>`

The NAME command allows you to assign a name of up to 10 characters to the analyzer for easier identification.

`<analyzer_name>` string of up to 10 alphanumeric characters

Example `OUTPUT XXX;":CONFig:NAME 'DRAMTEST'"`

Query `:CONFig:NAME?`

The NAME query returns the current analyzer name as an ASCII string.

Returned Format `[CONFig:NAME] <analyzer name><NL>`

`<analyzer_ name>`	string of up to 10 alphanumeric characters

Example

```
OUTPUT XXX;":CONFig:NAME?"
```

SYSWidth

Command

```
:CONFig:SYSWidth <width>
```

The SYSWidth command specifies the memory width and depth for a set of one or more HP16542A cards connected together as a logic analyzer module. For any combination master and expansion cards, there is a limited number of system width and depth configurations. Selecting the system width also determines the system memory depth. Memory depth also depends on the setting of the compare memory. The following table shows the valid combinations of memory width and depth.

Table 2-3 **Memory Width and Depth**

Number of Cards	<width> Parameter	Memory Depth Disabled[1]	Enabled[2]
1	16	1M	256K
	8	2M	512K
2	32	1M	256K
	16	2M	512K
	8	4M	1M
3	48	1M	256K
	24	2M	512K
	16	3M	1M
	8	6M	2M
4	64	1M	256K
	32	2M	512K
	16	4M	1M
	8	8M	2M
5	80	1M	256K
	40	2M	512K
	16	5M	1M
	8	10M	2M

1 Compare memory disabled
2 Compare memory enabled

Example

```
OUTPUT XXX;":CONFig:SYSWidth 16"
```

Query `:CONFig:SYSWidth?`

The SYSWidth query returns the current memory width and depth for the analyzer.

Returned Format `[CONFig:SYSWidth] <width>,<depth><NL>`

`<width>` `{8|16|24|32|40|48|64|80}`

`<depth>` `{256K|512K|1M|2M|3M|4M|5M|6M|8M|10M}`

Example

```
OUTPUT XXX;":CONFig:SYSWidth?"
```

TYPE

Command `:CONFig:TYPE <clock type>`

The TYPE command specifies one of two clock types: external (STATe) or internal (TIMing). In addition to selecting the clock type, the TYPE command changes the options available for other subsystem commands.

`<clock type>` `{TIMing|STATe}`

Example

```
OUTPUT XXX;":CONFig:TYPE TIMING"
```

Query `:CONFig:TYPE?`

The TYPE query returns the current clock type for the analyzer.

Returned Format `[:CONFig:TYPE] <clock_type><NL>`

`<clock_type>` `{TIMing|STATe}`

Example

```
OUTPUT XXX;":CONFig:TYPE?"
```

3

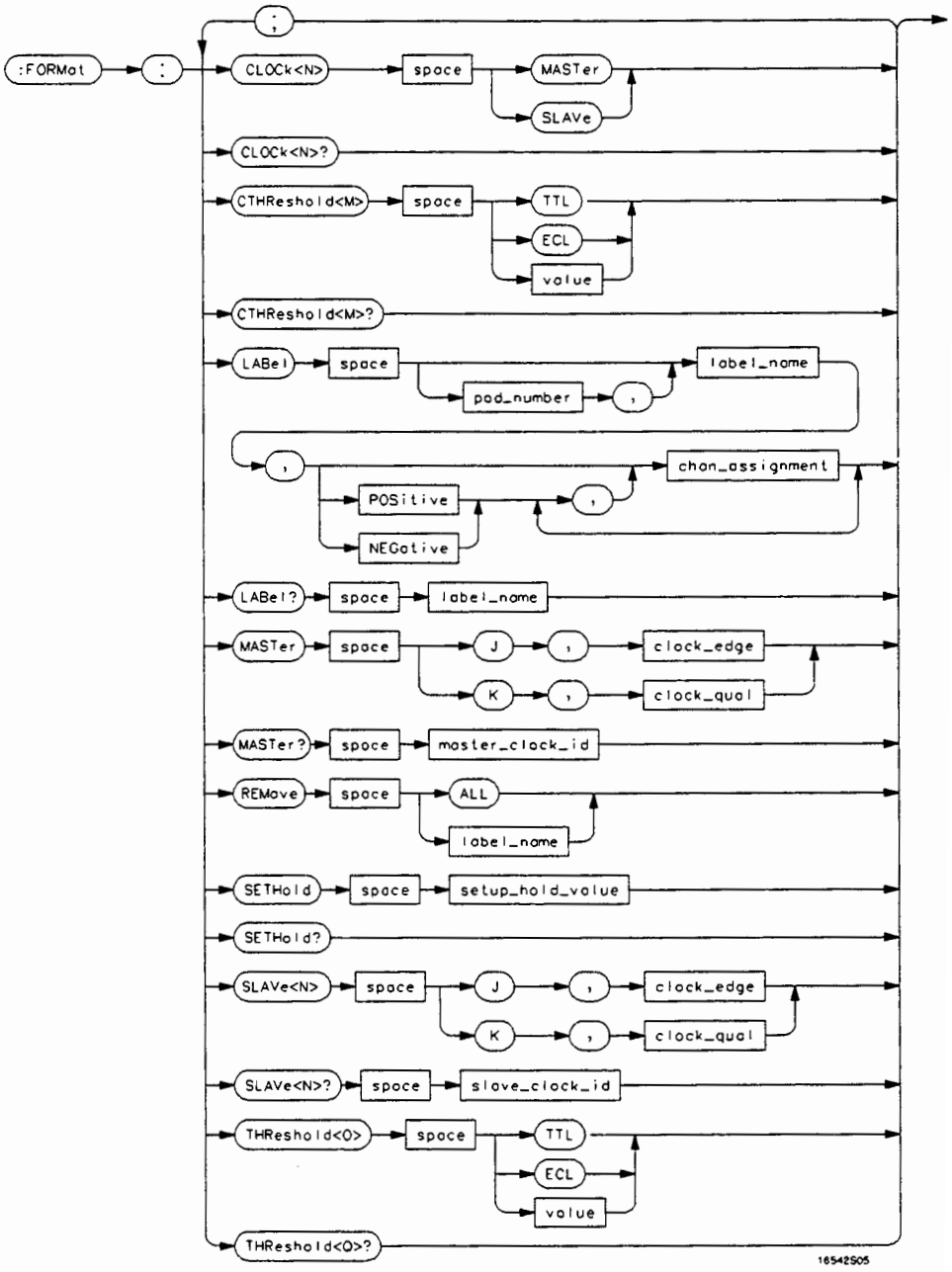
FORMat Subsystem

Introduction

The FORMat subsystem contains the commands available for the Format menu in the HP 16542A logic analyzer module. These commands are listed below:

- CLOCk
- CTHReshold
- LABel
- MASTer
- REMove
- SETHold
- SLAVe
- THReshold

Figure 3-1



FORMat Subsystem Syntax Diagram

Table 3-1 **FORMat Parameter Values**

Parameter	Value
<N>	integer from 1 to 4
<M>	integer from 0 to 4
<Q>	integer from 0 to 4
setup hold value	integer from 0 to 2
label_name	string of up to 6 alphanumeric characters
polarity	{POSitive | NEGative}
chan_assignment	format (integer from 0 to 65535)
master_clock_id	{J|K}
slave_clock_id	{J|K}
clock_qual	{OFF|RISing|FALLing|BOTH|LOW|HIGH}
clock_edge	{OFF|RISing|FALLing|BOTH}
value	voltage (real number) -3.5 V to +5.0 V
pod number	integer from 0 to 4

Table 3-2 **Power On Defaults**

Command	Default
CLOCk	MASTer
LABel	'Lab1',POStive,#B1111111111111111
MASTer	J, RISing
THReshold	TTL
CTHReshold	TTL
SETHold	0

FORMat

Selector

```
:FORMat
```

The FORMat selector is used as a part of a compound header to access the settings in the Format menu. The FORMat selector command always precedes any Format subsystem commands.

Example

```
OUTPUT XXX;":FORMat:MASTER J,RISING"
```

CLOCk

Command

```
:FORMat:CLOCk<N> <clock_mode>
```

The CLOCk command selects the clocking mode (master or slave clocking) for a given expansion card. When the MASTer option is specified, the pod on the expansion card samples data on the master clock. When the SLAVe option is specified the pod on the expansion card samples data on the slave clock. Expansion card numbering is relative to the position in the card cage from top to bottom (1 = top most expansion card).

> The CLOCk command is valid only when the analyzer is configured as a state analyzer. See the TYPE command in Chapter 2 for more information.

`<N>` `{1|2|3|4}` the expansion card number

`<clock_mode>` `{MASTer|SLAVe}`

Example

```
OUTPUT XXX;":FORMat:CLOCk2 SLAVE"
```

Query

```
:FORMat:CLOCk<N>?
```

The CLOCk query returns the current clocking mode for a given expansion card.

Returned Format

```
[:FORMat:CLOCk<N>] <clock_mode><NL>
```

`<N>` `{1|2|3|4}` the expansion card number

`<clock_mode>` `{MASTer|SLAVe}`

Example

```
OUTPUT XXX; ":FORMat:CLOCk?"
```

CTHReshold

Command

```
:FORMat:CTHReshold<M> {TTL|ECL|<value>}
```

The CTHReshold command allows you to set the voltage threshold for the master clock or each selected slave clock. The threshold levels available are ECL, TTL, and a specific voltage from -3.5V to +5.0V in 0.1 volt increments. A card number of "0" selects the master, and "1 through 4" selects expansion cards. The card number assigned to the expander card is relative to the top position in the module (1 = top slot used by the module). The master card can be positioned between expansion cards without affecting expansion card number order.

> The CTHReshold command is valid only when the analyzer is configured as a state analyzer. See the TYPE command in Chapter 2 for more information.

<M> card number {0|1|2|3|4} where 0 = master and 1-4 = expansion

<value> voltage (real number) -3.5 V to +5.0 V

TTL a voltage threshold value of +1.6V

ECL a voltage threshold value of -1.3V

Example

```
OUTPUT XXX;":FORMat:CTHReshold1 4.0"
```

Query

```
:FORMat:CTHReshold<M>?
```

The CTHReshold query returns the current threshold for a given clock.

Returned Format `[:FORMat:CTHReshold<M>] <value><NL>`

`<M>` card number {0|1|2|3|4} where 0 = master and 1-4 = expansion

`<value>` voltage (real number) -3.5 V to +5.0 V or TTL or ECL

Example

```
OUTPUT XXX;":FORMat:CTHReshold1?"
```

LABel

Command

```
:FORMat:LABel [<pod_number>,] <'name'>
[,<polarity>],<assignment>,[,<assignment>]
```

The LABel command allows you to specify polarity and to assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label is created. If more than 126 labels are specified, the last label (bottom label) in the Format menu is modified.

The first parameter is optional and is used to specify the first pod that is to have channels assigned. If the first parameter is not specified at the time the pod is specified, the left-most pod on the Format screen is assumed. The pods are numbered in the same order as they appear in the Format menu, with zero representing the leftmost pod and the topmost card in the analyzer card slots.

The polarity can be specified at any point after the label name.

Because pods contain 16 channels, the channel assignment value for a pod must be between 0 and 65535 (2^{16}-1). When giving the pod assignment in binary (base 2), each bit will correspond to a single channel. A "1" in a bit position means the associated channel in that pod is assigned to that pod and bit. A "0" in a bit position means the associated channel in that pod is excluded from the label. For example, assigning #B1111001100 is equivalent to entering "......****..**.." through the touch screen.

The maximum number of channels per label is 32. When the system width is set to eight in the configuration menu, only the 8 least significant channels on the pod are used. If a channel assignment value is greater than 255, only the 8 least significant channels are used and a warning is returned.

`<pod number>` An integer from 0 - 4

`<'name'>` A string of up to 6 case sensitive alphanumeric characters

`<polarity>` `{POSitive|NEGative}`

`<assignment>` a number representing the channel assignment for the specified pod

Examples

```
OUTPUT XXX;":FORMat:LABel 1,'STAT', POSitive, 65535,127,40312"
OUTPUT XXX;":FORMat:LABel 'SIG 1', 64, 12, 0, 20"
OUTPUT XXX;":FORMat:LABel 4,'ADDR', NEG, #B0011110010101010"
```

Query

```
:FORMat:LABel? <label_name>
```

The LABel query returns the current specification for the selected (by name) label. If the label does not exist, nothing is returned. The polarity is always returned as the second parameter. Numbers are always returned in decimal format.

Returned Format

```
[:FORMat:LABel] <label_name>,<polarity>[,<assignment>]...<NL>
```

`label_name` A string of up to 6 case sensitive alphanumeric characters

`<polarity>` `{POSitive|NEGative}`

`<assignment>` a number representing the channel assignment for the specified label

Example

```
OUTPUT XXX;":FORMat:LABel? 'DATA'"
```

MASTer

Command

```
:FORMat:MASTer <clock_id>,<clock_spec>
```

The MASTer clock command allows you to specify a master clock for the entire analyzer module. Each command deals with only one clock (J or K), therefore, a complete master clock specification for the analyzer requires two commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed while level specifications (LOW or HIGH) are ANDed. Only the K clock may contain a level specification.

> The MASTer command is valid only when the analyzer is configured as a state analyzer. See the TYPE command in Chapter 2 for more information.

> At least one clock edge must always be specified on the master clock.

`<clock_id>` `{J|K}`

`<clock_spec>` `{OFF|RISing|FALLing|BOTH}` for the J clock

`{OFF|RISing|FALLing|BOTH|LOW|HIGH}` for the K clock

Example

```
OUTPUT XXX;":FORMat:MASTer J, RISING"
```

Query

```
:FORMat:MASTer? <clock_id>
```

The MASTer query returns the clock specification for the specified clock.

Returned Format

```
[:FORMat:MASTer] <clock_id>,<clock_spec><NL>
```

`<clock_id>` `{J|K}`

`<clock_spec>` `{OFF|RISing|FALLing|BOTH}` for the J clock

`{OFF|RISing|FALLing|BOTH|LOW|HIGH}` for the K clock

Example

```
OUTPUT XXX;":FORMat:MASTer? J"
```

REMove

Command

```
:FORMat:REMove {<name>|ALL}
```

The REMove command allows you to delete all labels or to delete any one label.

`<name>` A string of up to 6 alphanumeric characters

Examples

```
OUTPUT XXX;":FORMat:REMove 'A'"
OUTPUT XXX;":FORMat:REMove ALL"
```

SETHold

Command

```
:FORMat:SETHold <L>
```

The SETHold command allows you to select one of three setup and holdtime combinations. An error is returned if the analyzer pod has not been calibrated.

`<L>` `{0|1|2}`

0 = 4/0 ns = 4 ns setup and 0 ns hold

1 = 2/2 ns = 2 ns setup and 2 ns hold

2 = 0/4 ns = 0 ns setup and 4 ns hold

Example

```
OUTPUT XXX;":FORMat:SETHold 2
```

Query `:FORMat:SETHold?`

The SETHold query returns the current module setup and hold setting.

Returned Format `[:FORMat:SETHold] <L><NL>`

<L> `{0|1|2|3}`

0 = 4/0 ns = 4 ns setup and 0 ns hold
1 = 2/2 ns = 2 ns setup and 2 ns hold
2 = 0/4 ns = 0 ns setup and 4 ns hold
3 = System not calibrated

Example

```
OUTPUT XXX;":FORMat:SETHold?"
```

SLAVe

Command `:FORMat:SLAVe<N> <clock_id>,<clock_spec>`

The SLAVe clock command allows you to specify a slave clock for a given expansion card. Each command deals with only one clock (J or K), therefore, a complete clock specification for an expander card requires two commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed while level specifications (LOW or HIGH) are ANDed. The J clock may only contain edge specifications and the K clock can contain either a level specification or an edge specification.

The SLAVe command is valid only when the analyzer is configured as a state analyzer. See the TYPE command in Chapter 2 for more information.

At least one edge must be specified on the expansion clock.

<N> {1|2|3|4}

<clock_id> {J|K}

<clock_spec> {OFF|RISing|FALLing|BOTH} for J clock

{OFF|RISing|FALLing|BOTH|LOW|HIGH} for K clock

Example

```
OUTPUT XXX;":FORMat:SLAVe2 J, RISING"
```

Query

```
:FORMat:SLAVe<N>? <clock_id>
```

The SLAVe query returns the clock specification for the specified clock. Expansion card number returned is relative to the position in the card cage from top to bottom (1 = top most expansion card).

Returned Format

```
[:FORMat:SLAVe<N>] <clock_id>,<clock_spec><NL>
```

<N> {1|2|3|4}

<clock_id> {J|K}

<clock_spec> {OFF|RISing|FALLing|BOTH} for J clock

{OFF|RISing|FALLing|BOTH|LOW|HIGH} for K clock

Example

```
OUTPUT XXX;":FORMat:SLAVe1? J"
```

THReshold

Command

```
:FORMat:THReshold<M> {TTL|ECL|<value>}
```

The THReshold command allows you to set the voltage threshold for a given master or expansion pod to ECL, TTL, or a specific voltage from -3.5V to +5.0V in 0.1 volt increments. Each pod on a given expansion card can have a different threshold level. A card number of "0" selects the master pod, and "1 through 4" selects expansion pods as they appear in the Format menu from left to right.

`<M>` pod number `{0|1|2|3|4}` where 0 = master and 1-4 = expansion

`<value>` voltage (real number) -3.5 V to +5.0 V

`TTL` default value of +1.6V

`ECL` default value of -1.3V

Example

```
OUTPUT XXX;":FORMat:THReshold1 4.0"
```

Query

```
:FORMat:THReshold<M>?
```

The THReshold query returns the current threshold for a given pod.

Returned Format

```
[:FORMat:THReshold<M>] <value><NL>
```

`<M>` pod number `{0|1|2|3|4}` where 0 = master and 1-4 = expansion

`<value>` voltage (real number) -3.5 V to +5.0 V or TTL or ECL

Example

```
OUTPUT XXX;":FORMat:THReshold4?"
```

4

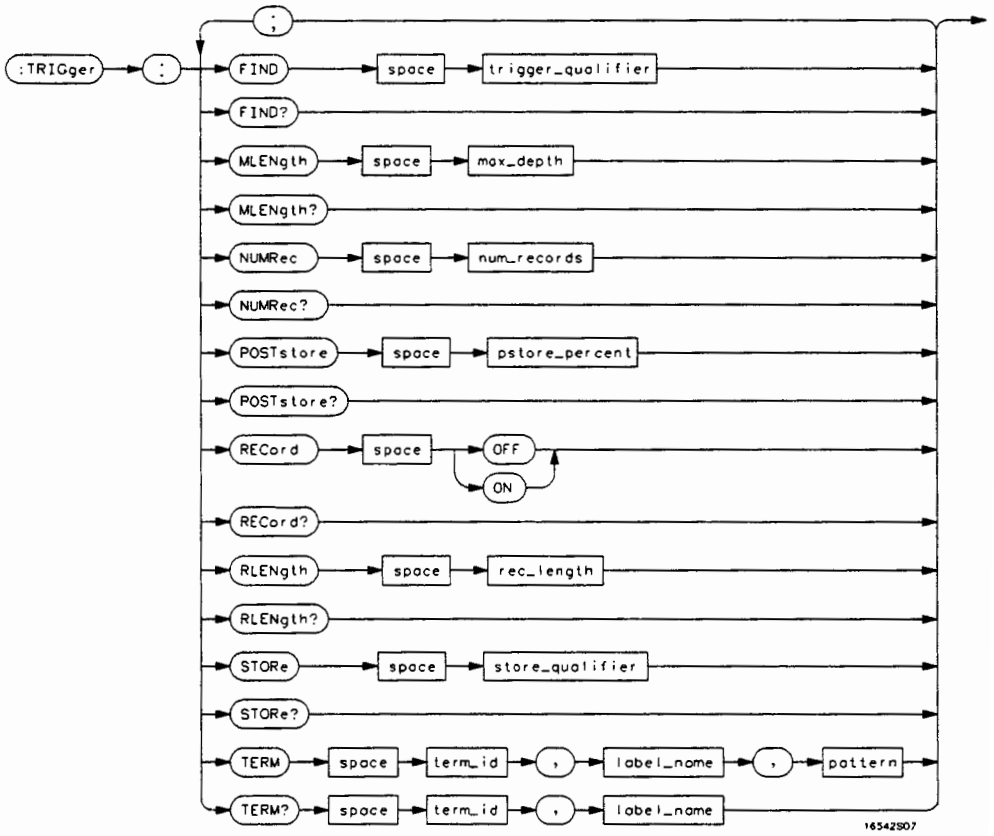
TRIGger Subsystem

Introduction

The TRIGger subsystem contains the commands available for the Trigger menu in the HP 16542A logic analyzer module. The TRIGger subsystem commands are listed below:

- FIND
- MLENgth
- NUMRec
- POSTstore
- RECord
- RLENgth
- STORe
- TERM

Figure 4-1



TRIGger Subsystem Syntax Diagram

Table 4-1 TRIGger Parameter Values

Parameter	Value
trigger_qualifier	Any one of these keywords: `ANYState,NOSTate, A, B, C, D, NOTA, NOTB, NOTC,` or `NOTD`.
label_name	string of up to 6 alphanumeric characters
store_qualifier	Any one of the following: `ANYState,NOSTate,any_term,` or `(expression)`
term_id	Any one of these keywords: `A, B, C,` or `D`.

pattern	Any one of the following:
	`#B{0| 1| X}...`
	`#Q{0| 1|2|3|4|5|6|7| X} . . .`
	`#H{0| 1|2|3|4|5|6|7|8|9|A|B|C|D|E|F| X}...`
	`{0| 1|2|3|4|5|6|7|8|9}...`

any_term	Any one of these keywords: `A, B, C, D, NOTA, NOTB, NOTC,` or `NOTD`
expression	Any one of the following: any_term any_term OR any_term any_term OR any_term OR any_term
mem_length	integer from 4096 through max_depth
rec_length	integer from 1 through max_depth
num_records	integer from 1 through max_recs
pstore_percent	Any one of the following integers: 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, or 100
max_depth	current system configuration depth (with or without compare memory enabled) minus 1024 states (See tables 5-3 through 5-4)
max_recs	max_depth/rec_length rounded down to nearest integer value

Table 4-2 **TRIGger Power On Default Values**

Command	Default
FIND	A
STORe	ANYState
MEMLength	See tables 4-3 through 4-7
TERM	A,'Lab1',#HXXXX B,'Lab1',#HXXXX C,'Lab1',#HXXXX D,'Lab1',#HXXXX
RECords	OFF
POSTstore	50

TRIGger

Selector

```
:TRIGger
```

The TRIGger selector is used as a part of a compound header to access the settings found in the State Trigger menu.

> It is also possible to use the TRACe selector in place of the TRIGger selector

Example

```
OUTPUT XXX;":TRIGger:TERM? A 'DATA'"
```

FIND

Command

```
:TRIGger:FIND <trigger_qualifier>
```

The FIND command defines the trigger qualifier which tells the analyzer when to start storing information. If the analyzer is in the state mode, the information stored depends on the store qualifier term as set by the STORe

command. The trigger qualifier is always stored in memory even if it does not match the store qualifier.

`<trigger_qualifier>` `{ANYState|NOSTate|<any_term>}`

`<any_term>` `{A|B|C|D|NOTA|NOTB|NOTC|NOTD}`

Example

```
OUTPUT XXX;":TRIGger:FIND ANYSTATE"
OUTPUT XXX;":TRIGger:FIND A"
```

Query `:TRIGger:FIND?`

The FIND query returns the current trigger qualifier.

Returned Format `[:TRIGger:FIND] <trigger_qualifier><NL>`

`<trigger_qualifier>` `{ANYState|NOSTate|<any_term>}`

`<any_term>` `{A|B|C|D|NOTA|NOTB|NOTC|NOTD}`

Example

```
OUTPUT XXX;":TRIGger:FIND?"
```

MLENgth

Command `:TRIGger:MLENgth <mem_length>`

The MLENgth command allows you to specify the analyzer memory depth. Valid memory depths range from 4096 states through the total system configuration depth minus 1024 states. Recall that the system configuration depth varies depending on the number of cards in the system, system width, and the condition of the compare memory. Tables 4-3 through 4-7 show the

possible memory depth values for all system configurations. The MLENgth command is only valid when record mode is turned off.

`<mem_length>` An integer from 4096 through max_depth

Example

```
OUTPUT XXX;":TRIGger:MLENgth 4096"
```

Query

:TRIGger:MLENgth?

The MLENgth query returns the current memory depth. The MLENgth query is only valid when record mode is turned off.

Returned Format

`[:TRIGger:MLENgth] <mem_length><NL>`

`<mem_length>` An integer from 4096 through max_depth

Example

```
OUTPUT XXX;":TRIGger:MLENgth?"
```

Table 4-3

MLENgth Maximum Memory Depth 1 Card System

16 Bit Width Disabled[1]	Enabled[2]	8 Bit Width Disabled[1]	Enabled[2]
1047552	261120	2096128	523264
524288	131072	1048576	262144
262144	65536	524288	131072
131072	32768	262144	65536
65536	16384	131072	32768
32768	8192	65536	16384
16384	4096	32768	8192
8192		16384	4096
4096		8192	
		4096	

[1] Compare Memory Disabled
[2] Compare Memory Enabled

Table 4-4

MLENgth Maximum Memory Depth 2 Card System

32 Bit Width Disabled[1]	Enabled[2]	16 Bit Width Disabled[1]	Enabled[2]	8 Bit Width Disabled[1]	Enabled[2]
1047552	261120	2096128	523264	4193280	1047552
524288	131072	1048576	262144	3145728	524288
262144	65536	524288	131072	2097152	262144
131072	32768	262144	65536	1048576	131072
65536	16384	131072	32768	524288	65536
32768	8192	65536	16384	262144	32768
16384	4096	32768	8192	131072	16384
8192		16384	4096	65536	8192
4096		8192		32768	4096
		4096		16384	
				8192	
				4096	

[1] Compare Memory Disabled
[2] Compare Memory Enabled

Table 4-5

MLENgth Maximum Memory Depth 3 Card System

48 Bit Width Disabled[1]	Enabled[2]	24 Bit Width Disabled[1]	Enabled[2]	16 Bit Width Disabled[1]	Enabled[2]	8 Bit Width Disabled[1]	Enabled[2]
1047552	261120	2096128	523264	3144704	1047552	6290432	2096128
524288	131072	1048576	262144	2097152	524288	5242880	1048576
262144	65536	524288	131072	1048576	262144	4194304	524288
131072	32768	262144	65536	524288	131072	3145728	262144
65536	16384	131072	32768	262144	65536	2097152	131072
32768	8192	65536	16384	131072	32768	1048576	65536
16384	4096	32768	8192	65536	16384	524288	32768
8192		16384	4096	32768	8192	262144	16384
4096		8192		16384	4096	131072	8192
		4096		8192		65536	4096
				4096		32768	
						16384	
						8192	
						4096	

[1] Compare Memory Disabled
[2] Compare Memory Enabled

Table 4-6

MLENgth Maximum Memory Depth 4 Card System

64 Bit Width		32 Bit Width		16 Bit Width		8 Bit Width	
Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]
1047552	261120	2096128	523264	4193280	1047552	8387584	2096128
524288	131072	1048576	262144	3145728	524288	7340032	1048576
262144	65536	524288	131072	2097152	262144	6291456	524288
131072	32768	262144	65536	1048576	131072	5242880	262144
65536	16384	131072	32768	524288	65536	4194304	131072
32768	8192	65536	16384	262144	32768	3145728	65536
16384	4096	32768	8192	131072	16384	2097152	32768
8192		16384	4096	65536	8192	1048576	16384
4096		8192		32768	4096	524288	8192
		4096		16384		262144	4096
				8192		131072	
				4096		65536	
						32768	
						16384	
						8192	
						4096	

[1] Compare Memory Disabled
[2] Compare Memory Enabled

Table 4-7

MLENgth Maximum Memory Depth 5 Card System

80 Bit Width Disabled[1]	Enabled[2]	40 Bit Width Disabled[1]	Enabled[2]	16 Bit Width Disabled[1]	Enabled[2]	8 Bit Width Disabled[1]	Enabled[2]
1047552	261120	2096128	523264	5241856	1047552	10484736	2096128
524288	131072	1048576	262144	4194304	524288	9437184	1048576
262144	65536	524288	131072	3145728	262144	8388608	524288
131072	32768	262144	65536	2097152	131072	7340032	262144
65536	16384	131072	32768	1048576	65536	6291456	131072
32768	8192	65536	16384	524288	32768	5242880	65536
16384	4096	32768	8192	262144	16384	4194304	32768
8192		16384	4096	131072	8192	3145728	16384
4096		8192		65536	4096	2097152	8192
		4096		32768		1048576	4096
				16384		524288	
				8192		262144	
				4096		131072	
						65536	
						32768	
						16384	
						8192	
						4096	

[1] Compare Memory Disabled
[2] Compare Memory Enabled

NUMRec

Command

```
:TRIGger:NUMRec <num_records>
```

The NUMRec command allows you to specify the number of records the logic analyzer acquires. The NUMRec command is only valid when record mode is turned on and the analyzer is in the state mode.

> The number of records times the record length cannot exceed the total system memory depth.

`<num_records>` An integer from 1 to max_recs (See table 4-8)

Example

```
OUTPUT XXX;":TRIGger:NUMRec 1"
```

Query

```
:TRIGger:NUMRec?
```

The NUMRec query returns the number of records currently specified. The NUMRec query is only valid when record mode is turned on and the analyzer is in the state mode.

Returned Format

```
[TRIGger:NUMRec] <num_records><NL>
```

`<num_records>` An integer from 1 to max_recs (See table 4-8)

Example

```
OUTPUT XXX;":TRIGger:NUMRec?"
```

Table 4-8

NUMRec Maximum Number of Records

Number of Cards	80, 64, or 48 Bits Wide		40, 32, or 24 Bits Wide		16 Bits Wide		8 Bits Wide	
	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]
5	1048576	262144	2097152	524288	5242880	1048576	10485760	2097152
4	1048576	262144	2097152	524288	4194304	1048576	8388608	2097152
3	1048576	262144	2097152	524288	3145728	1048576	6291456	2097152
2			1048576	262144	2097152	524288	4194304	1048576
1					1048576	262144	2097152	524288

[1] Compare Memory Disabled
[2] Compare Memory Enabled

POSTstore

Command

```
:TRIGger:POSTstore <pstore_pct>
```

The POSTstore command allows you to specify the percentage of data acquired after the trigger. The poststore percentage must be an integer from 0 through 100. If a number less than 0 is specified, the analyzer sets the value to 0. If a number greater than 100 is specified, the analyzer sets the value to 100. In both cases an error condition is reported. If the percentage specified is not a multiple of 10, the value is rounded down to the nearest multiple of 10. The POSTstore command is valid only when record mode is turned off.

`<pstore_pct>` `{0|10|20|30|40|50|60|70|80|90|100}`

Example

```
OUTPUT XXX;":TRIGger:POSTstore 0"
```

Query `:TRIGger:POSTstore?`

The POSTstore query returns the current poststore percentage. The POSTstore query is only valid when record mode is turned off.

Returned Format `[TRIGger:POSTstore] <pstore_pct><NL>`

`<pstore_pct>` `{0|10|20|30|40|50|60|70|80|90|100}`

Example

```
OUTPUT XXX;":TRIGger:POSTstore?"
```

RECord

Command `:TRIGger:RECord {ON|OFF}`

The RECord command allows you to turn the record mode on or off when the analyzer is in the state mode.

Example

```
OUTPUT XXX;":TRIGger:RECord ON"
```

Query `:TRIGger:RECord?`

The RECord query returns the current record mode status when the analyzer is in the state mode.

Returned Format `[TRIGger:RECord] {ON|OFF}<NL>`

Example

```
OUTPUT XXX;":TRIGger:RECord?"
```

RLENgth

Command

```
:TRIGger:RLENgth <rec_length>
```

The RLENgth command allows you to specify the record length when record mode is turned on and the analyzer is in the state mode. Valid record lengths range from one state through the total system configuration depth. Recall that the system configuration depth varies depending on the number of cards in the system, the system width, and the use of the compare memory buffer.

> The record length times the number of records cannot exceed the total system memory depth.

`<rec_length>` An integer from 1 through max_depth (See table 4-9)

Example

```
OUTPUT XXX;":TRIGger:RLENgth 1024"
```

Query

```
:TRIGger:RLENgth?
```

The RLENgth query returns the current record length. The RLENgth query is only valid when record mode is turned on and the analyzer is in the state mode.

Returned Format

```
[TRIGger:RLENgth] <rec_length><NL>
```

`<rec_length>` An integer from 1 through max_depth (See table 4-9)

Example

```
OUTPUT XXX;":TRIGger:RLENgth?"
```

Table 4-9

RLENgth Maximum Record Length

Number of Cards	80, 64, or 48 Bits Wide		40, 32, or 24 Bits Wide		16 Bits Wide		8 Bits Wide	
	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]	Disabled[1]	Enabled[2]
5	1048576	262144	2097152	524288	5242880	1048576	10485760	2097152
4	1048576	262144	2097152	524288	4194304	1048576	8388608	2097152
3	1048576	262144	2097152	524288	3145728	1048576	6291456	2097152
2			1048576	262144	2097152	524288	4194304	1048576
1					1048576	262144	2097152	524288

[1] Compare Memory Disabled
[2] Compare Memory Enabled

STORe

Command

```
:TRIGger:STORe <store_qualifier>
```

The STORe command defines the store qualifier. Any data matching the STORe qualifier is stored in memory as part of the current trigger data. The qualifier may be a single term or an expression. The trigger qualifier terms A, B, C, and D are defined by the TERM command.

An expression contains parentheses and the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is unchanged.

> The STORe command is valid only when the analyzer is configured as a state analyzer. See the TYPE command in Chapter 2 for more information.

`<store_qualifier>` `{ANYState|NOSTate|<any_term>|(<expression>)}`

`<any_term>` `{A|B|C|D|NOTA|NOTB|NOTC|NOTD}`

`<expression>` `{<any_term> [OR <any_term>] [OR <any_term>]}`

Examples

```
OUTPUT XXX;":TRIGger:STORe ANYSTATE"
OUTPUT XXX;":TRIGger:STORe (NOTA OR NOTC OR NOTD)"
```

Query `:TRIGger:STORe?`

The STORe query returns the current store qualifier specification.

Returned Format `[:TRIGger:STORe] <store_qualifier><NL>`

`<store_qualifier>` `{ANYState|NOSTate|<any_term>|(<expression>)}`

`<any_term>` `{A|B|C|D|NOTA|NOTB|NOTC|NOTD}`

`<expression>` `{<any_term> [OR <any_term>] [OR <any_term>]}`

Example

```
OUTPUT XXX;":TRIGger:STORe?"
```

TERM

Command `:TRIGger:TERM <term_id>,<label_name>,<pattern>`

The TERM command allows you to specify a pattern recognizer term. Each command modifies only one label in the given term. Because of this, a complete specification could require several commands. Since a label can contain 32 or less bits, the range of the pattern value is from 2^{32} - 1 through 0. Since the pattern parameter may contain don't cares and be represented in several bases, it is handled as a string of characters rather than a number.

`<term_id>` `{A|B|C|D}`

`<label_name>` A string of up to 6 alphanumeric characters

`<pattern>`
```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX;":TRIGger:TERM A,'DATA','255'"
OUTPUT XXX;":TRIGger:TERM B,'ABC','#BXXXX1101' "
```

Query

`:TRIGger:TERM? <term_id>,<label_name>`

The TERM query returns the specification of the term specified by term identification and label name.

Returned Format

`[:TRAce:TERM] <term_id>,<label_name>,<pattern><NL>`

`<term_id>` `{A|B|C|D}`

`<label_name>` A string of up to 6 alphanumeric characters

`<pattern>`
```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX;":TRIGger:TERM? B,'DATA' "
```

5

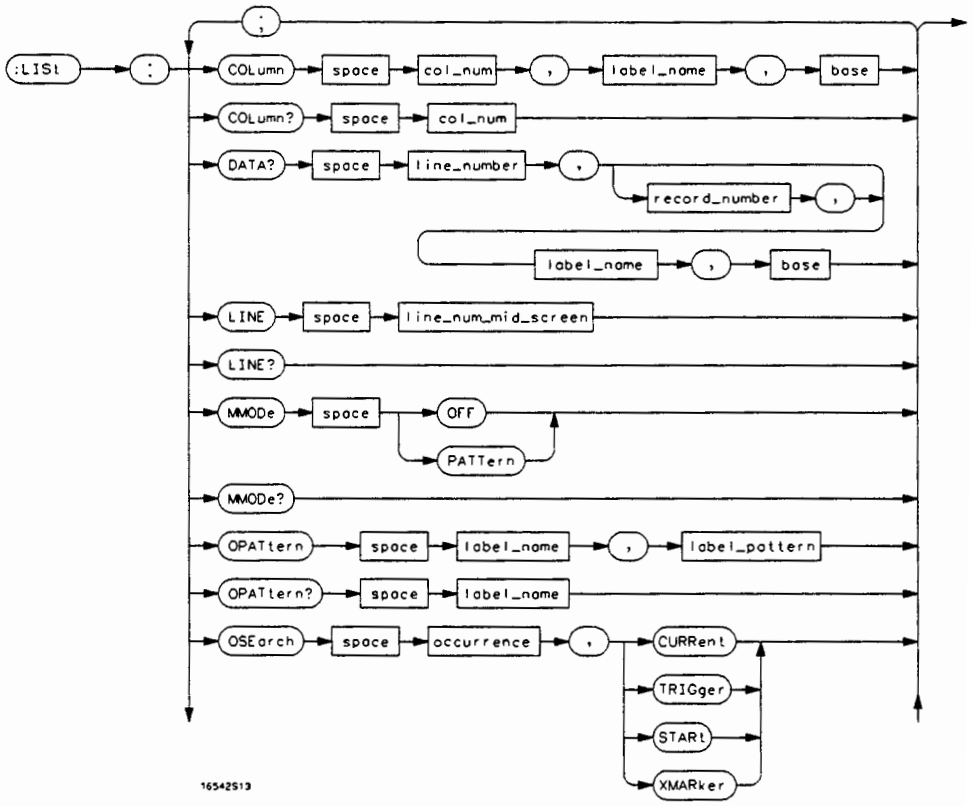
LIST Subsystem

Introduction

The LIST subsystem contains the following commands available for the State Listing menu in the HP 16542A logic analyzer module:

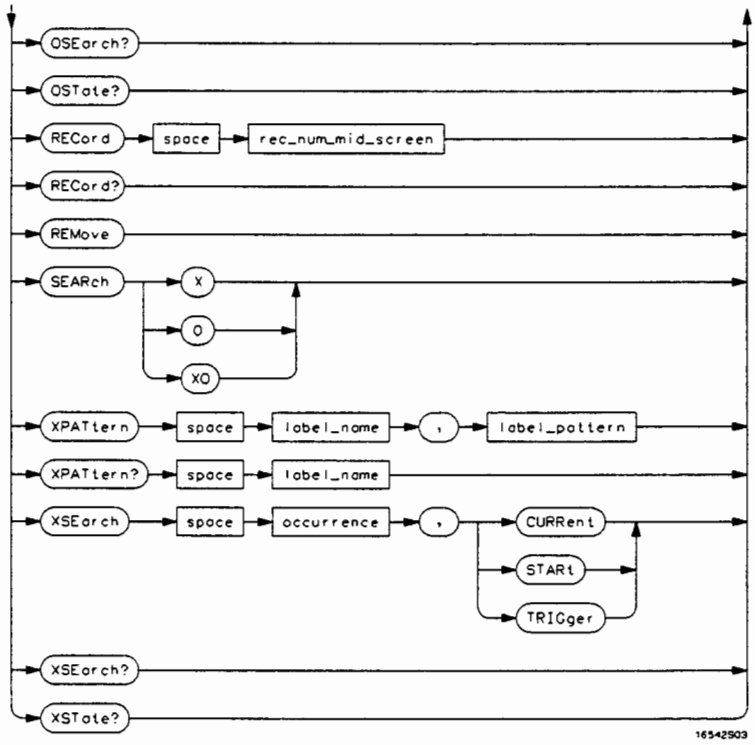
- COLumn
- DATA
- LINE
- MMODe
- OPATtern
- OSEarch
- OSTate
- RECord
- REMove
- SEARch
- XPATtern
- XSEarch
- XSTate

Figure 5-1



LIST Subsystem Syntax Diagram

Figure 5-1



LIST Subsystem Syntax Diagram (continued)

Table 5-1 **LIST Parameter Values**

Parameter	Values
col_num	integer from 1 to 61
line_number	integer from -10,484,736 to +10,484,736
label_name	a string of up to 6 alphanumeric characters
base	Any of the following keywords: `BINary, HEXadecimal, OCTal, DECimal, ASCii, SYMBol, IASSembler, or TWOS`
line_num_mid_screen	integer from -10,484,736 to +10,484,736

label_pattern	Any one of the following:
	`#B{0|1|X}...`
	`#Q{0|1|2|3|4|5|6|7|X}...`
	`#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...`
	`{0|1|2|3|4|5|6|7|8|9}...`

occurrence	integer from -10,484,736 to +10,484,736
rec_num_mid_screen	integer from 0 to maximum number of records
record_number	integer from 0 to 10,484,736

Table 5-2 **LIST Power On Default Values**

Command	Default
COLumn	1,'Lab1',HEX
MMODe	OFF

LIST

Selector `:LIST`

The LIST selector is used as part of a compound header to access those settings normally found in the Listing menu.

Example

```
OUTPUT XXX;":LIST:LINE 256"
```

COLumn

Command

```
:LIST:COLumn <col_num>,<label_name>,<base>
```

The COLumn command allows you to configure the analyzer listing display by assigning a label name and base to one of the 61 vertical columns in the menu. The left most column is column 1 and the right most column is column 61. When a label is assigned to a column it replaces the original label in that column. If there is no room in the list, an error is returned.

`<col_num>` An integer from 1 to 61

`<label_name>` A string of up to 6 alphanumeric characters

`<base>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler|TWOS}`

Examples

```
OUTPUT XXX;":LIST:COLumn 4,'A',HEX"
```

Query

```
:LIST:COLumn? <col_num>
```

The COLumn query returns the column number, label name, and base for the specified column. If the requested column number is not part of the listing, the last column number's information is returned.

Returned Format

```
[:LIST:COLumn] <col_num>,<label_name>,<base> <NL>
```

`<col_num>` An integer from 1 to 61

`<label_name>` A string of up to 6 alphanumeric characters

`<base>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler|TWOS}`

Example

```
OUTPUT XXX;":LIST:COLumn? 4"
```

DATA

Query

```
:LIST:DATA? <line_number>,[<record_number>,]
<label_name>,<base>
```

The DATA query returns the data value at the specified line number for the given label. If the analyzer is in record mode, the data value at the specified line and record numbers is returned. The returned data value is a string in the base specified by the base parameter.

Returned Format

```
[LIST:DATA] <line_number>,[<record_number>,]
<label_name>,<pattern_string><NL>
```

`<line_number>` An integer from -10,484,736 to +10,484,736

`<record_number>` An integer from 0 to 10,484,736

`<label_name>` A string of up to six alphanumeric characters

`<base>`

```
{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|
IASSembler}
```

`<pattern_string>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}...}
```

Example

```
OUTPUT XXX;":LIST:DATA? 4,'Data',HEX"
```

LINE

Command

```
:LIST:LINE <line_mid_screen>
```

The LINE command allows you to scroll the analyzer listing vertically. The command tells the analyzer to highlight the specified line number at the center of the screen. The line number is specified relative to the trigger. If record mode is enabled, the line number must be a line number within the current record.

`<line_mid_screen>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":LIST:LINE 10000"
```

Query

```
:LIST:LINE?
```

The LINE query returns the line number currently highlighted at the center of the screen.

Returned Format:

```
[:LIST:LINE] <line_mid_screen><NL>
```

`<line_mid_screen>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":LIST:LINE?"
```

MMODe

Command

```
:LIST:MMODe <marker_mode>
```

The MMODe command selects the marker mode which controls the movement of the X and O markers and the marker readouts. When PATTern is selected, the markers are placed on the patterns specified by the XPATtern and OPATtern commands. An error is returned if the analyzer is not in the state mode.

`<marker_mode>` `{OFF|PATTern}`

Example

```
OUTPUT XXX;":LIST:MMODe PATTern"
```

Query

```
:LIST:MMODe?
```

The MMODe query returns the current marker mode selected.

Returned Format

```
[:LIST:MMODe] <marker_mode><NL>
```

`<marker_mode>` `{OFF|PATTern}`

Example

```
OUTPUT XXX;":LIST:MMODe?"
```

OPATtern

Command

```
:LIST:OPATtern <label_name>,<label_pattern>
```

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is used with the OSEarch criteria when moving the marker. Since this command deals with a single label at a time, a complete specification could require several invocations. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Because the maximum number of bits allowed in a label is 32, the pattern value must be in the range of 0 to 2^{32} - 1 inclusive. The label_pattern parameter is handled as a string of characters rather than a number because it may contain don't cares.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Examples

```
OUTPUT XXX;":LIST:OPATtern 'DATA','255' "
OUTPUT XXX;":LIST:OPATtern 'ABC','#BXXXX1101' "
```

Query

```
:LIST:OPATtern? <label_name>
```

The OPATtern query returns the pattern recognizer term for a given label name. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is reported.

Returned Format

```
[:LIST:OPATtern] <label_name>,<label_pattern><NL>
```

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Example

```
OUTPUT XXX;":LIST:OPATtern? 'A'"
```

OSEarch

Command

```
:LIST:OSEarch <occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker. The OSEarch command tells the analyzer to search for the pattern as specified by the OPATtern command and place the O marker on this pattern. The origin parameter tells the analyzer to begin the search from the trigger, from the start of data, from the current marker position, or from the X marker.

The actual number of patterns searched by the analyzer is determined by the occurrence parameter. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin. An error is returned if the analyzer is not in the state mode.

`<occurrence>` An integer from -10,484,736 to +10,484,736

`<origin>` `{TRIGger|STARt|XMARker|CURRent}`

Example

```
OUTPUT XXX;":LIST:OSEarch +10,TRIGGER"
```

Query

```
:LIST:OSEarch?
```

The OSEarch query returns the search criteria for the O marker.

Returned Format

```
[:LIST:OSEarch] <occurrence>,<origin><NL>
```

`<occurrence>` An integer from -10,484,736 to +10,484,736

`<origin>` `{TRIGger|STARt|XMARker|CURRent}`

Example

```
OUTPUT XXX;":LIST:OSEarch?"
```

OSTate

Query

```
:LIST:OSTate?
```

The OSTate query returns the line and record number in the listing where the O marker resides (-10,484,736 to +10,484,736). If the specified number of occurrences of the pattern or the specified pattern is not in the listing, the number returned is 2,147,483,647 which indicates an error condition.

Returned Format

```
[:LIST:OSTate] [<record_num>,]<state_num><NL>
```

`<record_num>` An integer from 0 to +10,484,736

`<state_num>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":LIST:OSTate?"
```

RECord

Command

```
:LIST:RECord <rec_mid_screen>
```

The RECord command allows you to scroll the analyzer listing vertically on a record basis. This command tells the analyzer which record to highlight in the center of the screen. The RECord command is only valid when Record Mode is enabled.

`<rec_mid_screen>` An integer from 0 to the maximum number of records

Example

```
OUTPUT XXX;":LIST:RECord 5"
```

Query

```
:LIST:RECord?
```

The RECord query returns the record number currently highlighted at the center of the screen.

Returned Format

```
[LIST:RECord] <rec_mid_screen><NL>
```

<rec_mid_screen> An integer from 0 to the maximum number of records

Example

```
OUTPUT XXX;":LIST:RECord?"
```

REMove

Command

```
:LIST:REMove
```

The REMove command allows you to delete all labels from the LISting display except for the leftmost column.

Example

```
OUTPUT XXX;":LIST:REMove"
```

SEARch

Command

```
:LISt:SEARch <search_opt>
```

The SEARch command starts the search for the specified marker. The markers are placed according to the current search criteria as previously defined by the OPATtern, OSEarch, XPATtern, and XSEarch commands. An error is returned if the analyzer is not in pattern marker mode.

`<search_opt>` `{X|O|XO}`

Example

```
OUTPUT XXX;":LIST:SEARch"
```

XPATtern

Command

```
:LIST:XPATtern <label_name>,<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is used with the XSEarch criteria when moving the marker. Since this command deals with a single label at a time, a complete specification could require several invocations. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Because the maximum number of bits allowed in a label is 32, the pattern value must be in the range of 0 to 2^{32} - 1 inclusive. The label_pattern parameter is handled as a string of characters rather than a number because it may contain don't cares.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Examples

```
OUTPUT XXX;":LIST:XPATtern 'DATA','255' "
OUTPUT XXX;":LIST:XPATtern 'ABC','#BXXXX1101' "
```

Query

```
:LIST:XPATtern? <label_name>
```

The XPATtern query returns the pattern specification for a given label name. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Returned Format `[:LIST:XPATtern] <label_name>,<label_pattern><NL>`

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Example

```
OUTPUT XXX;":LIST:XPATtern? 'A'"
```

XSEarch

Command `:LIST:XSEarch <occurrence>,<origin>`

The XSEarch command defines the search criteria for the X marker. The XSEarch command tells the analyzer to search for the pattern as specified by the XPATtern command and place the X marker on this pattern. The origin parameter tells the analyzer to begin the search from the trigger, from the start of data, or from the current marker position.

The actual number of patterns searched by the analyzer is determined by the occurrence parameter. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin. An error is returned if the analyzer is not in pattern marker mode.

`<occurrence>` An integer from -10,484,736 to +10,484,736

`<origin>` `{TRIGger|STARt|CURRent}`

Example

```
OUTPUT XXX;":LIST:XSEarch +10,TRIGGER"
```

Query `:LIST:XSEarch?`

The XSEarch query returns the search criteria for the X marker.

Returned Format `[:LIST:XSEarch] <occurrence>,<origin> <NL>`

`<occurrence>` An integer from -10,484,736 to +10,484,736

`<origin>` `{TRIGger|STARt|CURRent}`

Example

```
OUTPUT XXX;":LIST:XSEarch?"
```

XSTate

Query `:LIST:XSTate?`

The XSTate query returns the line and record number in the listing where the X marker resides (-10,484,736 to +10,484,736). If the specified number of occurrences of the pattern or the specified pattern is not in the listing, the number returned is 2,147,483,647 which indicates and error condition.

Returned Format `[:LIST:XSTate] [<rec_num>,]<state_num><NL>`

`<rec_num>` An integer from 0 to +10,484,736

`<state_num>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":LIST:XSTate?"
```

6

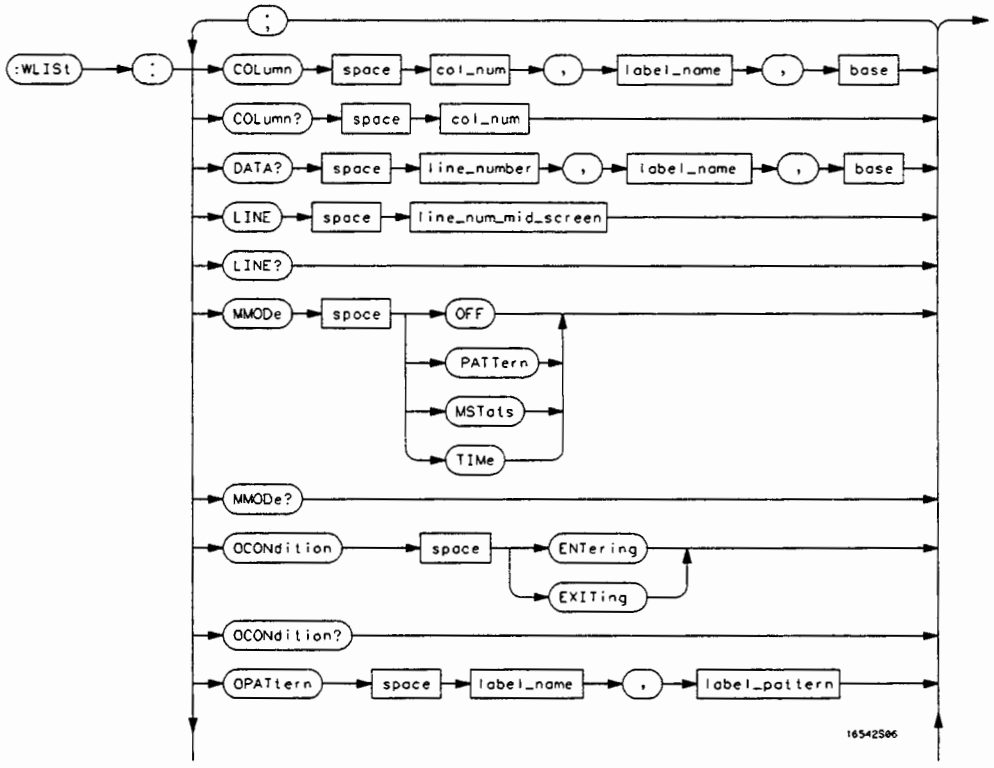
WLISt Subsystem

Introduction

The WLISt subsystem contains the following commands available for the WLISting menu in the HP 16542A logic analyzer module:

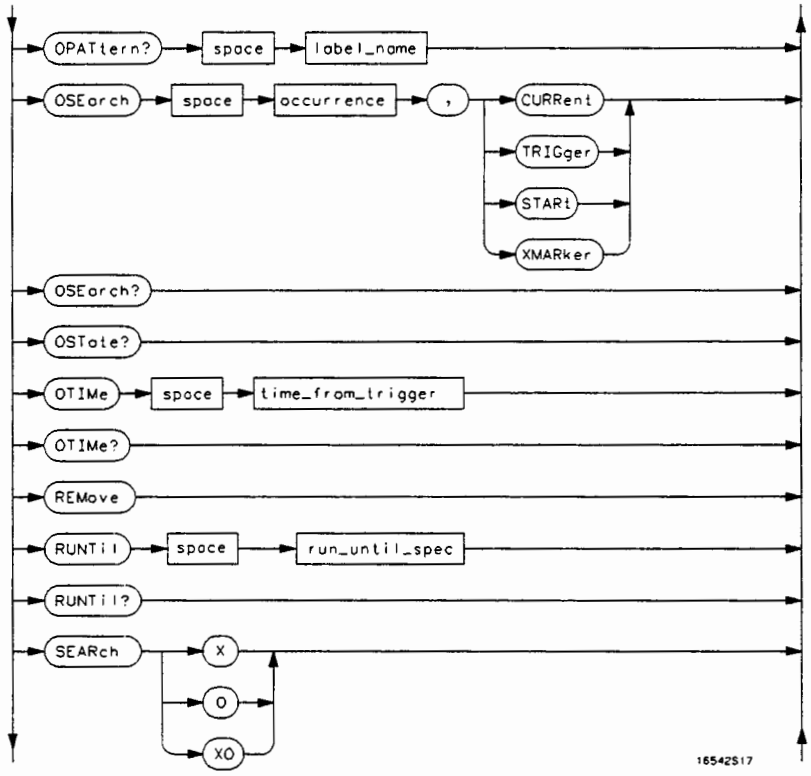
- COLumn
- DATA
- LINE
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OSTate
- OTIMe
- REMove
- RUNTil
- SEARch
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XCONdition
- XOTime
- XPATtern
- XSEarch
- XSTate
- XTIMe

Figure 6-1



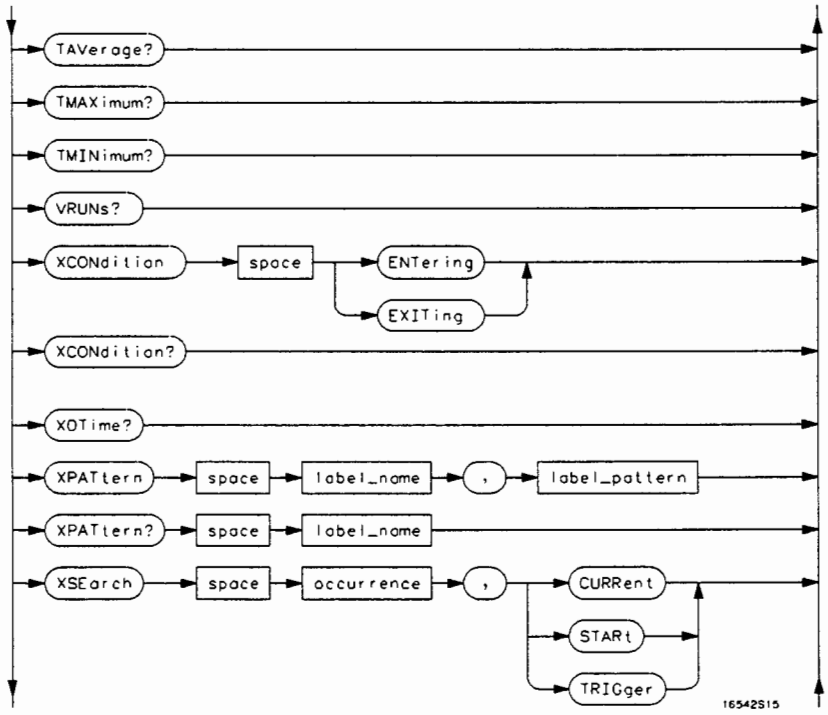
WLISt Subsystem Syntax Diagram

Figure 6-1



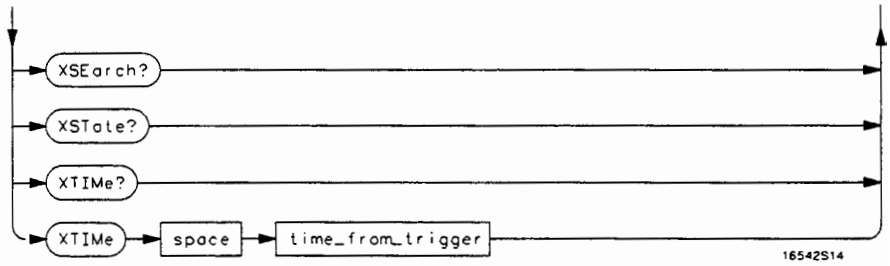
WLISt Subsystem Syntax Diagram (cont.)

Figure 6-1



WLISt Subsystem Syntax Diagram (cont.)

Figure 6-1



WLISt Subsystem Syntax Diagram (cont.)

Table 6-1 WLISt Parameter Values

Parameter	Values
base	Any of the following keywords: `BINary, HEXadecimal, OCTal, DECimal, ASCii, SYMBol, IASSembler, or TWOS`
col_num	integer from 1 to 61
line_number	integer from -10,484,736 to +10,484,736
label_name	a string of up to 6 alphanumeric characters
line_num_mid_screen	integer from -10,484,736 to +10,484,736

label_pattern	Any one of the following:
	`#B{0|1|X}...`
	`#Q{0|1|2|3|4|5|6|7|X}...`
	`#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...`
	`{0|1|2|3|4|5|6|7|8|9}...`

occurrence	integer from -10,484,736 to +10,484,736

run_until_spec	Any of the following
	`OFF`
	`LT, <value>`
	`GT, <value>`
	`EQUal`
	`NEQual`
	`INRange, <value>, <value>`
	`OUTRange, <value>, <value>`

value	real number
time_from_trigger	real from -104.84736 ms to +104.84736 ms

Table 6-2 WLISt Power On Default Values

Command	Default
COLumn	1,'Lab1',HEX
MMODe	OFF

WLISt

Selector

```
:WLISt
```

The WLISt selector is used as part of a compound header to access those settings normally found in the WLISting menu.

Example

```
OUTPUT XXX;":WLISt:LINE 256"
```

COLumn

Command

```
:WLISt:COLumn <col_num>,<label_name>,<base>
```

The COLumn command allows you to configure the analyzer WLISting display by assigning a label name and base to one of the 61 vertical columns in the menu. The left most column is column 1 and the right most column is column 61. When a label is assigned to a column it replaces the original label in that column. If there is no room in the list, an error is returned.

`<col_num>` integer from 1 to 61

`<label_name>` A string of up to 6 alphanumeric characters

`<base>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler|TWOS}`

Example

```
OUTPUT XXX;":WLISt:COLumn 4,'A',HEX"
```

Query

```
:WLISt:COLumn? <col_num>
```

The COLumn query returns the column number, label name, and base for the specified column. If the requested column is not part of the list display, the last column number information is returned.

Returned Format `[:WLISt:COLumn] <col_num>,<label_name>,<base> <NL>`

`<col_num>` An integer from 1 to 61

`<label_name>` A string of up to 6 alphanumeric characters

`<base>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler|TWOS}`

Example

```
OUTPUT XXX;":WLISt:COLumn? 4"
```

DATA

Query `:WLISt:DATA? <line_number>,<label_name>,<base>`

The DATA query returns the data value at a specified line number for a given label. The returned data value is a string in the base specified by the base parameter.

Returned Format `[WLISt:DATA] <line_number>,<label_name>,<pattern_string><NL>`

`<line_number>` An integer from -10,484,736 to +10,484,736

`<label_name>` A string of up to six alphanumeric characters

`<base>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler}`

`<pattern_string>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}...}
```

Example

```
OUTPUT XXX;":WLISt:DATA? 4,'Data',HEX"
```

LINE

Command

```
:WLISt:LINE <line_mid_screen>
```

The LINE command allows you to scroll the analyzer WLISting vertically. The command tells the analyzer to highlight the specified line number at the center of the screen. The line number is specified relative to the trigger.

`<line_mid_screen>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":WLISt:LINE 10000"
```

Query

```
:WLISt:LINE?
```

The LINE query returns the line number currently highlighted at the center of the screen.

Returned Format:

```
[:WLISt:LINE] <line_mid_screen><NL>
```

`<line_mid_screen>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":WLISt:LINE?"
```

MMODe

Command

```
:WLISt:MMODe <marker_mode>
```

The MMODe command selects the marker mode which controls the movement of the X and O markers and the marker readouts. When PATTern is selected, the markers are placed on the patterns specified by the XPATtern and OPATtern commands.

Selecting TIME, places the markers on time values with respect to the trigger, as specified by the XTIMe and OTIMe commands. In MSTats mode, markers are placed on patterns but the readouts display the number of valid runs, the minimum X to O marker time, the maximum X to O marker time, and the average X to O marker time.

An error is returned if the analyzer is not in timing mode.

<marker_mode> {OFF|PATTern|TIME|MSTats}

Example

```
OUTPUT XXX;":WLISt:MMODe PATTern"
```

Query

```
:WLISt:MMODe?
```

The MMODe query returns the current marker mode selected.

Returned Format

```
[:WLISt:MMODe] <marker_mode><NL>
```

<marker_mode> {OFF|PATTern|TIME|MSTat}

Example

```
OUTPUT XXX;":WLISt:MMODe?"
```

OCONdition

Command

```
:WLISt:OCONdition {ENTering|EXITing}
```

The OCONdition command specifies where the O marker is placed. The O marker can be placed on the entry or exit point of the OPATtern when in the PATTern marker mode.

An error is returned if the analyzer is not in the pattern marker mode.

Example

```
OUTPUT XXX; ":WLISt:OCONdition ENTERING"
```

Query

```
:WLISt:OCONdition?
```

The OCONdition query returns the current setting.

Returned Format

```
[:WLISt:OCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":WLISt:OCONdition?"
```

OPATtern

Command

```
:WLISt:OPATtern <label_name>,<label_pattern>
```

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is used with the OSEarch criteria when moving the marker. Since this command deals with a single label at a time, a complete specification could require several invocations. The analyzer must be in the pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Because the maximum number of bits allowed in a label is 32, the pattern value must be in the range of 0 to 2^{32} - 1 inclusive. The label_pattern parameter is handled as a string of characters rather than a number because it may contain don't cares.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Examples

```
OUTPUT XXX;":WLISt:OPATtern 'DATA','255' "
OUTPUT XXX;":WLISt:OPATtern 'ABC','#BXXXX1101' "
```

Query

```
:WLISt:OPATtern? <label_name>
```

In state marker mode, the OPATtern query returns the pattern recognizer term for a given label name. In timing marker mode, the pattern for the current O marker position is returned.

Returned Format

```
[:WLISt:OPATtern] <label_name>,<label_pattern><NL>
```

<label_name> A string of up to 6 alphanumeric characters

<label_pattern>

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Example

```
OUTPUT XXX;":WLISt:OPATtern? 'A'"
```

OSEarch

Command

```
:WLISt:OSEarch <occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker. The OSEarch command tells the analyzer to search for the pattern as specified by the OPATtern command and place the O marker on this pattern. The origin parameter tells the analyzer to begin the search from the trigger, from the start of data, from the current marker position, or from the X marker.

The actual number of patterns searched by the analyzer is determined by the occurrence parameter. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin. An error is returned if the analyzer is not in the pattern marker mode.

<occurrence> An integer from -10,484,736 to +10,484,736

<origin> `{TRIGger|STARt|XMARker|CURRent}`

Example

```
OUTPUT XXX;":WLISt:OSEarch +10,TRIGGER"
```

Query

```
:WLISt:OSEarch?
```

The OSEarch query returns the search criteria for the O marker.

Returned Format

```
[:WLISt:OSEarch] <occurrence>,<origin><NL>
```

<occurrence> An integer from -10,484,736 to +10,484,736

<origin> `{TRIGger|STARt|XMARker|CURRent}`

Example

```
OUTPUT XXX;":WLISt:OSEarch?"
```

OSTate

Query

```
:WLISt:OSTate?
```

The OSTate query returns the line number in the WLISting where the O marker resides (-10,484,736 to +10,484,736). If the specified number of occurrences of the pattern or the specified pattern is not in the WLISting, the number returned is 2,147,483,647 which indicates an error condition.

Returned Format

```
[:WLISt:OSTate] <state_num><NL>
```

<state_num> An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":WLISt:OSTate?"
```

OTIMe

Command

```
:WLISt:OTIMe <time_from_trigger>
```

The OTIMe command places the O marker on the data value whose distance from the trigger is specified by the time_from_trigger parameter. An error is returned if the analyzer is not in the timing mode with time markers on.

`<time_from_trigger>` A real from -104.84736 ms to +104.84736 ms

Example

```
OUTPUT XXX;":WLISt:OTIMe 10E-6"
```

Query

```
:WLISt:OTIMe?
```

The OTIMe query returns the time of the line number where the O marker resides.

Returned Format

```
[:WLISt:OTIMe] <time_from_trigger><NL>
```

`<time_from_trigger>` A real from -104.84736 ms to +104.84736 ms

Example

```
OUTPUT XXX;":WLISt:OTIMe?"
```

REMove

Command

```
:WLISt:REMove
```

The REMove command allows you to delete all labels from the WLISting display except for the leftmost column.

Example

```
OUTPUT XXX;":WLISt:REMove'"
```

RUNTil

Command

```
:WLISt:RUNTil <run_until_spec>
```

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers when the trigger mode is in repetitive. When OFF is selected, the analyzer will run until either the "STOP" touch screen field is touched or the STOP command is sent. Run until the time between X and O marker options are:

- Less Than (LT) a specified time value
- Greater Than (GT) a specified time value
- In the range (INRange) between two time values
- Out of the range (OUTRange) between two time values

There are two conditions that are based on a comparison of the acquired data and the compare data image. The analyzer will run until one of the following conditions is true:

- Every channel of every label has the same value (EQUal).
- Any channel of any label has a different value (NEQual).

An error is returned if the analyzer is not in timing mode.

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange<value>,<value>|OUTRange<value>,<value>|EQUal|NEQual}`

`<value>` A real number

Examples

```
OUTPUT XXX;":WLISt:RUNTil GT, 800.0E-6"
OUTPUT XXX;":WLISt:RUNTil INRANGe, 4.5, 5.5"
```

Query `:WLISt:RUNTil?`

The RUNTil query returns the current stop criteria.

Returned Format `[:WLISt:RUNTil] <run_until_spec><NL>`

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange<value>,<value>|OUTRange<value>,<value>|EQUal|NEQual}`

`<value>` A real number

Example

```
OUTPUT XXX;":WLISt:RUNTil?"
```

SEARch

Command `:WLISt:SEARch <search_opt>`

The SEARch command starts the search for the specified marker. The markers are placed according to the current search criteria as previously defined by the the OPATtern, OSEarch, XPATtern, and XSEarch commands. An error is returned if the analyzer is not in the pattern marker mode.

`<search_opt>` `{X|O|XO}`

Example

```
OUTPUT XXX;":WLISt:SEARch"
```

TAVerage

Query

```
:WLISt:TAVerage?
```

The TAVerage query returns the value of the average time between the X and O markers. If there is no valid data, the query returns 9.9E37.

An error is returned if the analyzer is not in timing mode with pattern markers turned on.

Returned Format

```
[:WLISt:TAVerage] <time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLISt:TAVerage?"
```

TMAXimum

Query

```
:WLISt:TMAXimum?
```

The TMAXimum query returns the value of the maximum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

An error is returned if the analyzer is not in timing mode with pattern markers turned on.

Returned Format

```
[:WLISt:TMAXimum] <time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLISt:TMAXimum?"
```

TMINimum

Query

```
:WLISt:TMINimum?
```

The TMINimum query returns the value of the minimum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

An error is returned if the analyzer is not in timing mode with pattern markers turned on.

Returned Format

```
[:WLISt:TMINimum] <time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":WLISt:TMINimum?"
```

VRUNs

Query

```
:WLISt:VRUNs?
```

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

An error is returned if the analyzer is not in timing mode with pattern markers turned on.

Returned Format

```
[:WLISt:VRUNs] <valid_runs>,<total_runs><NL>
```

`<valid_runs>` zero or positive integer

`<total_runs>` zero or positive integer

Example

```
OUTPUT XXX;":WLISt:VRUNs?"
```

XCONdition

Command

```
:WLISt:XCONdition {ENTering|EXITing}
```

The XCONdition command specifies where the X marker is placed. The X marker can be placed on the entry or exit point of the XPATtern when in the PATTern marker mode.

An error is returned if the analyzer is not in pattern marker mode.

Example

```
OUTPUT XXX; ":WLISt:XCONdition ENTERING"
```

Query

```
:WLISt:XCONdition?
```

The XCONdition query returns the current setting.

Returned Format

```
[:WLISt:XCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":WLISt:XCONdition?"
```

XOTime

Query

```
:WLISt:XOTime?
```

The XOTime query returns the time from the X marker to the O marker. If data is not valid, the query returns 9.9E37.

An error is returned if the analyzer is not in timing mode with timing markers turned on.

Returned Format

```
[:WLISt:XOTime] <time_value><NL>
```

`<time_value>` A real number between + 105 and - 105 ms

Example

```
OUTPUT XXX;":WLISt:XOTime?"
```

XPATtern

Command

```
:WLISt:XPATtern <label_name>,<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is used with the XSEarch criteria when moving the marker. Since this command deals with a single label at a time, a complete specification could require several invocations. The analyzer must be in the pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Because the maximum number of bits allowed in a label is 32, the pattern value must be in the range of 0 to 2^{32} - 1 inclusive. The label_pattern parameter is handled as a string of characters rather than a number because it may contain don't cares.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Examples

```
OUTPUT XXX;":WLISt:XPATtern 'DATA','255' "
OUTPUT XXX;":WLISt:XPATtern 'ABC','#BXXXX1101' "
```

Query

```
:WLISt:XPATtern? <label_name>
```

The XPATtern query, in pattern marker mode, returns the X marker pattern specification for a given label name. In timing marker mode, the data for the label at the current X marker position is returned.

Returned Format

```
[:WLISt:XPATtern] <label_name>,<label_pattern><NL>
```

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X}...|
 #Q{0|1|2|3|4|5|6|7|X}...|
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...|
   {0|1|2|3|4|5|6|7|8|9}... }
```

Example

```
OUTPUT XXX;":WLISt:XPATtern? 'A'"
```

XSEarch

Command

```
:WLISt:XSEarch <occurrence>,<origin>
```

The XSEarch command defines the search criteria for the X marker. The XSEarch command tells the analyzer to search for the pattern, as specified by the XPATtern command, and place the X marker on this pattern. The origin parameter tells the analyzer to begin a search from the trigger, from the current marker position, or from the start of data.

The actual number of patterns searched by the analyzer is determined by the occurrence parameter. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin. An error is returned if the analyzer is not in the pattern marker mode.

`<occurrence>` An integer from -10,484,736 to +10,484,736

`<origin>` `{TRIGger|STARt|CURRent}`

Example

```
OUTPUT XXX;":WLISt:XSEarch +10,TRIGGER"
```

Query `:WLISt:XSEarch?`

The XSEarch query returns the search criteria for the X marker.

Returned Format `[:WLISt:XSEarch] <occurrence>,<origin> <NL>`

`<occurrence>` An integer from -10,484,736 to +10,484,736

`<origin>` `{TRIGger|STARt|CURRent}`

Example

```
OUTPUT XXX;":WLISt:XSEarch?"
```

XSTate

Query `:WLISt:XSTate?`

The XSTate query returns the line number in the WLISting where the X marker resides (-10,484,736 to +10,484,736). If the specified number of occurrences of the pattern or the specified pattern is not in the WLISting, the number returned is 2,147,483,647 which indicates an error condition.

Returned Format `[:WLISt:XSTate] <state_num><NL>`

`<state_num>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":WLISt:XSTate?"
```

XTIMe

Command

```
:WLISt:XTIMe <time_from_trigger>
```

The XTIMe command places the X marker on the data value whose distance from the trigger is specified by the time_from_trigger parameter.

An error is returned if the analyzer is not in timing mode with time markers on.

`<time_from_trigger>` A real from -104.84736 ms to +104.84736 ms

Example

```
OUTPUT XXX;":WLISt:XTIMe 10E-6"
```

Query

```
:WLISt:XTIMe?
```

The XTIMe query returns the time of the line number where the X marker resides.

Returned Format

```
[:WLISt:XTIMe] <time_from_trigger><NL>
```

`<time_from_trigger>` A real from -104.84736 ms to +104.84736 ms

Example

```
OUTPUT XXX;":WLISt:XTIMe?"
```

7

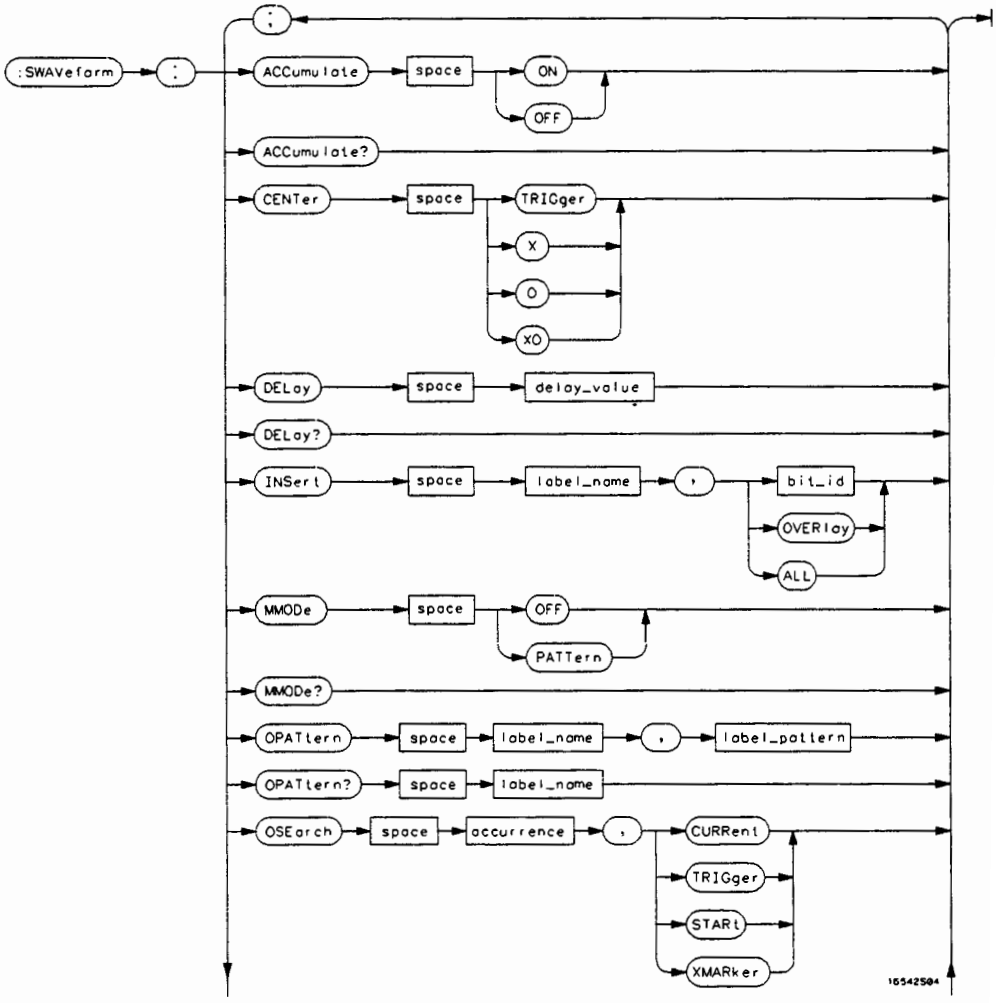
SWAVeform Subsystem

Introduction

The SWAVeform subsystem contains the commands available for the State Waveforms menu in the HP 16542A. These commands are listed below:

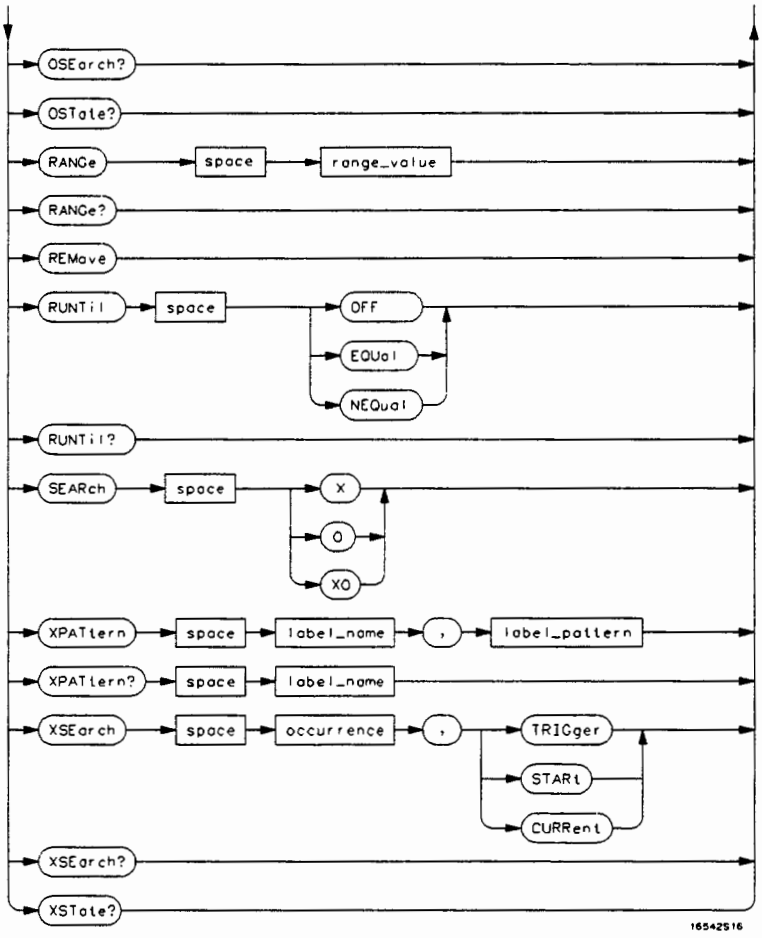
- ACCumulate
- CENTer
- DELay
- INSert
- MMODe
- OPATtern
- OSEarch
- OSTate
- RANGe
- REMove
- RUNTil
- SEARch
- XPATtern
- XSEarch
- XSTate

Figure 7-1



SWAVeform Subsystem Syntax Diagram

Figure 7-1



SWAVeform Subsystem Syntax Diagram

Table 7-1 **SWAVeform Parameter Values**

Parameter	Value
delay_value	integer from -10,484,736 to +10,484,736
bit_id	integer from 0 to 31
label_name	string of up to 6 alphanumeric characters

label_pattern	Any one of the following:
	`{#B{0|1|X}...`
	`#Q{0|1|2|3|4|5|6|7|X}...`
	`#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} ...`
	`{0|1|2|3|4|5|6|7|8|9}...}`

occurrence	integer from -10,484,736 to +10,484,736
range_value	integer from 10 to 10,484,736

Table 7-2 **Power On Defaults**

Command	Default
ACCumulate	OFF
DELay	0
MMODe	OFF
RANGe	10 (1 st/div)

SWAVeform

Selector

```
:SWAVeform
```

The SWAVeform selector is used as part of a compound header to access the settings found in the State Waveforms menu.

Example

```
OUTPUT XXX;":SWAVeform:DELAY -175"
```

ACCumulate

Command

```
:SWAVeform:ACCumulate <setting>
```

The ACCumulate command allows you to control whether the state waveform display gets erased between each individual run or whether subsequent waveforms are displayed over the previous ones.

`<setting>` `{0|OFF} or {1|ON}`

Example

```
OUTPUT XXX;":SWAVeform:ACCumulate ON"
```

Query

```
:SWAVeform:ACCumulate?
```

The ACCumulate query returns the current setting. The query always shows the setting as the number 0 (off) or 1 (on).

Returned Format

```
[:SWAVeform:ACCumulate] {0|1}<NL>
```

Example

```
OUTPUT XXX;":SWAVeform:ACCumulate?"
```

CENTer

Command

```
:SWAVeform:CENTer <center_arg>
```

The CENTer command redraws the waveforms centering them on the screen as specified by the center argument.

An error is returned if markers are not on or if the display is not set to the state waveform menu.

`<center_arg>` `{TRIGger|X|O|XO}`

Example

```
OUTPUT XXX;":SWAVeform:CENTer XO"
```

DELay

Command

```
:SWAVeform:DELay <delay_value>
```

The DELay command specifies the state from the trigger shown in the horizontal center of the state waveform display.

An error is returned if the analyzer is not in the state mode.

`<delay_value>` An integer in the range of ± the maximum memory length.

Example

```
OUTPUT XXX;":SWAVeform:DELay 29000"
```

Query

```
:SWAVeform:DELay?
```

The DELay query returns the current state delay value.

Returned Format

```
[:SWAVeform:DELay] <delay_value><NL>
```

`<delay_value>` An integer in the range of ± the maximum memory depth.

Example

```
OUTPUT XXX;":SWAVeform:DELay?"
```

INSert

Command

```
:SWAVeform:INSert <label_name>
[,{<bit_id>|OVERlay|ALL}]
```

The INSert command inserts waveforms in the state waveform display. The waveforms are added from top to bottom. When 96 waveforms are present, inserting additional waveforms replaces the last waveform.

The first parameter specifies the label name inserted. The optional second parameter specifies the label bit number, overlay, or all. If a number is specified, only the waveform for that bit number is added to the screen.

If OVERlay is specified, all the bits of the label are displayed as a composite overlaid waveform. If ALL is specified, all the bits are displayed sequentially. If the second parameter is not specified, ALL is assumed.

`<label_name>` A string of up to 6 alphanumeric characters

`<bit_id>` An integer from 0 to 31

Example

```
OUTPUT XXX;":SWAVeform:INSert 'WAVE',10"
```

MMODe

Command

```
:SWAVeform:MMODe {OFF|PATTern}
```

The MMODe (Marker Mode) command selects the mode controlling marker movement and the display of the marker readouts. When PATTern is selected, the markers are placed on patterns as specified by the OPATtern and XPATtern commands.

An error is returned if the analyzer is not in state mode.

Example

```
OUTPUT XXX; ":SWAVeform:MMODe PATTern"
```

Query `:SWAVeform:MMODe?`

The MMODe query returns the current marker mode.

Returned Format `[:SWAVeform:MMODe] <marker_mode><NL>`

<marker_mode> `{OFF|PATTern}`

Example

```
OUTPUT XXX;":SWAVeform:MMODe?"
```

OPATtern

Command `:SWAVeform:OPATtern <label_name>,<label_pattern>`

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is used with the OSEarch criteria when moving the marker. Since this command deals with a single label at a time, a complete specification could require several invocations. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Because the maximum number of bits allowed in a label is 32, the pattern value must be in the range of 0 to 2^{32} - 1 inclusive. The label_pattern parameter is handled as a string of characters rather than a number because it may contain don't cares.

<label_name> A string of up to 6 alphanumeric characters

<label_pattern>

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX; ":SWAVeform:OPATtern 'A','511'"
```

Query

```
:SWAVeform:OPATtern? <label_name>
```

The OPATtern query, in pattern marker mode, returns the pattern specification for a given label name.

Returned Format

```
[:SWAVeform:OPATtern] <label_name>,<label_pattern><NL>
```

<label_name> A string of up to 6 alphanumeric characters

<label_pattern>

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX;":SWAVeform:OPATtern? 'A'"
```

OSEarch

Command

```
:SWAVeform:OSEarch <occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker which is used with the associated OPATtern recognizer specification when moving markers on patterns. The origin parameter tells the marker to begin a search from the trigger, from the start, from the X marker or from the current marker position.

The actual occurrence the marker searches for is determined by the occurrence parameter, relative to the origin. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin. An error is returned if the analyzer if not in the pattern marker mode.

<origin> `{TRIGger|STARt|XMARker|CURRent}`

<occurrence> An integer from -10484736 to +10484736

Example

```
OUTPUT XXX; ":SWAVeform:OSEarch +10,TRIGGER"
```

Query

```
:SWAVeform:OSEarch?
```

The OSEarch query returns the search criteria for the O marker.

Returned Format

```
[:SWAVeform:OSEarch] <occurrence>,<origin><NL>
```

<origin>

```
{TRIGger|STARt|XMARker|CURRent}
```

<occurrence> An integer from -10484736 to +10484736

Example

```
OUTPUT XXX;":SWAVeform:OSEarch?"
```

OSTate

Query

```
:SWAVeform:OSTate?
```

The OSTate query returns the line and record number in the listing where the O marker resides (-10,484,736 to +10,484,736). If the specified number of occurrences of the pattern or the specified pattern is not in the listing, the number returned is 2,147,483,647 which indicates an error condition.

Returned Format

```
[:SWAVeform:OSTate] [<record_num>,]<state_num><NL>
```

<record_num> An integer from 0 to +10,484,736

<state_num> An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":SWAVeform:OSTate?"
```

RANGe

Command

```
:SWAVeform:RANGe <state_value>
```

The RANGe command specifies 10 times the number of states per division in the state waveform menu. There are 10 divisions in the state waveform display.

An error is returned if the analyzer is not in state mode.

`<state_value>` An integer whose range is from 10 to the maximum memory depth as defined by the MLENgth command in the TRIGger subsystem.

Example

```
OUTPUT XXX;":SWAVeform:RANGe 100"
```

Query

```
:SWAVeform:RANGe?
```

The RANGe query returns 10 times the number of states per division.

Returned Format

```
[:SWAVeform:RANGe] <time_value><NL>
```

`<time_range>` An integer whose range is from 1 to the maximum memory depth as defined by the MLENgth command in the TRIGger subsystem.

Example

```
OUTPUT XXX;":SWAVeform:RANGe?"
```

REMove

Command

:SWAVeform:REMove

The REMove command deletes all waveforms from the display.

Example

```
OUTPUT XXX;":SWAVeform:REMove"
```

RUNTil

Command

```
:SWAVeform:RUNTil <run_until_spec>
```

The RUNTil (run until) command defines stop criteria based on compare when the trigger mode is in repetitive. When OFF is selected, the analyzer will run until either the "STOP" touch screen field is touched or the STOP command is sent. When compare mode is enabled, there are two conditions that are based on a comparison of the acquired state data and the compare data image. The analyzer will run until one of the following conditions is true:

- Every channel of every label has the same value (EQUal).
- Any channel of any label has a different value (NEQual).

An error is returned if compare mode is disabled.

`<run_until_spec>` `{OFF|EQUal|NEQual}`

Examples

```
OUTPUT XXX;":SWAVeform:RUNTil EQUal"
```

Query

```
:SWAVeform:RUNTil?
```

The RUNTil query returns the current stop criteria.

Returned Format

```
[:SWAVeform:RUNTil] <run_until_spec><NL>
```

`<run_until_spec>` `{OFF|EQUal|NEQual}`

Example

```
OUTPUT XXX;":SWAVeform:RUNTil?"
```

SEARch

Command

```
:SWAVeform:SEARch <search_opt>
```

The SEARch command starts the search for the specified marker. The markers are placed according to the current search criteria as previously defined by the OPATtern, OSEarch, XPATtern, and XSEarch commands.

An error is returned if the analyzer is not in state mode with pattern markers on.

`<search_opt>` `{X|O|XO}`

Example

```
OUTPUT XXX;":SWAVeform:SEARch X"
```

XPATtern

Command

```
:SWAVeform:XPATtern <label_name>,<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is used with the XSEarch criteria when moving the marker. Since this command deals with a single label at a time, a complete specification could require several invocations. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is generated.

Because the maximum number of bits allowed in a label is 32, the pattern value must be in the range of 0 to 2^{32} - 1 inclusive. The label_pattern parameter is handled as a string of characters rather than a number because it may contain don't cares.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX; ":SWAVeform:XPATtern 'A','511'"
```

Query

```
:SWAVeform:XPATtern? <label_name>
```

The XPATtern query returns the pattern specification for a given label name. The analyzer must be in pattern marker mode, as set by the MMODe command, before using this command or else an error is reported.

Returned Format

```
[:SWAVeform:XPATtern] <label_name>,<label_pattern><NL>
```

<label_name> A string of up to 6 alphanumeric characters

<label_pattern>

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX;":SWAVeform:XPATtern? 'A'"
```

XSEarch

Command

```
:SWAVeform:XSEarch <occurrence>,<origin>
```

The XSEarch command defines the search criteria for the X marker. The XSEarch command tells the analyzer to search for the pattern as specified by the XPATtern command and place the X marker on this patterns. The origin parameter tells the analyzer to begin the search from the trigger, from the start of data, or from the current marker position.

The actual number of patterns searched by the analyzer is determined by the occurrence parameter. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command cause the marker search process to begin. An error is returned if the analyzer is not in pattern marker mode.

`<origin>` `{TRIGger|STARt|CURRent}`

`<occurrence>` An integer from -10484736 to +10484736

Example

```
OUTPUT XXX; ":SWAVeform:XSEarch,+10,TRIGGER"
```

Query

```
:SWAVeform:XSEarch?
```

The XSEarch query returns the search criteria for the X marker.

Returned Format `[:SWAVeform:XSEarch] <occurrence>,<origin><NL>`

`<origin>` `{TRIGger|STARt|CURRent}`

`<occurrence>` An integer from -10484736 to +10484736

Example

```
OUTPUT XXX;":SWAVeform:XSEarch?"
```

XSTate

Query

```
:SWAVeform:XSTate?
```

The XSTate query returns the line and record number in the listing where the X marker resides (-10,484,736 to +10,484,736). If the specified number of occurences of the pattern or the specified pattern is not in the listing, the number returned is 2,147,483,647 which indicates and error condition.

Returned Format `[:SWAVeform:XSTate] [<rec_num>,]<state_num><NL>`

`<rec_num>` An integer from 0 to +10,484,736

`<state_num>` An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":SWAVeform:XSTate?"
```

8

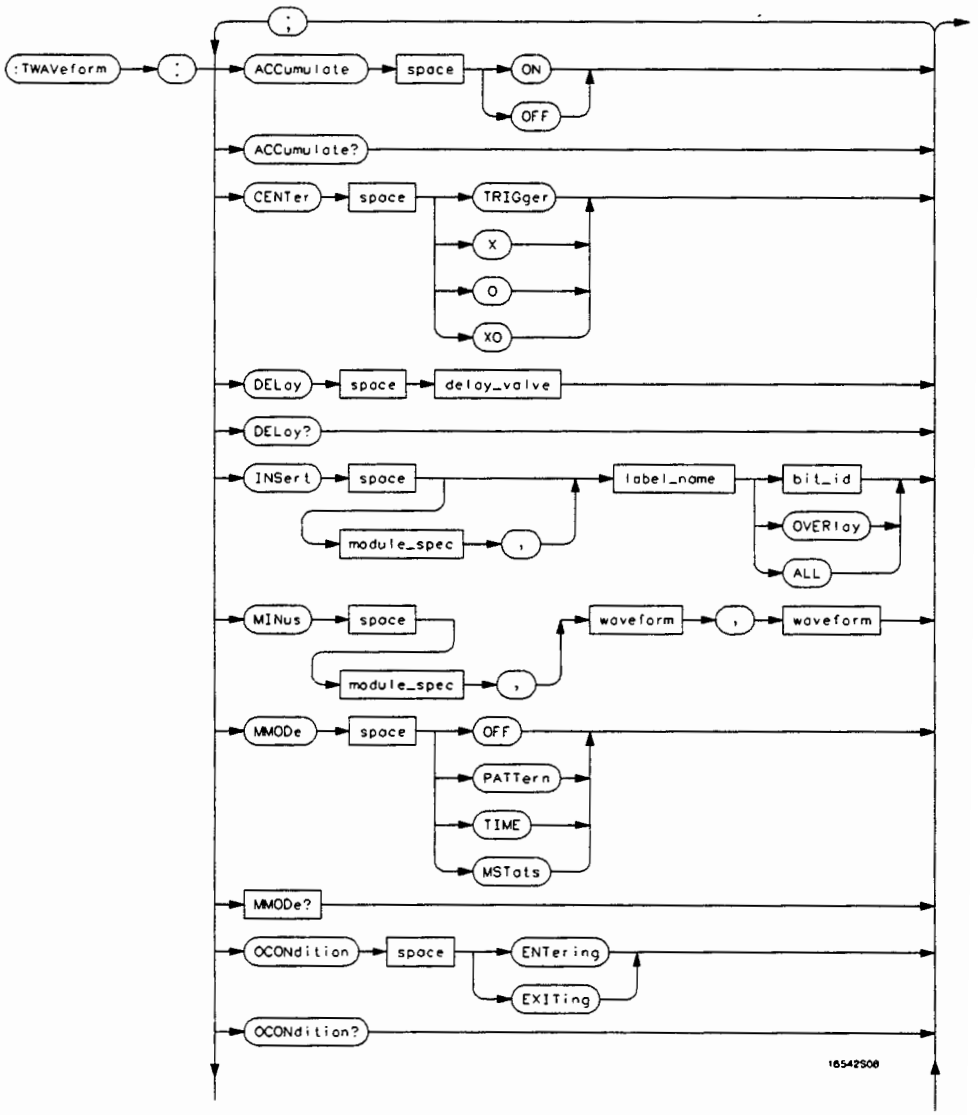
TWAVeform Subsystem

Introduction

The TWAVeform subsystem contains the commands available for the Timing Waveforms menu in the HP 16542A. These commands are listed below:

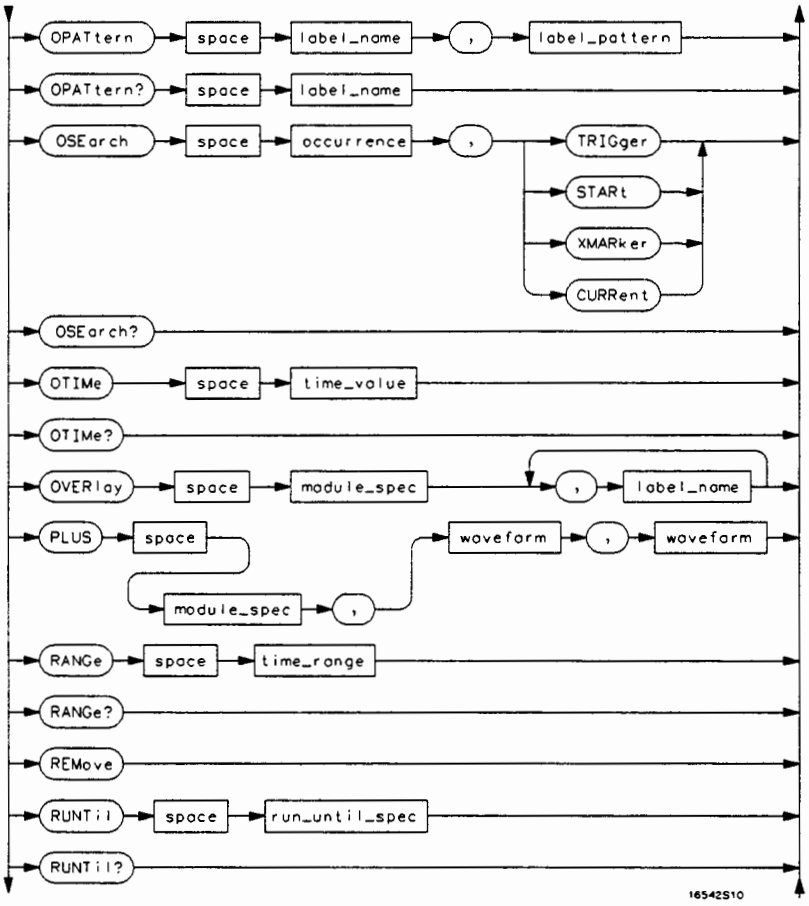
- ACCumulate
- CENTer
- DELay
- INSert
- MINus
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OTIMe
- OVERlay
- PLUS
- RANGe
- REMove
- RUNTil
- SEARch
- SPERiod
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XCONdition
- XOTime
- XPATtern
- XSEarch
- XTIMe

Figure 8-1



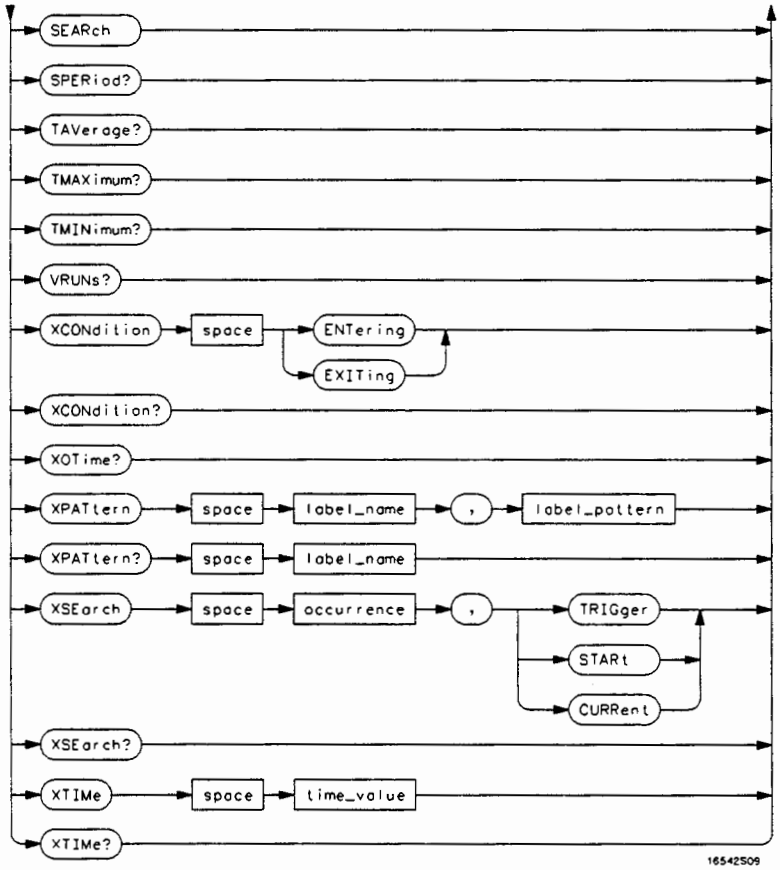
TWAVeform Subsystem Syntax Diagram

Figure 8-1



TWAVeform Subsystem Syntax Diagram (cont.)

Figure 8-1



TWAVeform Subsystem Syntax Diagram (cont.)

Table 8-1 **TWAVeform Parameter Values**

Parameter	Value
delay_value	real number between ±105ms
module_spec	integer from 1 to 10
bit_id	integer from 0 to 31
waveform	string containing <acquisition_spec>{1|2}
acquisition_spec	{A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)
label_name	string of up to 6 alphanumeric characters
label_pattern	Any one of the following: {#B{0|1|X}... #Q{0|1|2|3|4|5|6|7|X}... #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} ... {0|1|2|3|4|5|6|7|8|9}...}
occurrence	integer between ±10,484,736
time_value	real number between ±105ms
time_range	real number between 100 ns and 10 ks
run_until_spec	{LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual|OFF}
value	real number

Table 8-2 **Power On Defaults**

Command	Default
ACCumulate	OFF
DELay	0
MMODe	OFF
RANGe	10e-8 (100 ns/div)

TWAVeform

Selector

```
:TWAVeform
```

The TWAVeform selector is used as part of a compound header to access the settings found in the Timing Waveforms menu.

Example

```
OUTPUT XXX;":TWAVeform:DELay 100E-9"
```

ACCumulate

Command

```
:TWAVeform:ACCumulate <setting>
```

The ACCumulate command allows you to control whether the timing waveform display gets erased between each individual run or whether subsequent waveforms are allowed to be displayed over the previous ones.

`<setting>` `{0|OFF} or {1|ON}`

Example

```
OUTPUT XXX;":TWAVeform:ACCumulate ON"
```

Query

```
:TWAVeform:ACCumulate?
```

The ACCumulate query returns the current setting. The query always shows the setting as the number 0 (off) or 1 (on).

Returned Format

```
[:TWAVeform:ACCumulate] {0|1}<NL>
```

Example

```
OUTPUT XXX;":TWAVeform:ACCumulate?"
```

CENTer

Command

```
:TWAVeform:CENTer {X|O|XO|TRIGger}
```

The CENTer command allows you to center the waveform display about the specified markers. The markers are placed on the waveform using other commands (XSEarch, OSEarch, XTIMe, OTIMe, etc.).

An error is returned unless the timing waveform menu is selected and markers are turned on.

Example

```
OUTPUT XXX;":TWAVeform:CENTer X"
```

DELay

Command

```
:TWAVeform:DELay <delay_value>
```

The DELay command specifies the amount of time between the timing trigger and the horizontal center of the the timing waveform display. The allowable values for delay are -105ms to +105ms.

An error is returned if the analyzer is not in timing mode.

`<delay_value>` A real number between ±105ms

Example

```
OUTPUT XXX;":TWAVeform:DELay 100E-6"
```

Query

```
:TWAVeform:DELay?
```

The DELay query returns the current time offset (delay) value from the trigger.

Returned Format

```
[:TWAVeform:DELay] <delay_value><NL>
```

`<delay_value>` A real number between -2500 s and +2500 s

Example

```
OUTPUT XXX;":TWAVeform:DELay?"
```

INSert

Command

```
:TWAVeform:INSert [<module_spec>,]<label_name>
[,{<bit_id>|OVERlay|ALL}]
```

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom. When 96 waveforms are present, inserting additional waveforms replaces the last waveform.

Time-correlated waveforms from the oscilloscope and timing analyzer waveforms can also be inserted in the logic analyzer's timing waveforms display. Oscilloscope waveforms occupy the same display space as three logic analyzer waveforms. When inserting waveforms from another module, the optional first parameter must be used. 1...10 corresponds to modules A...J. If the module specifier is not used, the selected module is assumed.

The second parameter specifies the label name that will be inserted. The optional third parameter specifies the label bit number, overlay, or all. If a number is specified, only the waveform for that bit number is added to the screen.

If OVERlay is specified, all the bits of the label are displayed as a composite overlaid waveform. If ALL is specified, all the bits are displayed sequentially. If the third parameter is not specified, ALL is assumed.

`<module_spec>` An integer from 1 to 10

`<label_name>` A string of up to 6 alphanumeric characters

`<bit_id>` An integer from 0 to 31

Example

```
OUTPUT XXX;":TWAVeform:INSert 3, 'WAVE',10"
```

MINus

Command

```
:TWAVeform:MINus <module_spec>,<waveform>,
<waveform>
```

The MINus command inserts time-correlated A-B (A minus B) oscilloscope waveforms on the screen. The first parameter is the module specifier where the oscilloscope module resides with 1 through 10 representing slots A through J. The next two parameters specify which waveforms are subtracted from each other.

An error is returned if the logic analyzer is not in timing mode

> The MINus command is only valid for oscilloscope waveforms

`<module_spec>` An integer from 1 to 10

`<waveform>` `{A|B|C|D|E|F|G|H|I|J}{1|2}`

Example

```
OUTPUT XXX;":TWAVeform:MINus 2,'A1','A2'"
```

MMODe

Command

```
:TWAVeform:MMODe {OFF|PATTern|TIME|MSTats}
```

The MMODe command selects the marker mode which controls the movement of the X and O markers and the marker readouts. When PATTern is selected, the markers are placed on the patterns specified by the XPATtern and OPATtern commands.

Selecting TIME, places the markers on time values with respect to the trigger, as specified by the XTIMe and OTIMe commands. In MSTats mode, markers are placed on patterns but the readouts display the number of valid runs, the minimum X to O marker time, the maximum X to O marker time,

and the average X to O marker time. An error is returned if the analyzer is not in timing mode.

Example

```
OUTPUT XXX; ":TWAVeform:MMODe TIME"
```

Query

```
:TWAVeform:MMODe?
```

The MMODe query returns the current marker mode.

Returned Format

```
[:TWAVeform:MMODe] <marker_mode><NL>
```

<marker_mode>

```
{OFF|PATTern|TIME|MSTats}
```

Example

```
OUTPUT XXX;":TWAVeform:MMODe?"
```

OCONdition

Command

```
:TWAVeform:OCONdition {ENTering|EXITing}
```

The OCONdition command specifies where the O marker is placed. The O marker can be placed on the entry or exit point of the OPATtern when in the PATTern marker mode.

An error is returned if the analyzer is not in timing mode with pattern markers on.

Example

```
OUTPUT XXX; ":TWAVeform:OCONdition ENTERING"
```

Query

```
:TWAVeform:OCONdition?
```

The OCONdition query returns the current setting.

Returned Format

```
[:TWAVeform:OCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":TWAVeform:OCONdition?"
```

OPATtern

Command

```
:TWAVeform:OPATtern <label_name>,<label_pattern>
```

The OPATtern command allows you to construct a pattern recognizer term for the O marker which is then used with the OSEarch criteria and OCONdition when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and 2^{32} - 1, since a label may not have more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

An error is returned if the analyzer is is not in pattern marker mode.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX; ":TWAVeform:OPATtern 'A','511'"
```

Query

```
:TWAVeform:OPATtern? <label_name>
```

The OPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the O marker for a given label. If the O marker is not placed on valid data, don't cares (XX...X) are returned.

Returned Format

```
[:TWAVeform:OPATtern] <label_name>,<label_pattern><NL>
```

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX;":TWAVeform:OPATtern? 'A'"
```

OSEarch

Command

```
:TWAVeform:OSEarch <occurrence>,<origin>
```

The OSEarch command defines the search criteria for the O marker which is used with the associated OPATtern recognizer specification and the OCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search from the trigger, from the start, from the X marker or from the current marker position. The actual occurrence the marker searches for is determined by the occurrence parameter relative to the origin. An occurrence of 0 places the marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin. An error is returned if the analyzer is not in pattern marker mode.

`<origin>` `{TRIGger|STARt|XMARker|CURRent}`

`<occurrence>` An integer in the range of plus or minus the maximum system memory depth.

Example

```
OUTPUT XXX; ":TWAVeform:OSEarch +10,TRIGGER"
```

Query `:TWAVeform:OSEarch?`

The OSEarch query returns the search criteria for the O marker.

Returned Format `[:TWAVeform:OSEarch] <occurrence>,<origin><NL>`

`<origin>` `{TRIGger|STARt|XMARker|CURRent}`

`<occurrence>` An integer in the range of plus or minus the maximum system memory depth.

Example

```
OUTPUT XXX;":TWAVeform:OSEarch?"
```

OTIMe

Command `:TWAVeform:OTIMe <time_value>`

The OTIMe command positions the O marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

An error is returned if the analyzer is not in timing mode with time markers on.

`<time_value>` A real number between ±105ms

Example

```
OUTPUT XXX; ":TWAVeform:OTIME 30.0E-6"
```

Query `:TWAVeform:OTIMe?`

The OTIMe query returns the O marker position in time. If data is not valid, the query returns 9.9E37.

Returned Format `[:TWAVeform:OTIMe] <time_value><NL>`

`<time_value>` A real number between ±105ms

Example

```
OUTPUT XXX;":TWAVeform:OTIMe?"
```

OVERlay

Command

```
:TWAVeform:OVERlay <module_spec>,<label>
[,<label>]...
```

The OVERlay command overlays two or more oscilloscope waveforms and adds the resultant waveform to the current waveforms display. The first parameter of the command specifies where the oscilloscope module resides with 1 through 10 representing slots A through J. The next parameters are the labels of the waveforms that being overlaid.

An error is returned if the analyzer is not in timing mode.

> The OVERlay command is only valid for oscilloscope waveforms

`<module_spec>` An integer from 1 to 10

`<waveform>` `{A|B|C|D|E|F|G|H|I|J}{1|2}`

Example

```
OUTPUT XXX;":TWAVeform:OVERlay 4,'C1','C2'"
```

PLUS

Command

```
:TWAVeform:PLUS <module_spec>,<waveform>,
<waveform>
```

The PLUS command inserts time-correlated A+B (A plus B) oscilloscope waveforms on the screen. The first parameter is the module specifier where the oscilloscope module resides with 1 through 10 representing slots A through J. The next two parameters specify which waveforms are subtracted from each other.

An error is returned if the logic analyzer is not in timing mode

> The PLUS command is only valid for oscilloscope waveforms

`<module_spec>` An integer from 1 to 10

`<waveform>` `{A|B|C|D|E|F|G|H|I|J}{1|2}`

Example

```
OUTPUT XXX;":TWAVeform:MINus 2,'A1','A2'"
```

RANGe

Command

```
:TWAVeform:RANGe <time_value>
```

The RANGe command specifies the full-screen time in the timing waveform menu. It is equivalent to ten times the seconds-per-division setting on the display. The allowable values for RANGe are from 100ns to 10Ks.

`<time_value>` A real number between 100ns and 10Ks

Example `OUTPUT XXX;":TWAVeform:RANGe 100E-9"`

Query `:TWAVeform:RANGe?`

The RANGe query returns the current full-screen time.

Returned Format `[:TWAVeform:RANGe] <time_value><NL>`

`<time_range>` A real number between 100 us and 1 Ks

Example `OUTPUT XXX;":TWAVeform:RANGe?"`

REMove

Command `:TWAVeform:REMove`

The REMove command deletes all waveforms from the display.

Example `OUTPUT XXX;":TWAVeform:REMove"`

RUNTil

Command

```
:TWAVeform:RUNTil <run_until_spec>
```

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers when the trigger mode is in repetitive. When OFF is selected, the analyzer will run until either the "STOP" touch screen field is touched or the STOP command is sent. Run until the time between X and O marker options are:

- Less Than (LT) a specified time value
- Greater Than (GT) a specified time value
- In the range (INRange) between two time values
- Out of the range (OUTRange) between two time values

End points for the INRange and OUTRange should be at least 10 ns apart since this is the minimum time at which data is sampled.

There are two conditions that are based on a comparison of the acquired state data and the compare data image. The analyzer will run until one of the following conditions is true:

- Every channel of every label has the same value (EQUal).
- Any channel of any label has a different value (NEQual).

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange<value>,<value>|OUTRange<value>,<value>|EQUal|NEQual}`

`<value>` A real number

Examples

```
OUTPUT XXX;":TWAVeform:RUNTil GT, 800.0E-6"
OUTPUT XXX;":TWAVeform:RUNTil INRANGe, 4.5, 5.5"
```

Query

```
:TWAVeform:RUNTil?
```

The RUNTil query returns the current stop criteria.

Returned Format

```
[:TWAVeform:RUNTil] <run_until_spec> <value><NL>
```

`<run_until_spec>` `{OFF|LT,<value>|GT,<value>|INRange<value>,<value>|OUTRange<value>,<value>|EQUal|NEQual}`

`<value>` A real number

Example

```
OUTPUT XXX;":TWAVeform:RUNTil?"
```

SEARch

Command

```
:TWAVeform:SEARch
```

The SEARch command starts the search for both the X and O markers. The markers are placed according to the current search criteria as previously defined by the the OPATtern, OSEarch, XPATtern, and XSEarch commands.

Example

```
OUTPUT XXX;":TWAVeform:SEARch"
```

SPERiod

Query

```
:TWAVeform:SPERiod?
```

The SPERiod query always returns 10 us as the sample period.

Returned Format

```
[:TWAVeform:SPERiod] 10<NL>
```

Example

```
OUTPUT XXX;":TWAVeform:SPERiod?"
```

TAVerage

Query

```
:TWAVeform:TAVerage?
```

The TAVerage query returns the value of the average time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Returned Format

```
[:TWAVeform:TAVerage] <time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":TWAVeform:TAVerage?"
```

TMAXimum

Query

```
:TWAVeform:TMAXimum?
```

The TMAXimum query returns the value of the maximum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Returned Format

```
[:TWAVeform:TMAXimum]<time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":TWAVeform:TMAXimum?"
```

TMINimum

Query

```
:TWAVeform:TMINimum?
```

The TMINimum query returns the value of the minimum time between the X and O markers. If there is no valid data, the query returns 9.9E37.

Returned Format `[:TWAVeform:TMINimum] <time_value><NL>`

`<time_value>` A real number

Example

```
OUTPUT XXX;":TWAVeform:TMINimum?"
```

VRUNs

Query `:TWAVeform:VRUNs?`

The VRUNs query returns the number of valid runs and total number of runs made. Valid runs are those where the pattern search for both the X and O markers was successful resulting in valid delta time measurements.

Returned Format `[:TWAVeform:VRUNs] <valid_runs>,<total_runs><NL>`

`<valid_runs>` zero or positive An integer

`<total_runs>` zero or positive An integer

Example

```
OUTPUT XXX;":TWAVeform:VRUNs?"
```

XCONdition

Command `:TWAVeform:XCONdition {ENTering|EXITing}`

The XCONdition command specifies where the X marker is placed. The X marker can be placed on the entry or exit point of the XPATtern when in the PATTern marker mode.

Example

```
OUTPUT XXX; ":TWAVeform:XCONdition ENTERING"
```

Query

```
:TWAVeform:XCONdition?
```

The XCONdition query returns the current setting.

Returned Format

```
[:TWAVeform:XCONdition] {ENTering|EXITing}<NL>
```

Example

```
OUTPUT XXX;":TWAVeform:XCONdition?"
```

XOTime

Query

```
:TWAVeform:XOTime?
```

The XOTime query returns the time from the X marker to the O marker. If data is not valid, the query returns 9.9E37.

Returned Format

```
[:TWAVeform:XOTime] <time_value><NL>
```

`<time_value>` A real number

Example

```
OUTPUT XXX;":TWAVeform:XOTIMe?"
```

XPATtern

Command

```
:TWAVeform:XPATtern <label_name>,<label_pattern>
```

The XPATtern command allows you to construct a pattern recognizer term for the X marker which is then used with the XSEarch criteria and XCONdition when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several invocations.

When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. In whatever base is used, the value must be between 0 and 2^{32} - 1, since a label may not have

more than 32 bits. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX; ":TWAVeform:XPATtern 'A','511'"
```

Query

```
:TWAVeform:XPATtern? <label_name>
```

The XPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the X marker for a given label. If the X marker is not placed on valid data, don't cares (XX...X) are returned.

Returned Format

```
[:TWAVeform:XPATtern] <label_name>,<label_pattern><NL>
```

`<label_name>` A string of up to 6 alphanumeric characters

`<label_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
OUTPUT XXX;":TWAVeform:XPATtern? 'A'"
```

XSEarch

Command

```
:TWAVeform:XSEarch <occurrence>,<origin>
```

The XSEarch command defines the search criteria for the X marker which is then used with the associated XPATtern recognizer specification and the XCONdition when moving markers on patterns. The origin parameter tells the marker to begin a search from the trigger, from the start, or from the current marker position. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0, places a marker on the selected origin. With a negative occurrence, the marker searches before the origin. With a positive occurrence, the marker searches after the origin.

Issuing this command causes the marker search process to begin.

`<origin>` `{TRIGger|STARt|CURRent}`

`<occurrence>` An integer from -10484737 to +10484737

Example

```
OUTPUT XXX; ":TWAVeform:XSEarch,+10,TRIGGER"
```

Query

```
:TWAVeform:XSEarch?
```

The XSEarch query returns the search criteria for the X marker.

Returned Format

```
[:TWAVeform:XSEarch] <occurrence>,<origin><NL>
```

`<origin>` `{TRIGger|STARt|CURRent}`

`<occurrence>` An integer from -10484737 to +10484737

Example

```
OUTPUT XXX;":TWAVeform:XSEarch?"
```

XTIMe

Command

```
:TWAVeform:XTIMe <time_value>
```

The XTIMe command positions the X marker in time when the marker mode is TIME. If data is not valid, the command performs no action.

`<time_value>` A real number from -2.5Ks to +2.5Ks

Example

```
OUTPUT XXX; ":TWAVeform:XTIMe 40.0E-6"
```

Query

```
:TWAVeform:XTIMe?
```

The XTIMe query returns the X marker position in time. If data is not valid, the query returns 9.9E37.

Returned Format

```
[:TWAVeform:XTIMe] <time_value><NL>
```

`<time_value>` A real number from -2.5Ks to +2.5Ks

Example

```
OUTPUT XXX;":TWAVeform:XTIMe?"
```

9

CHARt Subsystem

Introduction

The CHARt subsystem provides the commands necessary for programming the HP 16542A to build charts of label activity, using data normally found in the Listing display. The chart's Y-axis is used to show data values for the label of your choice. The X-axis can be used in two different ways. In one, the X-axis represents states (shown as rows in the Listing display). In the other, the X-axis represents the data values for another label. When states are plotted along the X-axis, X and O markers are available. Since the Chart display is simply an alternative way of looking at the data in the Listing, the X and O markers can be manipulated through the LIST subsystem. Because the programming commands do not force the menus to switch, you can position the markers in the LIST subsystem and see the effects in the Chart display.

The commands in the CHARt subsystem are listed below:

- ACCumulate
- CENTer
- HAXis
- VAXis

Figure 9-1

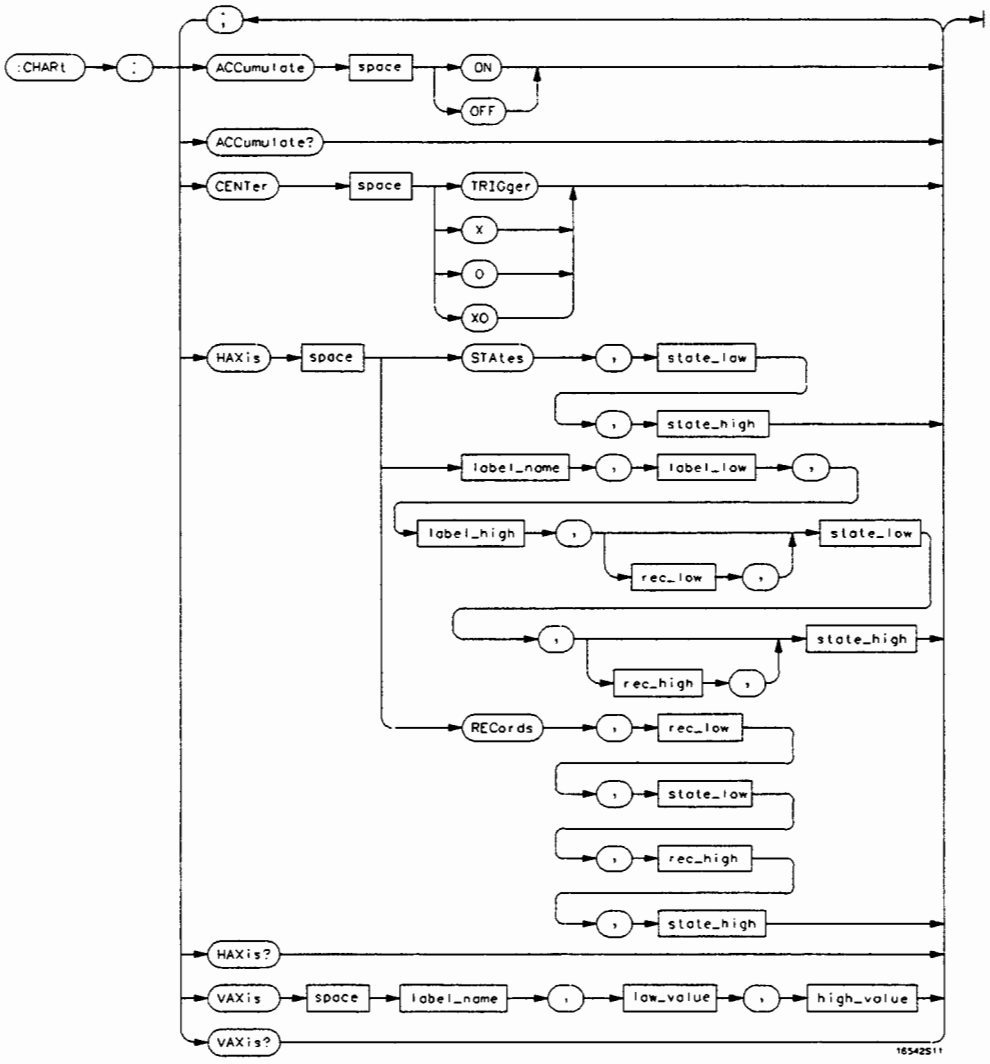


CHARt Subsystem Syntax Diagram

Table 9-1 **CHARt Parameter Values**

Parameter	Value
state_low	integer from -10,484,736 to +10,484,736
state_high	integer from <state_low_value> to +10,484,736
label_name	a string of up to 6 alphanumeric characters
label_low	string from 0 to 2^{32} - 1 (#HFFFF)
label_high	string from <label_low_value> to 2^{32} - 1 (#HFFFF)
low_value	string from 0 to 2^{32} - 1 (#HFFFF)
high_value	string from <label_low_value> to 2^{32} - 1 (#HFFFF)
rec_low	integer from -10,484,736 to +10,484,736
rec_high	integer from -10,484,736 to +10,484,736

Table 9-2 **Power On Defaults**

Command	Default
ACCumulate	OFF
HAXis	STAtes,-20,500
VAXis	#H0000,#HFFFF

CHARt

Selector

```
:CHARt
```

The CHARt selector is used as part of a compound header to access the settings found in the State Chart menu.

Example

```
OUTPUT XXX;":CHARt:VAXis 'A', '0', '9'"
```

ACCumulate

Command

```
:CHARt:ACCumulate {{ON|1} | {OFF|0}}
```

The ACCumulate command allows you to control whether the chart display gets erased between each individual run or whether subsequent waveforms are allowed to be displayed over the previous waveforms.

Example

```
OUTPUT XXX;":CHARt:ACCumulate OFF"
```

Query

```
CHARt:ACCumulate?
```

The ACCumulate query returns the current setting. The query always shows the setting as the number "0" (off) or "1" (on).

Returned Format

```
[CHARt:ACCumulate] {0 | 1}<NL>
```

Example

```
OUTPUT XXX;":CHARt:ACCumulate?"
```

CENTer

Command

```
:CHARt:CENTer <center_arg>
```

The CENTer command redraws the waveforms centering them on the screen as specified by the center argument. If the specified chart menu is not on screen, an error is returned.

`<center_arg>` `{TRIGger|X|O|XO}`

Example

```
OUTPUT XXX;":CHARt:CENTer XO"
```

HAXis

Command (for States)

```
CHARt:HAXis {STAtes,<state_low>,<state_high>}
```

Command (for Label)

```
CHARt:HAXis {<label_name>,<label_low>,
<label_high>,[<rec_low>,]<state_low>,
[<rec_high>,]<state_high>}
```

Command (for Records)

```
CHARt:HAXis RECords,<rec_low>,<state_low>,
<rec_high>,<state_high>}
```

The HAXis command allows you to select whether states, records, or a label's values will be plotted on the horizontal axis of the chart. The axis is scaled by specifying the high and low values.

> If the label version of the command is used, there must be four numbers following the label range in records on mode and two numbers following the label range in records off mode, otherwise an error is returned.

`<state_low>` An integer from -10,485,760 to 10,484,736

`<state_high>` An integer from -10,485,760 to 10,484,736

`<label_name>` A string of up to 6 alphanumeric characters

`<label_low>` A string from 0 to 2^{32}-1 (#HFFFF)

`<label_high>` A string from <label_low_value> to 2^{32}-1 (#HFFFF)

`<rec_low>` An integer from -10,485,760 to 10,484,736

`<rec_high>` An integer from -10,485,760 to 10,484,736

Examples

```
OUTPUT XXX;":CHARt:HAXis STATES,-100,100"
OUTPUT XXX;":CHARt:HAXis 'NAUJ','-511','511',20,100000"
```

Query

```
:CHARt:HAXis?
```

The HAXis query returns the current horizontal axis label and scaling.

Returned Format (for States)

```
[CHARt:HAXis] STAtes,<state_low>,<state_high><NL>
```

Returned Format (for Label)

```
[CHARt:HAXis] label_name>,<label_low>,<label_high>,
[<rec_low>,]<state_low>,[<rec_high>,]<state_high><NL>
```

Returned Format (for Records)

```
[CHARt:HAXis] RECords,<label_low>,<label_high>,
[<rec_low>,]<state_low>,[<rec_high>,]<state_high><NL>
```

`<state_low>` An integer from -10,484,736 to 10,484,736

`<state_high>` An integer from -10,484,736 to 10,484,736

`<label_name>` A string of up to 6 alphanumeric characters

`<label_low>` A string from 0 to 2^{32}-1 (#HFFFF)

`<label_high>` A string from <label_low_value> to 2^{32}-1 (#HFFFF)

`<rec_low>` An integer from -10,485,760 to 10,484,736

`<rec_high>` An integer from -10,485,760 to 10,484,736

Example

```
OUTPUT XXX;":CHARt:HAXis?"
```

VAXis

Command `CHARt:VAXis <label_name>,<low_value>,<high_value>`

The VAXis command allows you to choose which label will be plotted on the vertical axis of the chart and to scale the vertical axis by specifying the high value and low value.

`<label_name>` A string of up to 6 alphanumeric characters

`<low_value>` A string from 0 to 2^{32}-1 (#HFFFF)

`<high_value>` A string from <low_value> to 2^{32}-1 (#HFFFF)

Examples

```
OUTPUT XXX;":CHARt:VAXis 'SUM1', '0', '99'"
OUTPUT XXX;":CHARt:VAXis 'BUS', '#H00FF', '#H0500'"
```

Query `:CHARt:VAXis?`

The VAXis query returns the current vertical axis label and scaling.

Returned Format `[CHARt:VAXis] <label_name>,<low_value>,<high_value><NL>`

`<label_name>` A string of up to 6 alphanumeric characters

`<low_value>` A string from 0 to 2^{32}-1 (#HFFFF)

`<high_value>` A string from <low_value> to 2^{32}-1 (#HFFFF)

Example

```
OUTPUT XXX;":CHARt:VAXis?"
```

10

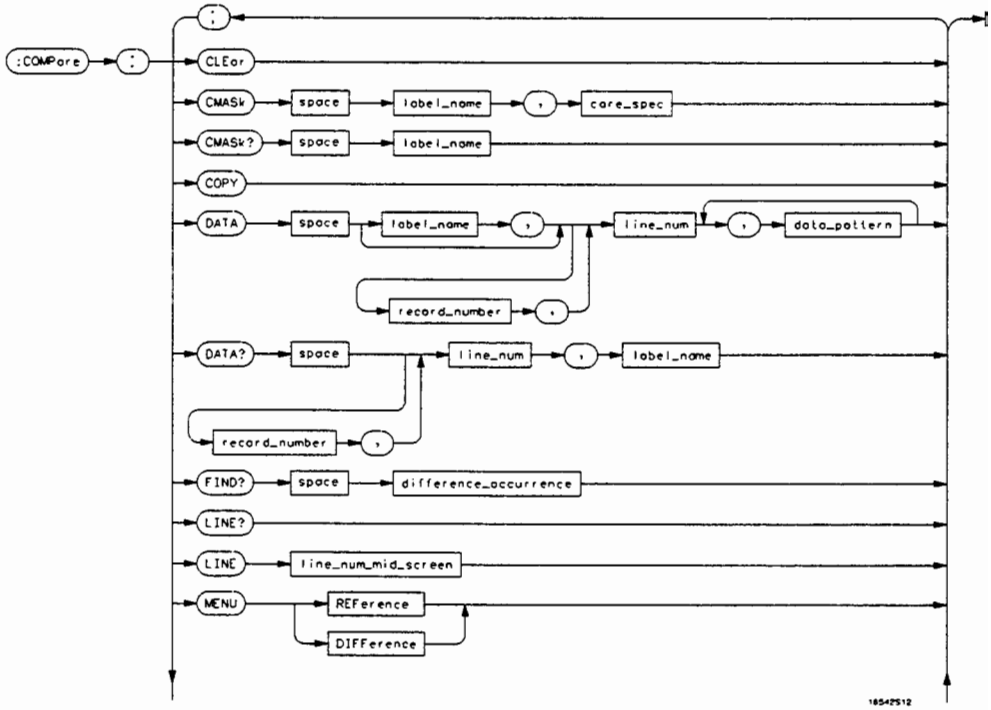
COMPare Subsystem

Introduction

Commands in the state COMPare subsystem provide the ability to do a bit-by-bit comparison between the acquired data and a compare data image. The commands are listed below:

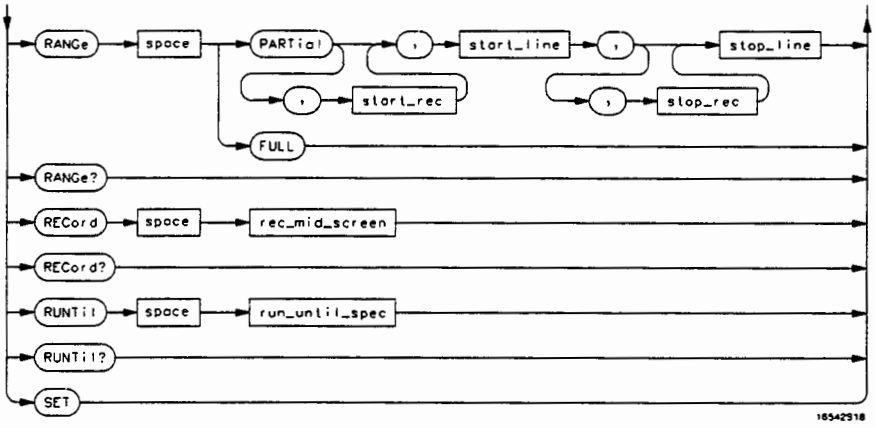
- CLEar
- CMASk
- COPY
- DATA
- FIND
- LINE
- MENU
- RANGe
- RECord
- RUNTil
- SET

Figure 10-1



COMPare Subsystem Syntax Diagram

Figure 10-1



COMPare Subsystem Syntax Diagram (cont.)

Table 10-1 **COMPare Parameter Values**

Parameter	Value
label_name	string of up to 6 characters
care_spec	string of characters "{*|.}..." * = care . = don't care
line_mid_screen	integer from -10,484,736 to +10,484,736
line_num	integer from -10,484,736 to +10,484,736

data_pattern	Any one of the following:
	`{#B{0|1|X}...`
	`#Q{0|1|2|3|4|5|6|7|X}...`
	`#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...`
	`{0|1|2|3|4|5|6|7|8|9}...}`

difference_occurence	integer from 1 to 10,484,736
start_line	integer from -10,484,736 to +10,484,736
stop_line	integer from <start_line> to +10,484,736
record_number	integer from 0 to +10,484,736
run_until_spec	{`OFF|LT,<value>|GT,<value>|INRange,<value>,<value>|OUTRange,<value>,<value>|EQual|NEQual`}
rec_mid_screen	integer from 0 to +10,484,736
start_rec	integer from 0 to +10,484,736
stop_rec	integer from <start_rec> to +10,484,736
value	real number

COMPare

Selector

`:COMPare`

The COMPare selector is used as part of a compound header to access the settings found in the Compare menu.

Example

```
OUTPUT XXX;":COMPare:FIND? 819"
```

CLEar

Command

```
:COMPare:CLEar
```

The CLEar command clears all "don't cares" in the reference listing and replaces them with zeros except when the CLEar command immediately follows the SET command.

Example

```
OUTPUT XXX;":COMPare:CLEar"
```

CMASk

Command

```
:COMPare:CMASk <label_name>,<care__spec>
```

The CMASk (Compare Mask) command allows you to set the bits in the channel mask for a given label in the compare listing image to "compares" or "don't compares."

`<label_name>` A string of up to 6 alphanumeric characters

`<care_spec>` A string of characters "{*|.}..." (32 characters maximum)

* = care
= don't care

Example

```
OUTPUT XXX;":COMPare:CMASk 'EKIM', '*.**..**'"
```

Query

```
:COMPare:CMASk? <label_name>
```

The CMASk query returns the state of the bits in the channel mask for a given label in the compare listing image.

Returned Format

```
[COMPare:CMASk] <label_name>,<care_spec><NL>
```

`<label_name>` A string of up to 6 alphanumeric characters

`<care_spec>` A string of characters "{*|.}..." (32 characters maximum)

* = care

. = don't care

Example

```
OUTPUT XXX;":COMPare:CMASk? 'POD5'"
```

COPY

Command

```
:COMPare:COPY
```

The COPY command copies the current acquired State Listing into the Compare Listing template. It does not affect the compare range or channel mask settings.

Example

```
OUTPUT XXX;":COMPare:COPY"
```

DATA

Command

```
:COMPare:DATA {[<label_name >,][<record_num>,]
<line_num>,<data_pattern>[,<data_pattern>]... }
```

The DATA command allows you to edit the compare listing image for a given label and state row. When DATA is sent to an instrument where no compare image is defined (such as at power-up) all other data in the image is set to don't cares.

If the label_name parameter is not specified, the left most label is assumed.

Because don't cares (Xs) are allowed in the data pattern, it must always be expressed as a string. You may still use different bases; though don't cares cannot be used in a decimal number.

`<label_name>` A string of up 6 alphanumeric characters

`<line_num>` An integer from –10,484,736 to +10,484,736

`<record_num>` An integer from 0 to 10,485,760

`<data_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Examples

```
OUTPUT XXX;":COMPare:DATA 'CLOCK', 42, '#B011X101X'"
OUTPUT XXX;":COMPare:DATA 'OUT3', 0, '#HFF40'"
OUTPUT XXX;":COMPare:DATA 129,'#BXX00','#B1101','#B10XX'"
OUTPUT XXX;":COMPare:DATA -511, '4', '64', '16', '256', '8',
'16'"
```

Query

`:COMPare:DATA? [<record_num>,]<line_num>, <label_name>`

The DATA query returns the value of the compare listing image for a given label and state line number. If the analyzer is in record mode, the record number is also returned.

Returned Format

`[COMPare:DATA] [<record_num>,]<line_num>,<label_name>, <data_pattern><NL>`

`<record_num>` An integer from 0 to 10,485,760

`<label_name>` A string of up 6 alphanumeric characters

`<line_num>` An integer from -10,484,736 to +10,484,736

`<data_pattern>`

```
{#B{0|1|X} . . . |
 #Q{0|1|2|3|4|5|6|7|X} . . . |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X} . . . |
   {0|1|2|3|4|5|6|7|8|9} . . . }
```

Example

```
10 DIM Label$[6], Response$[80]
15 PRINT "This program shows the values for a signal's Compare
listing"
20 INPUT "Enter signal label: ", Label$
25 OUTPUT XXX;":SYSTem:HEADER OFF"    !Turn headers off (from
responses)
30 OUTPUT XXX;":COMPare:RANGe?"
35 ENTER XXX; First, Last       !Read in the range's end-points
40 PRINT "LINE #", "VALUE of "; Label$
45 FOR State = First TO Last    !Print compare value for each
state
50   OUTPUT XXX;":COMPare:DATA? '" & Label$ & "'," &
VAL$(State)
55   ENTER XXX; Response$
60   PRINT State, Response$
65   NEXT State
70 END
```

FIND

Query

```
:COMPare:FIND? <diff_occurrence>
```

The FIND query is used to get the line number of a specified difference occurence (for example first, second, third) within the current compare range, as dictated by the RANGe command. A difference is counted for each line where at least one of the current labels has a discrepancy between its acquired state data listing and its compare data image.

Invoking the FIND query updates both the Listing and Compare displays so that the line number returned is in the center of the screen.

If record mode is on, a record number is returned before the line number. When the difference occurence is not found then a value of 2147483647 is returned.

An error is returned if the analyzer compare memory is disabled.

Returned Format

```
[COMPare:FIND] <difference_occurrence>,[<rec_number>,]
<line_number><NL>
```

<diff_occurrence>	An integer from 1 to 10,484,736
<rec_number>	An integer from 0 to +10,484,736
<line_number>	An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":COMPare:FIND? 26"
```

LINE

Command

```
:COMPare:LINE <line_mid_screen>
```

The LINE command allows you to scroll the compare listing vertically. The command specifies the state line number relative to the trigger that the analyzer will highlight at the center of the screen.

<line_mid_screen>	An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":COMPare:LINE 0"
```

Query

```
:COMPare:LINE?
```

The LINE query returns the line number for the state currently in the box at center screen.

Returned Format

```
[:COMPare:LINE] <line_mid_screen><NL>
```

<line_mid_screen>	An integer from -10,484,736 to +10,484,736

Example

```
OUTPUT XXX;":COMPare:LINE?"
```

MENU

Command

```
:COMPare:MENU {REFerence|DIFFerence}
```

The MENU command allows you to switch between the compare listing and the compare difference listing.

Example

```
OUTPUT XXX;":COMPare:MENU DIFFerence"
```

RANGe

Command

```
:COMPare:RANGe {FULL |
PARTial,[<start_record>,]<start_line>,
[<stop_record>,]<stop_line>
```

The RANGe command allows you to define the boundaries for the comparison. The range entered must be a subset of the lines in the acquire memory.

`<start_line>` An integer from -10,484,736 to +10,484,736

`<stop_line>` An integer from <start_line> to +10,484,736

`<start_record>` An integer from 0 to maximum number of records

`<stop_record>` An integer from <start_record> to maximum number of records

Examples

```
OUTPUT XXX;":COMPare:RANGe PARTIAL, -511, 512"
OUTPUT XXX;":COMPare:RANGe FULL"
```

Query `:COMPare:RANGe?`

The RANGe query returns the current boundaries for the comparison.

If record mode is on, a start record number precedes the start line number and the stop record number precedes the stop line number.

Returned Format
```
[COMPare:RANGe] {FULL |
PARTial,[<start_record>,]<start_line>,
[<stop_record>,]<stop_line>}<NL>
```

`<start_line>` An integer from –10,484,736 to +10,484,736

`<stop_line>` An integer from <start_line> to +10,484,736

`<start_record>` An integer from 0 to maximum number of records

`<stop_record>` An integer from <start_record> to maximum number of records

Example
```
OUTPUT XXX;":COMPare:RANGe?"
```

RECord

Command `:COMPare:RECord <line_mid_screen>`

The RECord command allows you to scroll the compare listing vertically. The command specifies the record number relative to the trigger that the analyzer will highlight at the center of the screen.

`<line_mid_screen>` An integer from 0 to +10,484,736

Example
```
OUTPUT XXX;":COMPare:RECord 0"
```

Query `:COMPare:RECord?`

The RECord query returns the record number currently in the center of the screen. The RECord query is only valid when Record Mode is enabled.

Returned Format

```
[COMPare:RECord?]  <rec_mid_screen><NL>
```

`<rec_mid_screen>` An integer from 0 to the maximum number of records

Example

```
OUTPUT XXX;":COMPare:RECord 5"
```

RUNTil

Command

```
:COMPare:RUNTil { OFF | LT, <value> | GT,<value> |
INRange, <value>, <value> | OUTRange, <value>,
<value> | EQUal | NEQual}
```

The RUNTil (run until) command allows you to define a stop condition when the trigger mode is repetitive. Specifying OFF causes the analyzer to make runs until either the STOP field is touched or the STOP command is issued.

There are four conditions based on the time between the X and O markers. Using this difference in the condition is effective only when time tags have been turned on (see the TAG command in the TRIGger subsystem). These four conditions are as follows:

- The difference is less than (LT) some value.
- The difference is greater than (GT) some value.
- The difference is inside some range (INRange).
- The difference is outside some range (OUTRange).

End points for the INRange and OUTRange should be at least 1μs apart since this is the minimum time resolution of the time tag counter.

There are two conditions which are based on a comparison of the acquired state data and the compare data image. You can run until one of the following conditions is true:

- Every channel of every label has the same value (EQUal).
- Any channel of any label has a different value (NEQual).

> The RUNTil instruction is available in both the LIST and COMPare subsystems.

`<value>` real number

Example

```
OUTPUT XXX;":COMPare:RUNTil EQUAL"
```

Query

```
:COMPare:RUNTil?
```

The RUNTil query returns the current stop criteria for the comparison when running in the repetitive trigger mode.

Returned Format

```
[COMPare:RUNTil] {OFF | LT, <value> | GT, <value> | INRange,
<value>, <value> | OUTRange, <value>, <value> | EQUal |
NEQual}<NL>
```

`<value>` real number

Example

```
OUTPUT XXX;":COMPare:RUNTil?"
```

SET

Command

```
:COMPare:SET
```

The SET command sets every state in the reference listing to "don't cares." If you send the SET command by mistake, you can immediately send the CLEar command to restore the previous data. This is the only time the CLEar command will not replace "don't cares" with zeroes

Example

```
OUTPUT XXX;":COMPare:SET"
```

11

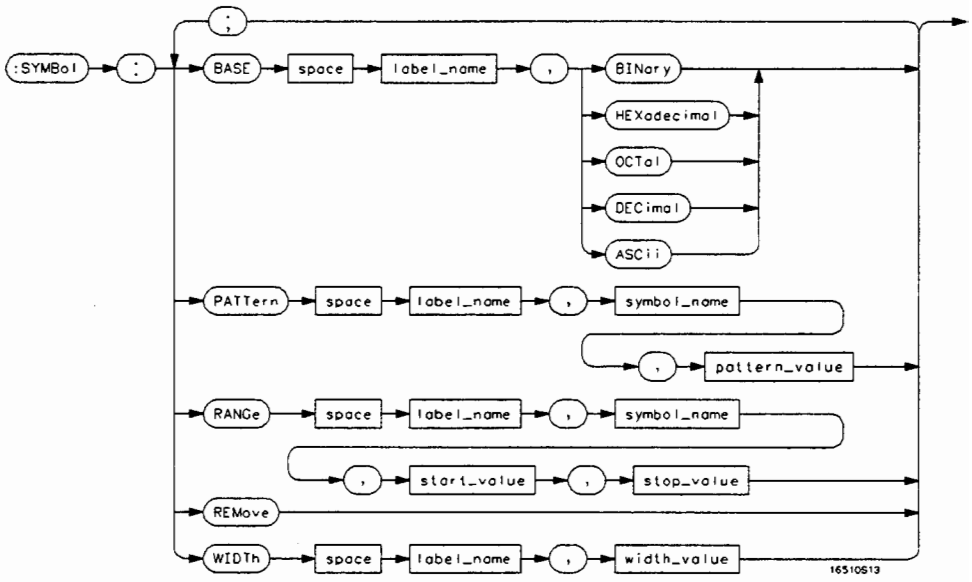
SYMBol Subsystem

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 16542A logic analyzer module. The commands in this subsystem are listed below:

- BASE
- PATTern
- RANGe
- REMove
- WIDTh

Figure 11-1



SYMBol subsystem Syntax Diagram

Table 11-1 **SYMBol Parameter Values**

Parameter	Value
label_name	string of up to 6 alphanumeric characters
symbol_name	string of up to 16 alphanumeric characters

pattern_value
start_value
stop_value
	Any on of the following:
	`#B{0|1|X}...`
	`#Q{0|1|2|3|4|5|6|7|X}...`
	`#H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}...`
	`{0|1|2|3|4|5|6|7|8|9}...`

width_value	integer from 1 to 16

Table 11-2 **SYMBol Power On Default Values**

Command	Default
BASE	'Lab1',HEX
PATTern	No defined symbols
WIDth	'Lab1',8

SYMBol

Selector

```
:SYMBol
```

The SYMBol selector is used as a part of a compound header to access the commands used to create symbols.

Example

```
OUTPUT XXX;":SYMBol:BASE 'DATA',BINary"
```

BASE

Command

```
:SYMBol:BASE <label_name>,<base_value>
```

The BASE command sets the base in which symbols for the specified label are displayed in the symbol menu. It also specifies the base in which the symbol offsets are displayed when symbols are used.

> BINary is not available for labels with more than 20 bits assigned. In this case the base will default to HEXadecimal.

`<label_name>` A string of up to 6 alphanumeric characters

`<base_value>` `{BINary|HEXadecimal|OCTal|DECimal|ASCii}`

Example

```
OUTPUT XXX;":SYMBol:BASE 'DATA',HEXadecimal"
```

PATTern

Command

```
:SYMBol:PATTern <label_name>,<symbol_name>,
<pattern_value>
```

The PATTern command allows you to create a pattern symbol for the specified label. Because don't cares (X's) are allowed in the pattern value, they must always be expressed as a string. You may still use different bases except for decimal numbers.

`<label_name>` A string of up to 6 alphanumeric characters

`<symbol_name>` A string of up to 16 alphanumeric characters

`<pattern_value>`

```
{#B{0|1|X}... |
 #Q{0|1|2|3|4|5|6|7|X}... |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F|X}... |
   {0|1|2|3|4|5|6|7|8|9}... }
```

Example

```
OUTPUT XXX;":SYMBol:PATtern 'STAT', 'MEM_RD','#H01XX'"
```

RANGe

Command

```
:SYMBol:RANGe <label_name>,<symbol_name>,
<start_value>,<stop_value>
```

The RANGe command allows you to create a range symbol containing a start value and a stop value for the specified label. The values may be in binary (#B), octal (#Q), hexadecimal (#H) or decimal (default). Don't cares (X's) are disallowed for this command.

`<label_name>` A string of up to 6 alphanumeric characters

`<symbol_name>` A string of up to 16 alphanumeric characters

`<start_value>`

```
{#B{0|1}... |
 #Q{0|1|2|3|4|5|6|7}... |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}... |
   {0|1|2|3|4|5|6|7|8|9}... }
```

`<stop_value>`

```
{#B{0|1}... |
 #Q{0|1|2|3|4|5|6|7}... |
 #H{0|1|2|3|4|5|6|7|8|9|A|B|C|D|E|F}... |
   {0|1|2|3|4|5|6|7|8|9}... }
```

Example

```
OUTPUT XXX;":SYMBol:RANGe 'STAT', 'IO_ACC','0','#H000F'"
```

REMove

Command

```
:SYMBol:REMove
```

The REMove command deletes all symbols from a specified machine.

Example

```
OUTPUT XXX;":SYMBol:REMove"
```

WIDTh

Command

```
:SYMBol:WIDTh <label_name>,<width_value>
```

The WIDTh command specifies the number of characters used when displaying symbol names. However, the WIDTh command does not affect the displayed length of the symbol offset value.

`<label_name>` A string of up to 6 alphanumeric characters

`<width_value>` An integer from 1 to 16

Example

```
OUTPUT XXX;":SYMBol:WIDTh 'DATA',9 "
```

12

DATA and SETup Commands

Introduction

The DATA and SETup commands are SYSTem commands that allow you to send and receive block data between the HP 16542A and a controller. Use the DATA instruction to transfer acquired state data and the SETup instruction to transfer the instrument configuration. This chapter explains how to use these commands.

The DATA and SETup commands are useful for the following tasks:

- Re-loading a configuration into the logic analyzer.
- Post-processing acquired data in a controller.

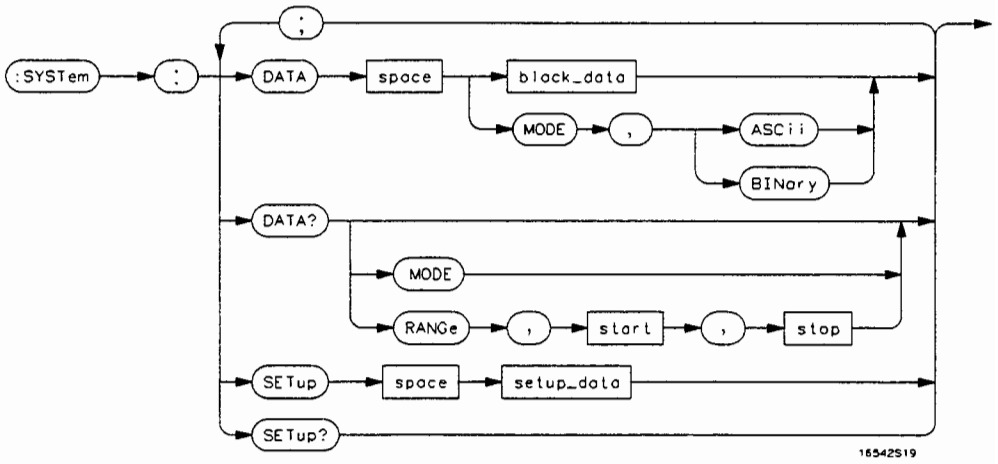
The format and length of data depends on the instruction being used and the configuration of the instrument. The SYSTem:DATA section describes each part of the data as it will appear when used by the DATA instruction. The beginning byte number, the length in bytes, and a short description is given for each part of the block data. This is intended to be used primarily for post-processing of data in the controller.

Do not change the data in the controller if you intend to send it back into the logic analyzer. Changes made to the data in the controller could have unpredictable results when sent back to the logic analyzer.

The commands in this subsystem are listed below:

- DATA
- SETUP

Figure 12-1



DATA and SETup SYStem Syntax Diagram

SYSTem:DATA

Command

```
:SYSTem:DATA <data_block>

:SYSTem:DATA MODE,{ASCii|BINary}
```

The SYSTem:DATA command transmits limited configuration information and acquisition memory information from the controller to the HP16542A State analyzer. The SYSTem:DATA query returns this same information from the HP16542A to the controller.

Because of the quantity of data available for loading from the HP16542A, a number of options are available for the DATA command and query to provide some flexibility in specifying what data is to be unloaded from the HP16542A State Analyzer.

The DATA command is used to transfer block data into the HP16542A State Analyzer. The block data consists of bytes of information captured by the analyzer. The number of bytes depends on the analyzer configuration and the information captured. The DATA command, without any range options, is the only way to load acquisition data back into the analyzer.

Care should be taken when transferring the data string into the logic analyzer. No parameter checking is performed, so out-of-range values could cause an instrument lockup.

The <block_data> parameter consists of a <block_length_specifier> and a variable number of <section>s. The <block_length_specifier> always takes the form #8DDDDDDDD. Each D represents an ASCII digit ('0' through '9'). The value of the string DDDDDDDD represents the total length of all following <section>s. For example, if the total length of the <section>s is 14522 bytes, the <block_length_specifier> would be #800014522.

Each <section> consists of a <section_header> and <section_data>. The <section_data> format varies for each <section> and may be any length. For the DATA instruction, there is only one <section> (the DATA section). The data section is composed of a preamble followed by the acquisition data. The data section varies in length based on analyzer configuration.

<block_data>	<block_length_specifier><section>
<block_length_specifier>	#8<length>
<length>	total size of the data section in bytes including the <section_header>
<section>	<section_header><section_data>
<section_header>	16 bytes, see page 6 for description
<section_data>	see page 6 for description

Example

```
OUTPUT XXX;":SYSTem:DATA" <block_data>
```

> The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for <length> don't forget to include the length of the section headers.

Query

```
:SYSTem:DATA?
```

```
:SYSTem:DATA? MODE
```

```
:SYSTem:DATA? RANGe,<start>,<stop>
```

The DATA query is used to transfer block data out of the HP16542A State Analyzer. The block data consists of bytes of information captured by the analyzer. The number of bytes depends on the analyzer configuration and the information captured. The DATA query (without any range options) unloads the full width and depth of the analyzer.

> The data sent by the SYSTem:DATA query reflects the configuration of the analyzer when the last run was performed. Any changes made since then through either front panel operations or remote control programming commands do not affect the stored data.

Returned Format

```
[:SYSTem:DATA] <block_data><NL>
```

Data <section_header> Description

The data <section_header> is always 16 bytes long. The data <section_header> bytes are defined as follows:

0 - 9 **10 bytes** - section name. Always 10 characters in length. The section name is left justified in the string, as required, by padding the name with blanks to the right. Example, "DATA ".

10 **1 byte** - reserved.

11 **1 byte** - module id. Always 2A hex (42 decimal) for the HP16542A.

12 - 15 **4 bytes** - number of bytes in <section_data>.

Data <section_data> Description

For the SYSTem:DATA instruction, the <section_data> parameter consists of two parts. The first part is 156 bytes of <data_preamble> information. The <data_preamble> documents the minimal analyzer configuration required to decode the acquisition data. The second part is a variable number (depends on analyzer configuration) of acquisition data bytes. The data_preamble and acquisition data are described below.

<data_preamble> Description

The data_preamble gives information describing the logic analyzer configuration and the data that was captured. The preamble consists of 156 bytes defined as follows:

0-1 **2 bytes** - instrument id. Always 409E hex (16542 decimal) for the HP16542A.

2-3 **2 bytes** - revision code.

4 **1 byte** - data acquisition mode. The analyzer data acquisition clocking mode. Can have the following values:

1 = timing
2 = state

5 **1 byte** - number of acquisition pods. Will be a number between 1 and 5.

6-20 **15 bytes** - reserved.

21-25 **5 bytes** - pod clocking. These bytes are one or zero based on the clocking mode selected for each pod. A '0' indicates that a pod is clocked on the master clock and a '1' indicates a pod is clocked on the slave clock.

Bytes 21 through 25 are assigned values from the top according to the number of pods in the configuration. If a pod is not used (see byte 5), the contents of the byte assigned to that pod has no meaning. For example, if there are two expansion cards, bytes 21 through 23 would have a value of one or zero. Bytes 24 and 25 would be don't cares. See figure 12-1 for an illustration of how bytes 21 through 25 are assigned for various analyzer configurations.

Figure 12-2

BYTE	MASTER AND FOUR EXPANDERS	MASTER AND THREE EXPANDERS	MASTER AND TWO EXPANDERS	MASTER AND ONE EXPANDER	MASTER ONLY
21	TOP EXPANDER	TOP EXPANDER	TOP EXPANDER	EXPANDER	MASTER
22	2ND TOP EXPANDER	MIDDLE EXPANDER	BOTTOM EXPANDER	MASTER	
23	2ND BOTTOM EXPANDER	BOTTOM EXPANDER	MASTER	UNUSED	
24	BOTTOM EXPANDER	MASTER			
25	MASTER				

16540E22

Clock Identification

26-33 **8 bytes** - sample period. This 64 bit binary number represents the number of pico-seconds per sample when in timing mode. The hex number 0000 0000 0000 2710 is equal to 10000 decimal (i.e.: 10,000 ps or 10 ns per sample).

34-55 **22 bytes** - reserved.

56-59 **4 bytes** - total number of valid states in the acquisition memory.

60-63 **4 bytes** - number of prestore states in acquisition memory (states before trigger).

64-67 **4 bytes** - number of poststore states in acquisition memory (states after trigger).

68-69 **2 bytes** - analyzer configuration. Will be a number between 0 and 7. Can be used to determine the number of bytes per state.

70 **1 byte** - acquisition data valid.

0 = no valid data
1 = data valid

This byte takes precedence over the data counts previously defined (bytes 56-59).

71 **1 byte** - trace point found.

0 = trace point forced
1 = trace point found

72-73 **2 bytes** - record mode.

0 = records off
1 = records on

74-77 **4 bytes** - memory length.

78-81 **4 bytes** - number of records.

82-85 **4 bytes** - record length.

86-87 **2 bytes** - percent prestore.

88-155 **68 bytes** - reserved.

Width (in bytes) of a state can be determined by using the number of acquisition pods (byte 5) and the analyzer configuration (bytes 68 and 69) with the table below.

Analyzer Configuration	State Width (in bytes)
0	2 × number of pods
1	1 × number of pods
2	2
3	1
4	2 × number of pods
5	1 × number of pods
6	2
7	1

The state that contains the trace point is always state number zero. Prestore states are numbered down from zero (negative numbers) and poststore states are numbered up from zero.

If the trace point was found (byte 71 set to one) the total number of states will equal the number of prestore states plus the number of poststore states plus one. If a tracepoint was forced (byte 71 cleared) the data will all be prestore data.

The location of the trace point in the data stream can be determined by using the number of prestore states as an offset into the data stream. If the trace point was forced, there is no data that actually corresponds to the trace point. Therefore, the last state in stream would be state number -1.

<acquisition_data> Description

The acquisition data starts with byte 156 of the <section_data>. The acquisition data can be in either binary or ASCII format (see DATA MODE command).

The number of bytes in the acquisition data will vary depending on the number of pods in the configuration, the number of states acquired and the mode of the transfer (ASCII vs. binary).

In binary transfer mode the total number of bytes is calculated as follows:

bytes = total valid states × width (in bytes)

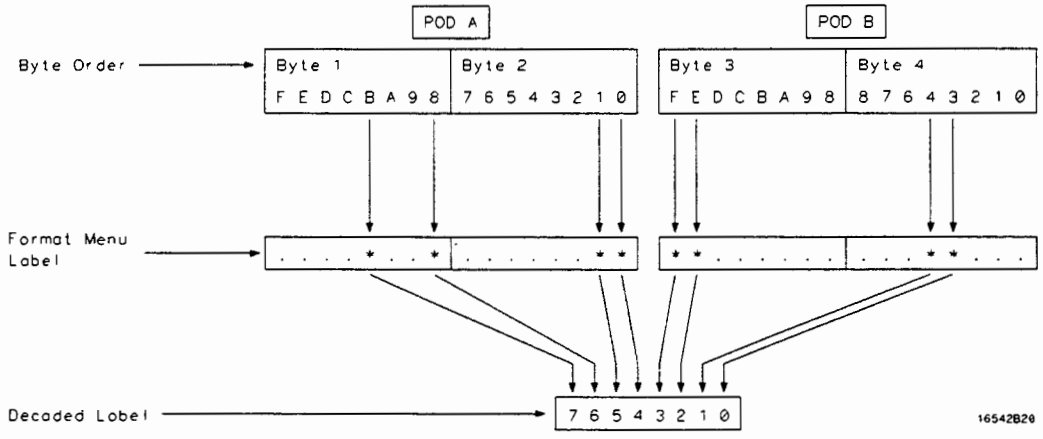
In ascii transfer mode the total number of bytes is calculated by transferring data using the SYStem:DATA? RANge query with the <start> and <stop> parameters set to 1. The total number of bytes that are transferred is calculated by taking the string length of the data returned by this query times the number of states you want to transfer. For example, if the string length of the single state query is 10 and the number of states you want to transfer is 100 then the total number of transferred bytes will be 1000.

The way in which data bytes are transfered in binary transfer mode is dependent on the number of HP 16542A expander cards in the system, the width as specified in the Configuration menu, and the memory length as specified in the Trigger menu. For example, in a 2 card system with boards in slots A and B and the Configuration width set to 32, the bytes are transferred in the following order:

POD	Channels	Byte number
A	15..8	1
A	7..0	2
B	15..8	3
B	7..0	4
A	15..8	5
.	.	.
.	.	.
.	.	.
B	7..0	last byte

The binary transfer is done without regard to labels or channel setups in the Format menu. The method used to extract and decode the channels from the binary data into their label is illustrated in figure 12-3.

Figure 12-3



Decoding the Binary Data Into Labels

Each channel bit assigned to a label is placed in the proper bit location starting with the most significant bit on the left.

If the system width for a 2 card system is 16 then the bytes are transferred in the following order:

POD	Channels	Byte number
All	15..8	1
All	7..0	2
All	15..8	3
All	7..0	4
All	15..8	5
.	.	.
.	.	.
.	.	.
All	7..0	last byte

If the system width for a 2 card system is 8 then the bytes are transferred in the following order:

POD	Channels	Byte number
All	7..0	1
All	7..0	2
All	7..0	3
All	7..0	4
All	7..0	5
All	7..0	6
All	7..0	7
.	.	.
.	.	.
.	.	.
All	7..0	last byte

In ASCII transfer mode, the total number of bytes is a function of what data labels are enabled, the base of the labels, the width of the labels, and whether record mode is enabled.

Query `SYSTem:DATA? RANGe <start>,<stop>`

The SYSTem:DATA? RANGe query is used to transfer a subset of the acquired state data from the analyzer to a controller. If the acquisition data was acquired with record mode enabled, the transfer range is expressed as record numbers. If record mode is disabled, the transfer range is expressed as states.

To transfer a single state or record, the start and stop values of the range must be the same.

> If either range limit exceeds the current range of data in the analyzer, the limit will be set to a legal value and an error code will be returned.

> The preamble data for a range transfer reflects the data in the range that was transferred. Specifically, the number of valid states, prestore states, and poststore states are changed to reflect the range of states in the data block.

Returned Format `[:SYSTem:DATA] <block_data><NL>`

`<start>` any valid state or record number

`<stop>` any valid state or record number greater than or equal to <start>

Command `:SYSTem:DATA MODE,{BINary|ASCii}`

The SYSTem:DATA MODE command is used to specify the data format of the acquisition data in subsequent data transfers. The format can be either BINARY or ASCII. The transfer mode applies to DATA queries only.

> Loading data into the HP16542A must always be a binary transfer.
> The binary transfer is much faster than the ASCII transfer. The binary transfer also uses a much smaller number of bytes to accomplish the data transfer.

Query `:SYSTem:DATA? MODE`

The SYSTem:DATA? MODE query is used to read the current data format selection.

Return Format `[:SYSTem:DATA] MODE {BINary|ASCii}<NL>`

Example

```
OUTPUT XXX;":SYSTem:DATA? MODE"
```

SYSTem:SETup

Command `SYSTem:SETup <block_data>`

The SYSTem:SETup command is used to load a configuration block into the logic analyzer. The configuration that is loaded should be a configuration block that was previously uploaded via a SYSTem:SETup query.

`<block_data>`	<block_length_specifier><section>...
`<block_length_specifier>`	#8<length>
`<length>`	total size of the configuration sections in bytes including the <section_header>
`<section>`	<section_header><section_data>
`<section_header>`	16 bytes, see following pages for description
`<section_data>`	format is unique to analyzer configuration

Query `:SYSTem:SETup?`

The SETup query is used to upload a configuration block from the analyzer into a controller.

Returned Format `[:SYSTem:SETup] <block data><NL>`

`<block_data>`	<block_length_specifier><section>...
`<block_length_specifier>`	#8<length>
`<length>`	total size of the configuration sections in bytes including the <section_header>
`<section>`	<section_header><section_data>
`<section_header>`	16 bytes, see page 15 for description
`<section_data>`	format is unique to analyzer configuration

Setup <section_header> Description

The <section_header> is always 16 bytes long. The <section_header> bytes are defined as follows:

0 - 9	**10 bytes** - section name. Always 10 characters in length. The section name is left justified in the string if needed (i.e.: padded with blanks to the right of the name). Example, "DATA ".
10	**1 byte** - reserved.
11	**1 byte** - module id. Always 2A hex (42 decimal) for the HP16542A.
12 - 15	**4 bytes** - number of bytes in <section_data>.

The total length of a section is 16 (for the section header) plus the length of the section data. So when calculating the value for <length>, don't forget to include the length of the section headers.

Example

```
OUTPUT XXX;":SYSTem:SETup?"
```

13

Examples

Introduction

This chapter contains programming examples for the HP16542A Logic Analyzer Module. The examples are written in the C programming language using the HP-IB interface as the communication channel. While the programs in this chapter run as shown, they are not guaranteed to run on different hardware configurations and should be modified for your specific application.

C Programming Examples

The C programming examples contained in this section are written using the ANSI C standard — ANSI X3.159-1989. By writing the examples using ANSI C, the examples should be easily ported to any computer which has an ANSI C compliant compiler. The compilers used for the example programs are the HP-UX C Version 8.0 Compiler and the Microsoft C/C++ Version 7.0 Compiler.

While the example programs contain some error handling capability, error checking is kept to a minimum in order to clearly show how to program a single HP16542A Logic Analyzer Module. The level of error checking should be increased for application programs. Also, the programs will require modification if you have more than one HP 16542A module.

The example programs are structured in such a way as to make them portable from one computer system to another. There are only four HP-IB function calls that are computer system dependent and may, therefore, require re-writing: Write_IO, Read_IO, Init_IO, and Set_EOI.

Example HP-IB function calls are shown for an HP9000 computer running the HP-UX unix operating system, a personnel computer (PC) running MS-DOS and using an HP HP-IB card, and a PC running MS-DOS and using a National Instrument's GPIB card.

The example programs, while written for specific applications, show typical procedures required by most application programs. The following applications are covered by the example programs:

- Image Processing
- Telecommunications
- ASIC Testing
- Uploading Data to a Lotus Spreadsheet

The function calls used in the application programs are described in the HP16542A Functions section of this chapter.

Example Programs' Header Files

The example programs use the following three headers files:

hpib.h
16542.h
proto.h

The hpib.h file contains the definitions and function prototypes for the HP-IB communications interface.

```
/*********************
 * File Name: hpib.h *
 *********************/

#include <stdlib.h>
#include <string.h>

/*********************************************************
 * Leave the following lines commented out if compiling *
 * on a PC with a National Instruments GPIB Interface   *
 *********************************************************/

/* #define UNIX       /* Uncomment if compiling on HP-UX
                         system */
/* #define HP         /* Uncomment if compiling on PC
                         with HP interface */

#ifdef UNIX
#define UNLISTEN       '?'    /* The HP-IB unlisten
                                 command */
```

```
#define BUS_ADDRESS     7     /* Command to get bus address
                                 for open interface */
#define HP16500_ADDRESS 7     /* Unix 16500 bus address */
#else
#ifdef HP
#define ISC             7L    /* HP HP-IB Interface
                                 Select Code */

#define HP16500_ADDRESS 707L /* PC HP-IB 16500 bus
                                 address */
#else  /* defines for National HP-IB card */
#define ISC             0     /* National HP-IB Interface
                                 Select Code */

#define HP16500_ADDRESS 7     /* National HP-IB 16500 bus
                                 address */
#endif /* HP */
#endif /* UNIX */

#define MAX_SIZE 16000        /* Maximum buffer size for
                                 Read_IO and Write_IO */

#define TRUE  1
#define FALSE 0
#define ON    1
#define OFF   0

/* Global variables used by Read_IO and Write_IO */

int eoi;
int hpib_id;

/* HP-IB Prototypes */

void error_handler( int, char * );
int  Init_IO( void );
int  Write_IO( int, char *, size_t );
int  Read_IO( int, char *, size_t );
int  Close_IO( int );
void Set_EOI( int );
```

The 16542.h file contains the definitions and data types for the HP 16542A functions.

```
/**********************
 * File Name: 16542.h *
 **********************/

#define MAX_LABELS  128
#define HEADER_SIZE 182
#define MAX_WIDTH    81

#define STATE  0
#define TIMING 1

#define RISING  0
#define FALLING 1
#define BOTH    2

#define BINARY  0
#define HEX     1
#define OCTAL   2
#define DECIMAL 3
#define ASCII   4
#define SYMBOL  5
#define TWOS    6

#define FROM_TRIGGER          0
#define FROM_START            1
#define FROM_XMARKER          2
#define FROM_CURRENT_POSITION 3

#define CARDA 1
#define CARDB 2
#define CARDC 3
#define CARDD 4
#define CARDE 5

struct list
{
   int mask[10][8];
   char label[7];
};
```

The proto.h file contains the function protoypes for the HP16542A.

```
/**********************
 * File Name: proto.h *
 **********************/

int  Check_spoll( char );
int  Compare( char *, char *, int );
int  Configure_16542( char *, int );
void Copy_to_reference( void );
void Enable_compare( void );
void fill_bit( int, unsigned, unsigned char * );
void Find_error( void );
int  Find_mask( struct list [], char *, const char *, int );
int  Format( int, int, char *, char *);
int  Get_Xmarker_State( int );
void Initialize_16500( void );
int  Read_data( const char * );
int  Read_setup( const char * );
int  Read_to_start( FILE *, int );
void Run_trace();
int  Save_listing( char *, struct list [], int, int, char * );
int  Save_lotus( char *, struct list [], int, int, char * );
int  Save_window( char *, char *, int, int, int, int );
int  Search_O_Pattern( long, int );
int  Search_X_Pattern( long, int );
void Select( int );
int  Set_Column_Label( int, char *, int );
int  Set_trigger( char, char *, char *, int );
int  Set_X_Pattern( char *, char * );
int  Set_O_Pattern( char *, char * );
int  Write_data( const char * );
int  Write_setup( const char * );
```

Imaging Example

Modern day imaging systems have reached a point where displayed image resolutions of 1024 by 1024 pixels is common place. Capturing an entire image requires a logic analyzer with memory depth of 1Mbyte or more when verifying image processing hardware.

While it is necessary to capture an entire image, you may only be interested in a small portion of the picture on screen. The picture of figure 13-1 shows an example image of a commonly used video test pattern. The box area represents the area of data that is of interest.

Figure 13-1

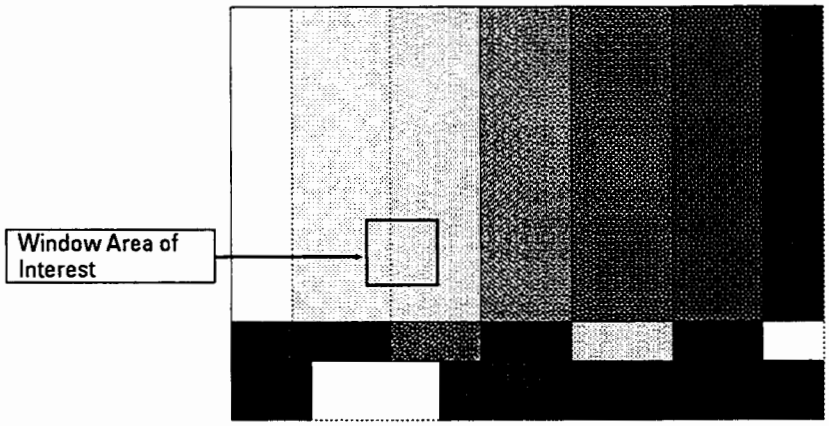


Image Window

The following example program shows how to acquire a whole image of data but window in on a portion of the screen image.

```
#include <stdio.h>
#include <string.h>
#include "hpib.h"
#include "16542.h"
#include "proto.h"

void
main( void )
{
   struct list listing[80];
   int last;
   char outfile[80];

   /* Initialize the IO port */

   hpib_id = Init_IO();

   /* Initialize the logic analyzer */

   Initialize_16500();

   /* Select the HP16542A card */

   Select( CARDC );

   /* Setup the configuration menu */

   Configure_16542( "Record", STATE );

   /* Setup the format menu */

   Format( 0, FALLING, "Data", "0000000011111111" );

   /* Setup the trigger menu */

   Set_trigger( 'A', "Data", "00", HEX );

   /* Capture the data */

   Run_trace();

   /* Upload the data from the analyzer */

   Read_data( "win.dat" );
```

```
    last = 0;

    /* Set the data mask used to decode the
     * data associated with a label
     */

    Find_mask( listing, "0000000011111111",
               "Data", last );

    /* Save the data in a temporary file */

    Save_listing( "win.dat", listing, last, 2, "win.lst" );
    sprintf( outfile, "win.win" );

    /* Save a small window of data */

    Save_window( "win.lst", outfile, 0, 10, 0, 10 );

    /* Close the IO port */

    Close_IO( hpib_id );
}
```

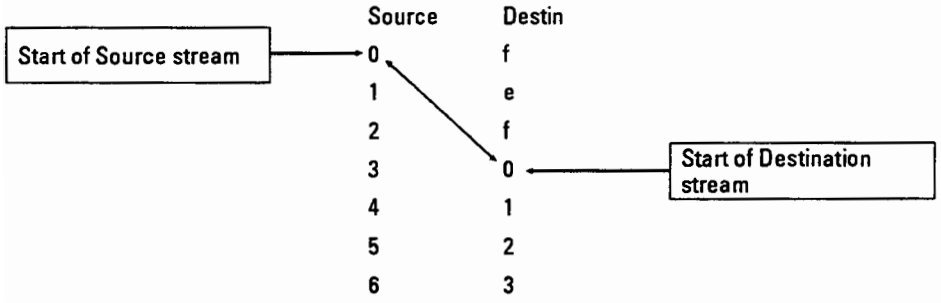
Telecommunications Example

When sending a digital data stream through a communications channel, there are several stages where the data stream can become corrupted. For certain applications, the amount of data in the data stream is quite large requiring the HP16542A Logic Analyzer Module to capture the data.

Finding the cause of the corrupted data involves capturing both the source and destination data streams during a single logic analyzer acquisition. The source data is then compared to the destination data. The two data streams should be identical unless the destination data stream is corrupted during transmission.

Unfortunately, communications channels have some propagation delay associated with them. This delay causes the source and destination data acquired by the logic analyzer to be off by some number of states in the listing menu. This makes comparing the two data streams difficult because the source data bytes are not next to the destination data bytes in the listing menu.

The following figure shows an example listing where the destination stream arrives at the logic analyzer 3 states later than the source stream.



The following program takes care of this problem by reading both data streams into the computer. The two data streams are compared taking into account the offset caused by the propagation delay.

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"
#include "proto.h"

void
main( void )
{
   struct list listing[80];
   int last;

   /* Initialize the IO port */

   hpib_id = Init_IO();

   /* Initialize the logic analyzer */

   Initialize_16500();

   /* Select the HP16542A card */

   Select( CARDC );

   /* Setup the configuration menu */

   Configure_16542( "Telecom", STATE );
```

```
   /* Setup the format menu */

   Format( 0, FALLING, "In", "0000000011111111" );
   Format( 0, FALLING, "Out", "1111111100000000" );

   /* Setup the trigger menu */

   Set_trigger( 'A', "In", "00", HEX );

   /* Capture the data */

   Run_trace();

   /* Upload the data from the analyzer */

   Read_data( "telecom.dat" );

   last = 0;

   Find_mask( listing, "0000000011111111",
        "In", last );

   Save_listing( "telecom.dat", listing, last, 2,
                 "telecom.in" );
   last = 0;

   Find_mask( listing, "1111111100000000",
              "Out", last );

   Save_listing( "telecom.dat", listing, last, 2,
                 "telecom.out" );

   /* Compare the uploaded data with the
    * data in a reference file using
    * an offset starting position
    */

   Compare( "telecom.in", "telecom.out", 4 );

   /* Close the IO port */

   Close_IO( hpib_id );
}
```

ASIC Testing Example

As the complexity of integrated circuits in the form of ASIC devices has increased, the ability to verify correct operation of integrated circuits has become more difficult and time consuming. Using the HP 16542A logic analyzer, an ASIC's functionality can be verified to a high degree of certainty in a time efficient manner using a "Go/No Go" test.

One method of testing an ASIC is to apply test data to the input pins while the logic analyzer captures the response data on the critical output pins of the ASIC. By comparing the acquired data against previously captured data from a known good reference ASIC, the correct operation of the ASIC under test can be verified.

The following example program downloads from a computer a previously acquired known good ASIC acquisition to the compare memory of the HP16542 module. Data is then acquired from an ASIC under test and compared against the compare memory data.

Also shown, is an example program which sets up the analyzer in compare mode, captures the data from a reference ASIC, and uploads the setup and data files to a computer.

```
/************************************************
 * Program Name: download.c                     *
 * Description:                                 *
 *      This program downloads the setup and data *
 * files used to compare a reference ASCI to an  *
 * ASIC under test.                             *
 ************************************************/
#include <stdio.h>
#include "hpib.h"
#include "16542.h"
#include "proto.h"

void
main( void )
{
   /* Initialize the IO port */

   hpib_id = Init_IO();
```

```
    /* Initialize the logic analyzer */

    Initialize_16500();

    /* Select the HP16542A card */

    Select( CARDC );

    /* Download the setup data to the analyzer */

    Write_setup( "asic.set" );

    /* Download the data to the analyzer */

    Write_data( "asic.dat" );

    /* Copy the downloaded data to the compare memory */

    Copy_to_reference();

    /* Capture the data from the ASIC under test */

    Run_trace();

    /* Check for differences between the captured data
     * and the reference data
     */

    Find_error();

    /* Close the IO port */

    Close_IO( hpib_id );
}
```

```
/************************************************
 * Program Name: upload.c                       *
 * Description:                                 *
 *      This program setups the analyzer for a  *
 * future comparison of a reference ASIC to     *
 * an ASIC under test.  The program also uploads *
 * both the analyzer setup and the reference    *
 * ASIC's data.                                 *
 ************************************************/

#include <stdio.h>
#include "hpib.h"
#include "16542.h"
#include "proto.h"

void
main( void )
{

   /* Initialize the IO port */

   hpib_id = Init_IO();


   /* Initialize the logic analyzer */

   Initialize_16500();

   /* Select the HP16542A card */

   Select( CARDC );

   /* Setup the configuration menu */

   Configure_16542( "ASIC Test", STATE );

   /* Enable the compare memory */

   Enable_compare();

   /* Setup the format menu */

   Format( 0, FALLING, "Data", "0000000011111111" );
```

```
    /* Setup the trigger menu */

    Set_trigger( 'A', "Data", "00", HEX );

    /* Capture the reference data */

    Run_trace();

    /* Upload the setup data from the analyzer */

    Read_setup( "asic.set" );

    /* Upload the data from the analyzer */

    Read_data( "asic.dat" );

    /* Close the IO port */

    Close_IO( hpib_id );
}
```

Lotus Spreadsheet Example

Many applications require post processing of information captured by a logic analyzer to determine if the hardware being tested is performing properly. One such case is a digital filter where the expected output of the digital filter can be calculated given the input data to the filter.

This can be accomplished by using a spreadsheet and the input and output data of the digital filter captured by the logic analyzer. The input and output data from the digital filter is read into the computer and placed in a file in the format required by the spreadsheet. By importing the file into the spreadsheet and applying the digital filter's mathematical algorithm to the input data using the spreadsheet math functions, a comparison can be made between the computed output of the digital filter and the measured output of the digital filter.

The following example shows how to acquire the data from a circuit under test, how to read the information into the computer, and how to save the information in a Lotus spreadsheet import file. The program does not take into account that there is a limit to how much data any particular spreadsheet can import from a file.

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"
#include "proto.h"

struct list listing[80];
int last;

void
main( void )
{

   /* Initialize the IO port */

   hpib_id = Init_IO();

   /* Initialize the logic analyzer */

   Initialize_16500();


   /* Select the HP16542A card */

   Select( CARDC );

   /* Setup the configuration menu */

   Configure_16542( "Telecom", STATE );

   /* Setup the format menu */

   Format( 0, FALLING, "Data", "0000000010101010" );
   Format( 0, FALLING, "Next", "0000000011111111" );

   /* Setup the trigger menu */

   Set_trigger( 'A', "Data", "00", HEX );

   /* Capture the data */

   Run_trace();
```

```
    /* Upload the data from the analyzer */

    Read_data( "telecom.dat" );

    /* Set the data mask used to decode the
     * binary data for each label */

    last = Find_mask( listing, "0000000010101010",
                      "Data", last );
    Find_mask( listing, "0000000011111111",
               "Next", last );

    /* Save the data in a lotus formated
     * ASCII file
     */

    Save_lotus( "telecom.dat", listing, last, 2,
                "telecom.imp" );

    /* Close the IO port */

    Close_IO( hpib_id );
}
```

HP16542A Functions

The HP16542A functions build the appropriate command strings sent to the HP16542A Logic Analyzer Module using the generic I/O function, Write_IO. Data is read from the analyzer using the generic I/O function, Read_IO. An application program uses these functions to setup the Configuration Menu, Format Menu, and Trigger Menu.

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/*****************************************************
 * Function Name: Initialize_16500                   *
 * Passed Parameters: None                           *
 * Return Value: TRUE                                *
 * Description:                                      *
 *      The function initializes the HP16500 logic   *
 * analyzer by clearing all status registers and     *
 * enabling all Service Request Status and Standard  *
 * Event Status register bits.                       *
 *****************************************************/
void
Initialize_16500( void )
{
   char id[80];
   int term;

   /* Clear the 16500 */

   Write_IO( hpib_id, "*CLS", 4 );

   /* Enable all Service Request Status Register bits */

   Write_IO( hpib_id, "*SRE 255", 8 );

   /* Enable operation complete bit */

   Write_IO( hpib_id, "*OPC", 4 );

   /* Enable all Standard Event Status Register bits */

   Write_IO( hpib_id, "*ESE 255", 8 );

   /* Identify the instrument */

   Write_IO( hpib_id, "*IDN?", 5 );
   term = Read_IO( hpib_id, id, 80 );
   id[term] = '\0';
   printf( "%s\n\n", id );

   Write_IO( hpib_id, ":CARDcage?", 10 );
```

```
   term = Read_IO( hpib_id, id, 80 );
   id[term] = '\0';
   printf( "%s\n\n", id );

}
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/************************************************
 * Function Name: Select                        *
 * Passed Parameters:                           *
 *    card_number - integer specifying the card *
 *                  number.                     *
 * Return Value: None                           *
 * Description:                                 *
 *      The function determines which card in the *
 * HP16500 card cage commands are sent.  This   *
 * function must be invoked before sending any  *
 * commands to the HP16542A 100MHz State Analyzer *
 * Card.                                        *
 ************************************************/
void
Select( int card_number )
{
   char command[80];

   sprintf( command, ":SELECT %d", card_number );

   Write_IO( hpib_id, command, strlen( command ) );

}
```

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/************************************************
 * Function Name: Configure_16542                *
 * Passed Parameters:                            *
 *    name - string of 10 or fewer characters    *
 *    clock - integer equal to STATE or TIMING   *
 * Return Value:                                 *
 *    TRUE - successful completion of function   *
 *    FALSE - unsuccessful completion of function *
 * Description:                                  *
 *      The function sets the Machine name and   *
 * type fields of the configuration menu.Machine *
 * type must be either STATE or TIMING otherwise *
 * an error message is generated.  The function  *
 * also removes any previously defined labels in *
 * the format menu.                              *
 ************************************************/
int
Configure_16542( char *name, int clock )
{
   char command[50];

   if( strlen( name ) > 10 )
   {
      printf("Configure_16542 Error: Name is greater");
      printf(" than 10 characters\n");
      return( FALSE );
   }

   switch( clock )
   {
      case STATE:
         sprintf( command, ":CONF:NAME '%s';TYPE STATE",
                  name );
         break;
      case TIMING:
         sprintf( command, ":CONF:NAME '%s';TYPE TIMING",
                  name );
         break;
      default:
         printf("Configure_16542 Error: Clock must be");
         printf(" STATE or TIMING\n");
```

```
        return( FALSE );
    }

    Write_IO( hpib_id, command, strlen( command ) );

    /* Remove all the old labels */

    sprintf( command, ":FORM:REMove ALL" );
    Write_IO( hpib_id, command, strlen( command ) );

    return( TRUE );
}
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/*****************************************************
 * Function Name: Format                             *
 * Passed Parameters:                                *
 *    pod_number - integer specifying the pod        *
 *                 number.                           *
 *    clock - integer equal to RISING, FALLING, or   *
 *            BOTH.                                  *
 *    name - string of 6 or less characters          *
 *           specifying the label name.              *
 *    pins - string of 16 or less characters         *
 *           specifying the pin assignments for      *
 *           the pod in binary.  A "1" represents an *
 *           active channel.                         *
 *                                                   *
 *               example: 10100011                   *
 *                                                   *
 * Return Value:                                     *
 *    TRUE - successful completion of function       *
 *    FALSE - unsuccessful completion of function    *
 * Description:                                      *
 *      The function sets the voltage threshold for  *
 * the clock and pins to the TTL voltage level.      *
 * The master J clock edge is set according to the   *
 * value of the clock parameter.  A label is created *
 * with the name specified by the name parameter and *
 * assigned pod channels as specified by the pins    *
 * parameter.                                        *
 *****************************************************/
```

```
int
Format( int pod_number, int clock, char *name, char *pins )
{
   char command[160];

   if( strlen( name ) > 6 )
   {
      printf( "format error: Signal name must be 6 ");
      printf( "characters or less\n");
      return( FALSE );
   }

   if( strlen( pins ) > 16 )
   {
      printf("format error: Pins must be 16 characters ");
      printf("or less\n");
      return( FALSE );
   }

   sprintf( command, ":FORM:THR%d TTL;CTHR%d TTL;",
            pod_number, pod_number );

   switch( clock )
   {
      case RISING:
         strcat( command, "MAST J,RISing" );
         break;
      case FALLING:
         strcat( command, "MAST J,FALLing" );
         break;
      case BOTH:
         strcat( command, "MAST J,BOTH" );
         break;
      default:
         printf( "format error: Clock must be RISING, ");
         printf( "FALLING, or BOTH\n" );
         return( FALSE );
   }

   Write_IO( hpib_id, command, strlen( command ) );
   sprintf( command, ":FORM:LABEL %d, '%s',POS,#B%s",
            pod_number, name, pins);
   Write_IO( hpib_id, command, strlen( command ) );
   return( TRUE );
}
```

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/*****************************************************
 * Function Name: Set_trigger                        *
 * Passed Parameters:                                *
 *    term - character A through D                   *
 *    signal_label - string of 6 or less characters  *
 *                   specifying the signal name.     *
 *    pattern - string specifying the data           *
 *              pattern searched for during trace.   *
 *    base - integer specifying the number base of   *
 *           pattern and must be BINARY, HEX,        *
 *           OCTAL, or DECIMAL.                      *
 * Return Value:                                     *
 *    TRUE - successful completion of function       *
 *    FALSE - unsuccessful completion of function    *
 * Description:                                      *
 *      The function sets the search pattern used    *
 * as a trigger condition.  The trigger condition    *
 * is specified by the pattern parameter in the      *
 * numerical base specified by the base parameter.   *
 * The signal_label parameter specifies which        *
 * label's data is used in the search for the        *
 * trigger condition.                                *
 *****************************************************/
int
Set_trigger( char term, char *signal_label,
             char *pattern, int base )
{
   char command[160];

   if( strlen( signal_label ) > 6 )
   {
      printf("Set_trigger error: Signal name must be 6 ");
      printf("characters or less\n");
      return( FALSE );
   }
```

```
   if( ( term < 'A' ) || ( term > 'D' ) )
   {
      printf("Set_trigger error: Term must be A through D\n");
      return( FALSE );
   }

   switch( base )
   {
      case BINARY:
         sprintf( command, ":TRAC:TERM %c, '%s', '#B%s'",
                  term, signal_label, pattern );
         break;
      case HEX:
         sprintf( command, ":TRAC:TERM %c, '%s', '#H%s'",
                  term, signal_label, pattern );
         break;
      case OCTAL:
         sprintf( command, ":TRAC:TERM %c, '%s', '#Q%s'",
                  term, signal_label, pattern );
         break;
      case DECIMAL:
         sprintf( command, ":TRAC:TERM %c, '%s', '%s'",
                  term, signal_label, pattern );
         break;
      default:
         printf( "Set_trigger error: Base must " );
         printf( "be BINARY, HEX, OCTAL, or DECIMAL\n" );
         return( FALSE );
   }

   Write_IO( hpib_id, command, strlen( command ) );
   strcpy( command, ":TRIG:POSTstore 100;MLENgth 4096" );
   Write_IO( hpib_id, command, strlen( command ) );
   return( TRUE );
}
```

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/************************************************
 * Function Name: Run_trace                     *
 * Passed Parameters: None.                     *
 * Return Value: None.                          *
 * Description:                                 *
 *     The function places the logic analyzer in *
 * the single acquisition mode and starts the    *
 * analyzer running.                            *
 ************************************************/
void
Run_trace( void )
{
   char command[160];
   int term;

   strcpy( command, ":RMODE SINGLE" );
   Write_IO( hpib_id, command, strlen( command ) );

   strcpy( command, ":START" );
   Write_IO( hpib_id, command, strlen( command ) );

/* See if command has completed */

   do
   {
      strcpy( command, "*OPC?" );
      Write_IO( hpib_id, command, strlen( command ) );
      term = Read_IO( hpib_id, command, 80 );
      command[term] = '\0';
   } while( command[0] != '1' );

}
```

```
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/*************************************************
 * Function Name: Read_data                      *
 * Passed Parameters:                            *
 *    data_file_name - char pointer to the name of *
 *                     the data file to which the  *
 *                     binary data read from the   *
 *                     logic analyzer is stored.   *
 * Return Value:                                 *
 *    TRUE - successful completion of function   *
 * Description:                                  *
 *      The function opens the file specified by *
 * the data_file_name parameter and writes the   *
 * binary data from the logic analyzer to this   *
 * file.  The function returns TRUE when all     *
 * information has been read.                    *
 *************************************************/
int
Read_data( const char *data_file_name )
{
   char command[160];
   char buffer[MAX_SIZE];
   int bytes;
   FILE *data_file;

   data_file = fopen( data_file_name, "wb" );
   strcpy( command, ":SYSTEM:HEADER OFF" );
   Write_IO( hpib_id, command, strlen( command ) );

   strcpy( command, ":SYStem:DATA?" );
   Write_IO( hpib_id, command, strlen( command ) );

   bytes = Read_IO( hpib_id, buffer, (size_t) MAX_SIZE );

   while( bytes == MAX_SIZE )
   {
      fwrite( buffer, 1, sizeof( buffer ), data_file );
      bytes = Read_IO( hpib_id, buffer, (size_t) MAX_SIZE );
   }

   fwrite( buffer, 1, (size_t) bytes, data_file );
```

```
   fclose( data_file );
   return( TRUE );
}

/*************************************************
 * Function Name: Compare                        *
 * Passed Parameters:                            *
 *    reference_file - char pointer to the name of *
 *                     the file containing the     *
 *                     reference data.             *
 *    listing_file - char pointer to the name of   *
 *                   the file containing the data  *
 *                   which is compared to the      *
 *                   reference data.               *
 *    offset - integer value which determines the  *
 *             start of data in the listing file.  *
 * Return Value:                                 *
 *    TRUE - successful completion of function     *
 * Description:                                  *
 *      The function compares the data in the      *
 * reference file to the data in the listing file. *
 * Any differences found are displayed on the      *
 * screen.  The offset value determines which data *
 * value is used as the starting point in the      *
 * listing file.  This allows you to align the     *
 * data in the listing file with that of the data  *
 * in the reference file if there are some number  *
 * of states that must be ignored before the       *
 * comparison begins.                            *
 *************************************************/
int
Compare( char *reference_file, char *listing_file,
         int offset )
{
   FILE *ref;
   FILE *list;
   char ref_buffer[MAX_SIZE];
   char list_buffer[MAX_SIZE];
   int i;
   long line_count;
   int bytes;

   ref = fopen( reference_file, "rb" );
```

```
if( ref == NULL )
{
   printf("Compare error: Could not open file %s\n",
           reference_file );
   return( FALSE );
}

list = fopen( listing_file, "rb" );

if( list == NULL )
{
   printf("Compare error: Could not open file %s\n",
          listing_file );
   return( FALSE );
}

fread( list_buffer, 1, offset, list );

line_count = 0L;

bytes = fread( list_buffer, 1,
               sizeof( list_buffer), list );
while( bytes == MAX_SIZE )
{
   fread( ref_buffer, 1, sizeof( ref_buffer ), ref );

   for( i = 0; i < bytes; i++ )
   {
      line_count++;

      if( ref_buffer[i] != list_buffer[i] )
      {
         printf("Line %d Reference = %u Listing = %u\n",
                line_count, (0xff & ref_buffer[i]),
                (0xff & list_buffer[i]) );
      }
   }

   bytes = fread( list_buffer, 1,
                  sizeof( list_buffer), list );
}

fread( ref_buffer, 1, sizeof( ref_buffer ), ref );
```

```
   for( i = 0; i < bytes; i++ )
   {
      line_count++;

      if( ref_buffer[i] != list_buffer[i] )
      {
         printf("Line %d Reference = %u Listing = %u\n",
                line_count, (0xff & ref_buffer[i]),
                (0xff & list_buffer[i]) );
      }
   }

   return( TRUE );
}

/*****************************************************
/* Function Name: Write_data                         *
 * Passed Parameters:                                *
 *    data_file_name - char pointer to the name of   *
 *                     the data file from which the  *
 *                     binary data is read.          *
 * Return Value:                                     *
 *    TRUE - successful completion of function       *
 * Description:                                      *
 *      The function opens the file specified by     *
 * the data_file_name parameter and writes the       *
 * binary data from the file to the logic analyzer.  *
 * The function returns TRUE when all information    *
 * has been read.                                    *
 *****************************************************/
int
Write_data( const char *data_file_name )
{
   char command[80];
   char buffer[MAX_SIZE];
   size_t bytes;
   FILE *data_file;

   data_file = fopen( data_file_name, "rb" );
   strcpy( command, ":SYSTEM:HEADER OFF" );
   Write_IO( hpib_id, command, strlen( command ) );

   Set_EOI( OFF );
   strcpy( command, ":SYStem:DATA " );
   Write_IO( hpib_id, command, strlen( command ) );
```

```
   while( (bytes = fread( buffer, 1, (size_t) MAX_SIZE,
                          data_file) ) == MAX_SIZE )
   {
      Write_IO( hpib_id, buffer, bytes );
   }

   Set_EOI( ON );
   Write_IO( hpib_id, buffer, bytes );
   printf("Downloaded %d\n", bytes);

   fclose( data_file );
   return( TRUE );
}

/*****************************************************
 * Function Name: Write_setup                        *
 * Passed Parameters:                                *
 *    setup_file_name - char pointer to the name of  *
 *                      the setup file from which    *
 *                      the binary setup data is     *
 *                      read.                        *
 * Return Value:                                     *
 *    TRUE - successful completion of function       *
 * Description:                                      *
 *      The function opens the file specified by     *
 * the setup_file_name parameter and writes the      *
 * binary setup from the logic analyzer to this      *
 * file.  The function returns TRUE when all         *
 * information has been read.                        *
 *****************************************************/
int
Write_setup( const char *setup_file_name )
{
   char command[MAX_SIZE];
   char buffer[MAX_SIZE];
   size_t bytes;
   FILE *setup_file;

   setup_file = fopen( setup_file_name, "rb" );

   Set_EOI( OFF );
   strcpy( command, ":SYStem:SETup " );
   Write_IO( hpib_id, command, strlen( command ) );
```

```
   bytes = fread( buffer, 1, (size_t) MAX_SIZE,
                  setup_file );
   while( bytes == MAX_SIZE )
   {
      Write_IO( hpib_id, buffer, bytes );
      bytes = fread( buffer, 1, (size_t) MAX_SIZE,
                     setup_file );
   }

   Set_EOI( ON );
   Write_IO( hpib_id, buffer, bytes );

   fclose( setup_file );
   return( TRUE );
}

/*************************************************
 * Function Name: Read_setup                     *
 * Passed Parameters:                            *
 *    setup_file_name - char pointer to the name of *
 *                      the setup file to which the *
 *                      binary setup setup read     *
 *                      from the logic analyzer is  *
 *                      stored.                     *
 * Return Value:                                 *
 *    TRUE - successful completion of function   *
 * Description:                                  *
 *      The function opens the file specified by *
 * the setup_file_name parameter and writes the  *
 * binary setup from the logic analyzer to this  *
 * file.  The function returns TRUE when all     *
 * information has been read.                    *
 *************************************************/
int
Read_setup( const char *setup_file_name )
{
   char command[160];
   char buffer[MAX_SIZE];
   int bytes;
   FILE *setup_file;

   setup_file = fopen( setup_file_name, "wb" );

   strcpy( command, ":SYSTEM:HEADER OFF" );
   Write_IO( hpib_id, command, strlen( command ) );
```

```
   strcpy( command, ":SYStem:SETup?" );
   Write_IO( hpib_id, command, strlen( command ) );

   bytes = Read_IO( hpib_id, buffer, (size_t) MAX_SIZE);
   while( bytes == MAX_SIZE )
   {
      fwrite( buffer, 1, sizeof( buffer ), setup_file );
      bytes = Read_IO( hpib_id, buffer, (size_t) MAX_SIZE);
   }

   fwrite( buffer, 1, (size_t) bytes, setup_file );

   fclose( setup_file );
   return( TRUE );
}
#include <stdio.h>
#include <string.h>
#include "hpib.h"
#include "16542.h"
#include "proto.h"

#define MAX_COLUMN 64

unsigned char bit_list[8] = { 1,  2,  4,   8,
                              16, 32, 64, 128 };

/*****************************************************
 * Function Name: Find_mask                          *
 * Passed Parameters:                                *
 *    listing - array of structures containing the   *
 *              bit map and name for each label.     *
 *    mask - string of characters with a 1           *
 *           specifying a valid channel for a        *
 *           label.                                  *
 *    label - string specifying the name of the      *
 *            label.                                 *
 *    last - Integer specifying the last label       *
 *           in the listing array.                   *
 * Return Value:                                     *
 *    last - integer specifying the last label       *
 *           in the listing array.                   *
```

```
 * Description:                                          *
 *      The function copies the label name to            *
 * label field of the listing structure.  The            *
 * mask string contains a '1' for each assigned          *
 * channel of the label as shown in the format           *
 * menu.  For each '1' in the mask, a location is        *
 * calculated which points to the bit location           *
 * in the uploaded binary data for that channel.         *
 * For more information on the format of the             *
 * binary data, see the Data and Setup Commands          *
 * chapter.                                              *
 ********************************************************/
int
Find_mask( struct list listing[MAX_LABELS],
           char *mask, const char *label, int last )
{
   int pod_number;
   int i;
   int j;   /* Bit location in binary data */
   int k;   /* Bit location in label */

   strcpy( listing[last].label, label );

   /* Initialize mask to -1 which
    * represents a don't care channel
    */

   for( j = 0; j < 10; j++ )
   {
      for( i = 0; i  < 8; i++ )
         listing[last].mask[j][i] = -1;
   }

   k = 0;
   j = 0;
   pod_number = 0;
```

```
   /* Search for ones from the end of the mask
    * string because the least significant bit
    * is there.  The most significant bit is
    * the first character of the string.
    */

   for( i = (int) strlen( mask ); i >= 0; i-- )
   {
      if( (i/8) != pod_number )
      {
         j = 0;
         k = 0;
         pod_number = i/8;
      }

      if( mask[i] == '1' )
      {
         listing[last].mask[pod_number][k] = j;
         k++;
      }
      j++;
   }

   last++;

   return( last );
}

/*****************************************************
 * Function Name: Save_listing                       *
 * Passed Parameters:                                *
 *    data_file_name - string name of data file.     *
 *    listing - array of structures containing       *
 *              label names and bit mask.            *
 *    last - integer specifying the number of        *
 *           labels in the listing array.            *
 *    width - integer specifying the number of 8     *
 *            bit wide channels for each pod.        *
 *    ouput_file_name - string name of output file.  *
 * Return Value:                                     *
 *    TRUE - successful completion of function       *
 *    FALSE - unsuccessful completion of function    *
 * Description:                                      *
 *      The function reads in the binary data and    *
```

```
 * creates a number according to the mask for each  *
 * label.  The result is written to the output      *
 * file in binary format.                           *
 ***************************************************/
int
Save_listing( char *data_file_name,
              struct list listing[MAX_LABELS],
              int last, int width,
              char *out_file_name )
{
   FILE *data_file;
   FILE *out_file;
   char buffer[HEADER_SIZE];
   unsigned char init_number;
   unsigned char *word;
   int bytes;
   int i;        /* index of current label */
   int l;        /* index of mask bit being tested */
   int location;
   int k;        /* location of byte in buffer */

   word = &init_number;

   out_file = fopen( out_file_name, "wb" );

   if( out_file == NULL )
   {
      printf("Save_listing: Could not open output file %s\n",
             out_file_name );
      return( FALSE );
   }

   data_file = fopen( data_file_name, "rb" );

   if( data_file == NULL )
   {
      printf("Save_listing: Could not open file %s\n",
             data_file_name );
      return( FALSE );
   }

   /* Read past the header to the data */

   fread( buffer, 1, HEADER_SIZE, data_file );
```

```
while( (bytes = fread(buffer, 1, width, data_file)) ==
        width )
{
   /* Decode the bits for each label */

   for( i = 0; i <= last; i++ )
   {
      *word = 0;
      location = 0;

      for( k = 0; k < bytes; k++ )
      {
         l = 0;
         while( listing[i].mask[k][l] != -1 )
         {
            /* Set the bit for the label */

            switch( listing[i].mask[k][l] )
            {
               case 0:
                  fill_bit( location, (buffer[k] & 0x01),
                            word );
                  break;
               case 1:
                  fill_bit( location, (buffer[k] & 0x02),
                            word );
                  break;
               case 2:
                  fill_bit( location, (buffer[k] & 0x04),
                            word );
                  break;
               case 3:
                  fill_bit( location, (buffer[k] & 0x08),
                            word );
                  break;
               case 4:
                  fill_bit( location, (buffer[k] & 0x10),
                            word );
                  break;
               case 5:
                  fill_bit( location, (buffer[k] & 0x20),
                            word );
                  break;
```

```
                case 6:
                   fill_bit( location, (buffer[k] & 0x40),
                             word );
                   break;
                case 7:
                   fill_bit( location, (buffer[k] & 0x80),
                             word );
                   break;
             }             /* End Switch */

             location++;
             l++;
          }              /* End While */
       }                 /* End For k */

       fwrite( word, 1, 1, out_file);
    }                    /* End For i */
  }                      /* End While */

  fclose( out_file );
  fclose( data_file );
  return( TRUE );
}

/************************************************
 * Function Name: Save_window                   *
 * Passed Parameters:                           *
 *    data_file_name - string name of data file.*
 *    out_file_name - string name of output file.*
 *    start_col - integer specifying the starting*
 *                column of the window.         *
 *    end_col - integer specifying the ending   *
 *              column of the window.           *
 *    start_row - integer specifying the starting*
 *                row of the window.            *
 *    end_row - integer specifying the ending row*
 *              of the window.                  *
 * Return Value:                                *
 *    TRUE - successful completion of function  *
 *    FALSE - unsuccessful completion of function*
 * Description:                                 *
 *      The function reads in the binary data   *
 * starting at the start_row and start_col data.*
 * The data is assumed to be the sequentially   *
 * captured data of a bit mapped image where each*
```

```
 * data byte is a pixel.  It is also assumed that  *
 * the data is captured from left to right and top *
 * to bottom beginning with the upper left hand    *
 * corner of the image.                            *
 *                                                 *
 *  The defined integer, MAX_COLUMN, is the number *
 * of pixels for each row of the image.            *
 **************************************************/
int
Save_window( char *data_file_name, char *out_file_name,
             int start_col, int end_col,
             int start_row, int end_row )
{
   FILE *data_file;
   FILE *out_file;
   char buffer[MAX_SIZE];
   int number;
   int start;
   int i;
   int j;

   out_file = fopen( out_file_name, "w" );

   if( out_file == NULL )
   {
      printf("Save_window: Could not open output file %s\n",
              out_file_name );
      return( FALSE );
   }

   data_file = fopen( data_file_name, "rb" );

   if( data_file == NULL )
   {
      printf("Save_window: Could not open file %s\n",
              data_file_name );
      return( FALSE );
   }

   start = start_row * MAX_COLUMN + start_col - 1;

   /* Read to the start of data of interest */

   Read_to_start( data_file, start );
```

```
   for( i = start_row; i <= end_row; i++ )
   {
      number = fread( buffer, 1,
                      (size_t) (end_col - start_col + 1),
                      data_file );

      for(j = 0; j < number; j++ )
         fprintf( out_file, "%u\n", (0xff & buffer[j]) );

      start = MAX_COLUMN - end_col + start_col - 1;
      fread( buffer, 1, (size_t) start, data_file );
   }

   fclose( out_file );
   fclose( data_file );
   return( TRUE );
}

/************************************************
 * Function Name: fill_bit                      *
 * Passed Parameters:                           *
 *    location - integer location of bit in     *
 *               bit_list array.                *
 *    value - unsigned bit value being tested.  *
 *    word - unsigned char value whose bits     *
 *           are set according to value and     *
 *           location.                          *
 * Return Value: None.                          *
 * Description:                                 *
 *      This function tests if value is non-zero. *
 * If value is non-zero, sets the bit in word   *
 * determined by location.                      *
 ************************************************/
void
fill_bit( int location, unsigned value,
          unsigned char *word )
{
   if( value > 0 )
      *word = *word + bit_list[location];
}
```

```
/*************************************************
 * Function Name: Read_to_start                  *
 * Passed Parameters:                            *
 *    data_file - pointer to data file.          *
 *    number - integer number of bytes to the    *
 *             beginning of the window.          *
 * Return Value:                                 *
 *    TRUE - successful completion of function   *
 *    FALSE - unsuccessful completion of function *
 * Description:                                  *
 *      The function moves the file pointer      *
 * to the first byte of the window.              *
 *************************************************/
int
Read_to_start( FILE *data_file, int number )
{
   int number_read;
   char buffer[MAX_SIZE];

   number_read = number;
   if( number <= 0 )
      return( TRUE );

   if( number > MAX_SIZE )
   {
      while( number_read > MAX_SIZE )
      {
         fread( buffer, 1, (size_t) MAX_SIZE, data_file );
         number_read = number_read - MAX_SIZE;
      }

      fread( buffer, 1, (size_t) number_read, data_file );
   }
   else
   {
      fread( buffer, 1, (size_t) number, data_file );
   }

   return( TRUE );
}
```

```
/*****************************************************
 * Function Name: Save_lotus                         *
 * Passed Parameters:                                *
 *    data_file_name - string name of data file.     *
 *    listing - array of structures containing       *
 *              label names and bit mask.            *
 *    last - integer specifying the number of        *
 *           labels in the listing array.            *
 *    width - integer specifying the number of 8     *
 *            bit wide channels for each pod.        *
 *    ouput_file_name - string name of output file.  *
 * Return Value:                                     *
 *    TRUE - successful completion of function       *
 *    FALSE - unsuccessful completion of function    *
 * Description:                                      *
 *      The function reads in the binary data and    *
 * creates a number according to the mask for each   *
 * label.  The result is written to the output file  *
 * in a format which can be imported into a Lotus    *
 * spreadsheet.                                      *
 *****************************************************/
int
Save_lotus( char *data_file_name,
            struct list listing[MAX_LABELS],
            int last, int width,
            char *out_file_name )
{
   FILE *data_file;
   FILE *out_file;
   unsigned char number;
   unsigned char *word;
   char buffer[HEADER_SIZE];
   int bytes;
   int i;        /* index of current label */
   int l;        /* index of mask bit being tested */
   int location;
   int k;        /* index of current byte in buffer */

   word = &number;

   out_file = fopen( out_file_name, "w" );
```

```
if( out_file == NULL )
{
   printf("Show_lotus: Could not open file %s\n",
           out_file_name );
   return( FALSE );
}

data_file = fopen( data_file_name, "rb" );

if( data_file == NULL )
{
   printf("Show_lotus: Could not open file %s\n",
           data_file_name );
   return( FALSE );
}

/* Read past the header to the data */

fread( buffer, 1, HEADER_SIZE, data_file );

/* Write the labels to the output file */

for(i = 0; i <= last; i++)
   fprintf( out_file, "\"%s\",", listing[i].label );
fprintf( out_file, "\n" );

while( (bytes = fread( buffer, 1, width, data_file)) ==
        width )
{
   /* Decode the bits for each label */

   for( i = 0; i <= last; i++ )
   {
      *word = 0;
      location = 0;

      for( k = 0; k < bytes; k++ )
      {
         l = 0;
```

```
      while( listing[i].mask[k][l] != -1 )
      {
         /* Set the bit for the label */

         switch( listing[i].mask[k][l] )
         {
            case 0:
               fill_bit( location, buffer[k] & 0x01,
                         word );
               break;
            case 1:
               fill_bit( location, buffer[k] & 0x02,
                         word );
               break;
            case 2:
               fill_bit( location, buffer[k] & 0x04,
                         word );
               break;
            case 3:
               fill_bit( location, buffer[k] & 0x08,
                         word );
               break;
            case 4:
               fill_bit( location, buffer[k] & 0x10,
                         word );
               break;
            case 5:
               fill_bit( location, buffer[k] & 0x20,
                         word );
               break;
            case 6:
               fill_bit( location, buffer[k] & 0x40,
                         word );
               break;
            case 7:
               fill_bit( location, buffer[k] & 0x80,
                         word );
               break;
         }
         location++;
         l++;
      }                  /* End While */
   }                     /* End For k */
   fprintf( out_file, "%d,", *word);
}                        /* End For i */
```

```
      fprintf( out_file, "\n" );
   }                          /* End While */
}
#include <stdio.h>
#include "hpib.h"
#include "16542.h"

/************************************************
 * Function Name: Copy_to_reference             *
 * Passed Parameters:                           *
 * Return Value:                                *
 * Description:                                 *
 *      This function copies the listing to the *
 * compare memory.                              *
 ************************************************/
void
Copy_to_reference( void )
{
   char command[80];

   sprintf( command, ":COMPare:COPY" );

   Write_IO( hpib_id, command, strlen( command ) );
}

/************************************************
 * Function Name: Find_error                    *
 * Passed Parameters:                           *
 * Return Value:                                *
 * Description:                                 *
 *      This function searches for a difference in *
 * newly acquired data and the compare memory data *
 ************************************************/
void
Find_error( void )
{
   char command[80];
   int term;

   sprintf( command, ":COMPare:FIND? 1" );

   Write_IO( hpib_id, command, strlen( command ) );
```

```
   term = Read_IO( hpib_id, command, 80 );

   command[term] = '\0';
   printf("Errors = %s\n", command );
}
/************************************************
 * Function Name: Enable_compare                *
 * Passed Parameters:                           *
 * Return Value:                                *
 * Description:                                 *
 *      This function enables the compare memory *
 ************************************************/
void
Enable_compare( void )
{
   char command[80];

   sprintf( command, ":CONFig:CMEMory ENABled" );

   Write_IO( hpib_id, command, strlen( command ) );

}
```

HP-IB Interface Functions

The four generic HP-IB functions, Write_IO, Read_IO, Init_IO, and Set_EOI, must be re-written if one of the sets of given example functions does not work on your computer system.

The same header file (hpib.h) is used by different HP-IB functions. There are two lines containing define statements which are commented out that must be set for the particular system you are working on. If you are working on an HP-UX computer system, you must uncomment the #define UNIX line. For a PC based system using the Hewlett-Packard HP-IB card, uncomment the #define HP line. When using the National Instruments GPIB card on a PC, leave both lines commented out.

The contents of the hpib.h header file is located on page 13-3 of this chapter.

HP-UX HP-IB Interface Functions

The HP9000 series computers contain a built-in HP-IB card which must be configured before being used to control the HP16542A Logic Analyzer. This configuration must be performed by a Unix Super User. The following command was used to configure the interface for the programming examples:

```
mknod /dev/hpibb c 21 0x071f00
```

The Hewlett-Packard DIL library is used by the HP-IB functions and must be installed on your computer system before compiling the functions. If this library is not installed, contact your Unix Super User.

```
/*****************************************************
 * Program title: hpibunix.c                         *
 * Description:                                      *
 *                                                   *
 *      The following list of functions are used to  *
 * read and write binary data to an HP-IB port on a  *
 * HP Unix controller:                               *
 *                                                   *
 *      Read_IO( hpib_id, buffer, length );          *
 *      Write_IO( hpib_id, buffer, length );         *
 *                                                   *
 * Prior to using these functions, you must          *
 * initialize the HP-IB interface by invoking the    *
 * following function:                               *
 *                                                   *
 *      Init_IO();                                   *
 *****************************************************/

#include <stdio.h>
#include <sys/types.h>
#include <fcntl.h>
#include <dvio.h>
#include <errno.h>
#include "hpib.h"

#define ACTIVE 4

extern char *sys_errlist[];
extern int errno;
```

```
/* List of error messages */

static char error_list[4][80] =
   { "HP-IB interface problem.",
     "Could not open HP-IB device file.",
     "Error reading data from HP16500.",
     "Error writing data to HP16500."
   };

/*****************************************************
 * Function Name: error_handler                      *
 * Passed Parameters:                                *
 *    error_number - integer index value which       *
 *                   points to the error message     *
 *                   string in the error_list array. *
 *    name - char pointer pointing to the name of    *
 *           the function detecting the error.       *
 * Return value: None                                *
 * Description:                                      *
 *     This function is called when one of the hpib  *
 * functions detects a catastrophic error.  The      *
 * name of the function detecting the error and a    *
 * string describing the error are printed to the    *
 * screen and to an error log file.  This function   *
 * exits to UNIX on completion.                      *
 *****************************************************/
void
error_handler( int error_number, char *name )
{
   FILE *error_file;

   error_file = fopen( "error.log", "a" );

   if( error_file != NULL )
   {
      fprintf( error_file, "%s: %s\n", name,
               error_list[error_number] );
      fclose( error_file );
   }

   printf( "%s: %s\n", name, error_list[error_number] );

   exit( 0 );
}
```

```
/*****************************************************
 * Function Name: Init_IO                            *
 * Passed Parameters: None                           *
 * Return value:                                     *
 *   hpib_id - integer file descriptor for the open  *
 *             hpib device file.                     *
 * Description:                                      *
 * The Init_IO function must be called before using  *
 * the read or write functions.  After successfully  *
 * opening the HP-IB interface, the function resets  *
 * the interface, locks the interface, and returns   *
 * the interface file ID.  Locking the interface     *
 * prevents any other processes from using the       *
 * interface.                                        *
 *                                                   *
 * A timeout of 4 second is set in case the HP16500  *
 * or the interface does not respond during an I/O   *
 * operation.                                        *
 *****************************************************/
int
Init_IO()
{
   int hpib_id;
   char command[3];

   hpib_id = open( "/dev/hpibb", 2 );

   if( hpib_id == -1 )
   {
      printf("%s\n", sys_errlist[errno] );
      error_handler( 1, "OPEN" );
   }

   io_lock( hpib_id );

   if( io_reset( hpib_id ) == -1 )
   {
      printf("%s\n", sys_errlist[errno] );
      Close_IO( hpib_id );
      error_handler( 0, "RESET" );
   }

   /* Set the interface timeout to 4 second */

   io_timeout_ctl( hpib_id, 40000000L );
```

```
    /* Send the device clear command to HP16500 */

    command[0] = UNLISTEN;
    command[1] = listen_address( HP16500_ADDRESS );
    command[2] = 4;

    hpib_send_cmnd( hpib_id, command, 3 );

    hpib_eoi_ctl( hpib_id, ON );
    Set_EOI( ON );

    return( hpib_id );
}

/*******************************************************
 * Function Name: Write_IO                             *
 * Passed Parameters:                                  *
 *    hpib_id - integer HP-IB device file              *
 *              descriptor for the open hpib device    *
 *              file.                                  *
 *    buffer - char pointer to the buffer area         *
 *             where the data written to the HP-IB     *
 *             interface is stored.                    *
 *    length - size_t value specifying the number of   *
 *             data bytes to write to the HP-IB        *
 *             interface.                              *
 * Return value:                                       *
 *    bytes_written - integer number specifying the    *
 *                    number of data bytes written     *
 *                    to the HP-IB interface.          *
 * Description:                                        *
 *     This function writes the number of data         *
 * bytes as specified by the length parameter from     *
 * the buffer to the GPIB interface.  If the           *
 * global variable eoi is TRUE an EOI is sent with     *
 * the last data byte.  The function returns the       *
 * actual number of data bytes written.                *
 *******************************************************/
int
Write_IO( int hpib_id, char *buffer, size_t length )
{
    int bytes_written;
    int interface;
    char command[4];
```

```
    struct iodetail *iovec;
    struct iodetail *temp;
    struct iodetail init[2];

    interface = hpib_bus_status( hpib_id, BUS_ADDRESS );

    iovec = &init[0];
    temp = iovec;

    command[0] = UNLISTEN;
    command[1] = talk_address( interface );
    command[2] = listen_address( HP16500_ADDRESS );
    command[3] = '\0';

    temp->mode = HPIBWRITE | HPIBATN;
    temp->buf = command;
    temp->count = strlen( command );

    temp++;

    if( eoi )
    {
       temp->mode = HPIBWRITE | HPIBEOI;
    }
    else
    {
       temp->mode = HPIBWRITE;
    }
    temp->buf = buffer;
    temp->count = (int) length;

    bytes_written = hpib_io( hpib_id, iovec, 2 );

    if( bytes_written < 0 )
    {
       Close_IO( hpib_id );
       printf("%s\n", sys_errlist[errno] );
       error_handler( 3, "WRITE" );
    }

    bytes_written = temp->count;
    return( bytes_written );
}
```

```
/*****************************************************
 * Function Name: Read_IO                            *
 * Passed Parameters:                                *
 *    hpib_id - integer HP-IB device file            *
 *              descriptor for the open hpib device  *
 *              file.                                *
 *    buffer - char pointer to the buffer area       *
 *             where the data read from the HP-IB    *
 *             interface is stored.                  *
 *    length - size_t value specifying the number    *
 *             of data bytes to read from the HP-IB  *
 *             interface.                            *
 * Return value:                                     *
 *    bytes_read - integer number specifying the     *
 *                 number of data bytes read from    *
 *                 the HP-IB interface.              *
 * Description:                                      *
 *     This function reads the number of data        *
 * bytes as specified by the length parameter and    *
 * stores the data bytes in the memory area pointed  *
 * to by the buffer parameter.  The function returns *
 * the actual number of data bytes read which can be *
 * less the number of bytes requested.               *
 *****************************************************/
int
Read_IO( int hpib_id, char *buffer, size_t length )
{
   int bytes_read;
   int interface;
   char command[4];
   struct iodetail *iovec;
   struct iodetail *temp;
   struct iodetail init[2];

   interface = hpib_bus_status( hpib_id, BUS_ADDRESS );

   iovec = &init[0];
   temp = iovec;

   command[0] = UNLISTEN;
   command[1] = talk_address( HP16500_ADDRESS );
   command[2] = listen_address( interface );
   command[3] = '\0';

   temp->mode = HPIBWRITE | HPIBATN;
```

```
    temp->buf = command;
    temp->count = strlen( command );

    temp++;

    temp->mode = HPIBREAD;
    temp->buf = buffer;
    temp->count = (int) length;

    bytes_read = hpib_io( hpib_id, iovec, 2 );

    if( bytes_read < 0 )
    {
        Close_IO( hpib_id );
        error_handler( 2, "READ" );
    }

    bytes_read = (size_t) temp->count;

    return( bytes_read );
}

int
talk_address( int address )
{
    return( address + 64 );
}

int
listen_address( int address )
{
    return( address + 32 );
}
```

```
/*****************************************************
 * Function Name: Close_IO                           *
 * Passed Parameters:                                *
 *    hpib_id - integer value representing the open *
 *              HP-IB device file.                   *
 * Return value: TRUE                                *
 * Description:                                      *
 *      This function unlocks and closes the open    *
 * hpib device file.                                 *
 *****************************************************/
int
Close_IO( int hpib_id )
{
   if( hpib_id != -1 )
   {
      io_unlock( hpib_id );
      close( hpib_id );
   }

   return( TRUE );
}

/*****************************************************
 * Function Name: Set_EOI                            *
 * Passed Parameters:                                *
 *    value - integer value set to ON or OFF which *
 *            determines whether or not an EOI is   *
 *            sent as the last byte of information *
 *            is written during a Write_IO          *
 *            operation.                            *
 * Return value: None                                *
 * Description:                                      *
 *      This function sets the global variable eoi *
 * equal to the passed parameter value.  The value *
 * parameter should be set to ON or OFF as defined *
 * in hpib.h.  The value of eoi affects only the    *
 * Write_IO operation.                               *
 *****************************************************/
void
Set_EOI( int value )
{
   eoi = value;
}
```

Hewlett-Packard HP-IB Functions for the PC

The HP-IB functions use the Microsoft C Library file MCL_HPIB.LIB which comes with the Hewlett-Packard HP-IB card. This library must be copied to your system before the example programs are compiled. Refer to the documentation that came with the HP-IB card for instructions on installing the library.

```
/*****************************************************
 * Program title: hpibpc.c                           *
 * Description:                                      *
 *      The following list of functions are used to  *
 * read and write binary data to an HP-IB port on a  *
 * PC using the Hewlett-Packard HP-IB interface      *
 * card:                                             *
 *                                                   *
 *      Read_IO( hpib_id, buffer, length );          *
 *      Write_IO( hpib_id, buffer, length );         *
 *                                                   *
 * Prior to using these functions, you must          *
 * initialize the HP-IB interface by invoking the    *
 * following function:                               *
 *                                                   *
 *      Init_IO();                                   *
 *****************************************************/

#include <stdio.h>
#include <fcntl.h>
#include "hpib.h"

#define NOERR 0

/* HP-IB Command Library Prototypes */

int ioclear( long );
int ioreset( long );
int iotimeout( long, float );
int iooutputs( long, char *, int );
int ioenters( long, char *, int * );
int ioabort( long );
int ioeoi( long, int );
int ioeol( long, char *, int );
int iomatch( long, int, int );
```

```
/* List of error messages */

static char error_list[3][80] =
   { "HP-IB interface problem.",
     "Error reading data from HP16500.",
     "Error writing data to HP16500."
   };

/*****************************************************
 * Function Name: error_handler                      *
 * Passed Parameters:                                *
 *    error_number - integer index value which       *
 *                   points to the error message     *
 *                   string in the error_list array. *
 *    name - char pointer pointing to the name of    *
 *           the function detecting the error.       *
 * Return value: None                                *
 * Description:                                      *
 *     This function is called when one of the hpib  *
 * functions detects a catastrophic error.  The      *
 * name of the function detecting the error and a    *
 * string describing the error are print to the      *
 * screen and to an error log file.  This function   *
 * exits to DOS on completion.                       *
 *****************************************************/
void
error_handler( int error_number, char *name )
{
   FILE *error_file;

   error_file = fopen( "error.log", "a" );

   if( error_file != NULL )
   {
      fprintf( error_file, "%s: %s\n", name,
               error_list[error_number] );
      fclose( error_file );
   }

   printf( "%s: %s\n", name, error_list[error_number] );

   exit( 0 );
}
```

```
/*************************************************
 * Function Name: Init_IO                        *
 * Passed Parameters: None                       *
 * Return value: TRUE                            *
 * Description:                                  *
 *     This function must be called before any of *
 * the other HP-IB functions.  The function      *
 * initializes the HP-IB hardware and sets the   *
 * inactivity timeout for I/O operations.  This  *
 * function does not return if a fatal error is  *
 * encountered.                                  *
 *************************************************/
int
Init_IO()
{
   int error;

   error = ioreset( ISC );

   if( error != NOERR )
   {
      printf("Error = %d\n", error);
      error_handler( 0, "IORESET" );
   }

   error = iotimeout( ISC, (float) 5.0 );

   if( error != NOERR )
   {
      error_handler( 0, "IOTIMEOUT" );
   }

   error = ioclear( HP16500_ADDRESS );

   if( error != NOERR )
   {
      error_handler( 0, "IOCLEAR" );
   }

   /* Turn off End of Line string for Read_IO */

   error = iomatch( ISC, 10, 0 );

   /* Enable EOI */
```

```
   Set_EOI( ON );
   error = ioeoi( ISC, ON );

   /* Turn off End of Line string for Write_IO */

   error = ioeol( ISC, "OFF", 0 );

   return( 0 );
}

/*****************************************************
 * Function Name: Write_IO                           *
 * Passed Parameters:                                *
 *    hpib_id - integer HP-IB device file            *
 *              identifier for the open hpib device  *
 *              file.                                *
 *    buffer - char pointer to the buffer area       *
 *             where the data written to the HP-IB   *
 *             interface is stored.                  *
 *    length - size_t value specifying the number    *
 *             of data bytes to write to the HP-IB   *
 *             interface.                            *
 * Return value:                                     *
 *    bytes_read - integer number specifying the     *
 *                 number of data bytes written to   *
 *                 the HP-IB interface.              *
 * Description:                                      *
 *     This function writes the number of data       *
 * bytes as specified by the length parameter from   *
 * the buffer to the HP-IB interface.  If the        *
 * global variable eoi is TRUE an EOI is sent with   *
 * the last data byte.  The function returns the     *
 * actual number of data bytes written.              *
 *****************************************************/
int
Write_IO( int hpib_id, char *buffer, size_t length )
{
   int error;

   if( eoi )
      error = ioeoi( ISC, ON );
   else
      error = ioeoi( ISC, OFF );

   error = iooutputs( HP16500_ADDRESS, buffer, (int) length );
```

```
   if( error != NOERR )
   {
      error_handler( 2, "IOOUTPUTS" );
   }

   return( (int) length );
}

/*****************************************************
 * Function Name: Read_IO                            *
 * Passed Parameters:                                *
 *    hpib_id - integer HP-IB device file            *
 *              identifier for the open hpib device  *
 *              file.                                *
 *    buffer - char pointer to the buffer area       *
 *             where the data read from the HP-IB    *
 *             interface is stored.                  *
 *    length - size_t value specifying the number    *
 *             of data bytes to read from the HP-IB  *
 *             interface.                            *
 * Return value:                                     *
 *    bytes_read - integer number specifying the     *
 *                 number of data bytes read from    *
 *                 the HP-IB interface.              *
 * Description:                                      *
 *     This function reads the number of data bytes  *
 * as specified by the length parameter and stores   *
 * the data bytes in the memory area pointed to by   *
 * the buffer parameter.  The function returns the   *
 * actual number of data bytes read which can be     *
 * less the number of bytes requested.               *
 *****************************************************/
int
Read_IO( int hpib_id, char *buffer, size_t length )
{
   int error;
   int bytes_read;

   bytes_read = (int) length;

   error = ioenters( HP16500_ADDRESS, buffer,
                     &bytes_read );
```

```
   if( error != NOERR )
   {
      error_handler( 1, "IOENTERS" );
   }

   return( bytes_read );
}

/*************************************************
 * Function Name: Close_IO                       *
 * Passed Parameters:                            *
 *    hpib_id - integer value representing the open *
 *              HP-IB device file.                 *
 * Return value: TRUE                            *
 * Description:                                  *
 *      This function does nothing on PC based   *
 * HP-IB platforms.                              *
 *************************************************/
int
Close_IO( int hpib_id )
{
   return( TRUE );
}

/*************************************************
 * Function Name: Set_EOI                        *
 * Passed Parameters:                            *
 *    value - integer value set to ON or OFF which *
 *            determines whether or not an EOI is  *
 *            sent as the last byte of information *
 *            is written during a Write_IO         *
 *            operation.                           *
 * Return value: None                            *
 * Description:                                  *
 *      This function sets the global variable eoi *
 * equal to the passed parameter value.  The value *
 * parameter should be set to ON or OFF as defined *
 * in hpib.h.  The value of eoi affects only the   *
 * Write_IO operation.                           *
 *************************************************/
```

```
void
Set_EOI( int value )
{
   eoi = value;
}
```

National Instrument's GPIB Functions for the PC

The HP-IB functions use the Microsoft C Object file MCIB.OBJ which comes with the National Instrument GPIB card. This object file must be copied to your system before the example programs are compiled. Refer to the documentation that came with the GPIB card for instructions on installing the object file.

```
/*****************************************************
 * Program title: hpibnat.c                          *
 * Description:                                      *
 *                                                   *
 *      The following list of functions are used to  *
 * read and write binary data to a GPIB port on a    *
 * PC using the National Instruments GPIB interface  *
 * card:                                             *
 *                                                   *
 *      Read_IO( hpib_id, buffer, length );          *
 *      Write_IO( hpib_id, buffer, length );         *
 *                                                   *
 * Prior to using these functions, you must          *
 * initialize the GPIB interface by invoking the     *
 * following function:                               *
 *                                                   *
 *      Init_IO();                                   *
 *****************************************************/

#include <stdio.h>
#include <fcntl.h>
#include "c:\gpib-pc\decl.h"
#include "hpib.h"

/* GPIB Command Library Prototypes */

void SendIFC( int );
int ibtmo( int, int );
```

```
/* List of error messages */

static char error_list[3][80] =
   { "GPIB interface problem.",
     "Error reading data from HP16500.",
     "Error writing data to HP16500."
   };

/*******************************************************
 * Function Name: error_handler                        *
 * Passed Parameters:                                  *
 *    error_number - integer index value which         *
 *                   points to the error message       *
 *                   string in the error_list array.   *
 *    name - char pointer pointing to the name of      *
 *           the function detecting the error.         *
 * Return value: None                                  *
 * Description:                                        *
 *     This function is called when one of the hpib    *
 * functions detects a catastrophic error.  The        *
 * name of the function detecting the error and a      *
 * string describing the error are printed to the      *
 * screen and to an error log file.  This function     *
 * exits to DOS on completion.                         *
 *******************************************************/
void
error_handler( int error_number, char *name )
{
   FILE *error_file;

   error_file = fopen( "error.log", "a" );

   if( error_file != NULL )
   {
      fprintf( error_file, "%s: %s\n", name,
               error_list[error_number] );
      fclose( error_file );
   }

   printf( "%s: %s\n", name, error_list[error_number] );

   exit( 0 );
}
```

```
/************************************************
 * Function Name: Init_IO                       *
 * Passed Parameters: None                      *
 * Return value: TRUE                           *
 * Description:                                 *
 *     This function must be called before any of *
 * the other GPIB functions.  The function      *
 * initializes the GPIB hardware and sets the   *
 * inactivity timeout for I/O operations.  This *
 * function does not return if a fatal error is *
 * encountered.                                 *
 ************************************************/
int
Init_IO()
{

   SendIFC( ISC );

   if( ibsta & ERR )
   {
      error_handler( 0, "SendIFC" );
   }

   /* Set timeout to 3 seconds */

   ibtmo( ISC, 12 );

   if( ibsta & ERR )
   {
      error_handler( 0, "ibtmo" );
   }

   DevClear( ISC, HP16500_ADDRESS );

   if( ibsta & ERR )
   {
      error_handler( 0, "DevClear" );
   }

   Set_EOI( ON );

   return( TRUE );
}
```

```
/*************************************************
 * Function Name: Write_IO                       *
 * Passed Parameters:                            *
 *    hpib_id - integer Not used on PCs.         *
 *    buffer - char pointer to the buffer area   *
 *             where the data written to the GPIB *
 *             interface is stored.              *
 *    length - size_t value specifying the number *
 *             of data bytes to write to the GPIB *
 *             interface.                        *
 * Return value:                                 *
 *    ibcnt - integer number of data bytes written *
 *            to the GPIB interface.             *
 * Description:                                  *
 *     This function writes the number of data   *
 * bytes as specified by the length parameter from *
 * the buffer to the GPIB interface.  If the     *
 * global variable eoi is TRUE an EOI is sent with *
 * the last data byte.  The function returns the *
 * actual number of data bytes written.          *
 *************************************************/
int
Write_IO( int hpib_id, char *buffer, size_t length )
{

   if( eoi )
   {
      Send( ISC, HP16500_ADDRESS, buffer, (int) length,
            DABend );
   }
   else
   {
      Send( ISC, HP16500_ADDRESS, buffer, (int) length,
            NULLend );
   }

   if( ibsta & ERR )
   {
      error_handler( 2, "Send" );
   }

   return( (int) ibcnt );
}
```

```
/*****************************************************
 * Function Name: Read_IO                            *
 * Passed Parameters:                                *
 *    hpib_id - integer Not used on PCs.             *
 *    buffer - char pointer to the buffer area       *
 *             where the data read from the GPIB     *
 *             interface is stored.                  *
 *    length - size_t value specifying the number    *
 *             of data bytes to read from the GPIB   *
 *             interface.                            *
 * Return value:                                     *
 *    ibcnt - integer number of data bytes read      *
 *            from the GPIB interface.               *
 * Description:                                      *
 *     This function reads the number of data bytes *
 * as specified by the length parameter and stores   *
 * the data bytes in the memory area pointed to by   *
 * the buffer parameter.  The function returns the   *
 * actual number of data bytes read which can be     *
 * less the number of bytes requested.               *
 *****************************************************/
int
Read_IO( int hpib_id, char *buffer, size_t length )
{

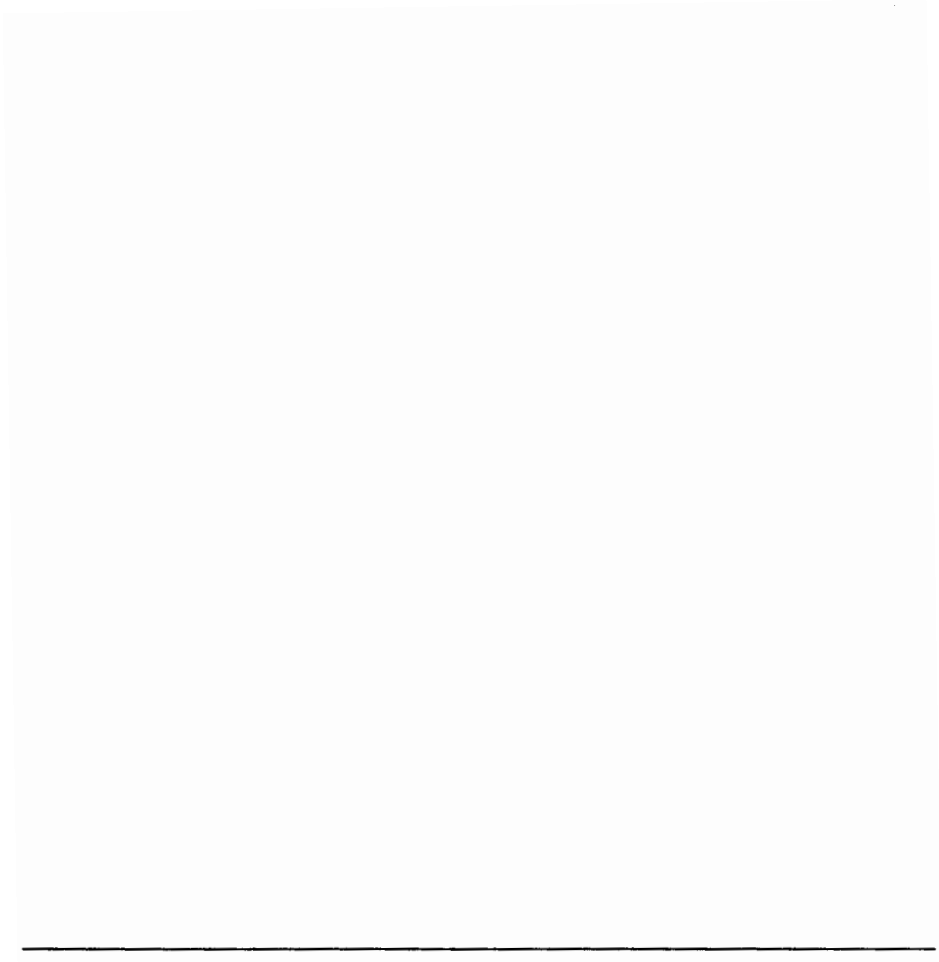
   Receive( ISC, HP16500_ADDRESS, buffer,
            (unsigned long) length, STOPend );

   if( ibsta & ERR )
   {
      error_handler( 1, "Receive" );
   }

   return( (int) ibcnt );
}
```

```
/*************************************************
 * Function Name: Close_IO                       *
 * Passed Parameters:                            *
 *    hpib_id - integer value representing the open *
 *              GPIB device file.                  *
 * Return value: TRUE                            *
 * Description:                                  *
 *      This function does nothing on PC based GPIB *
 * platforms.                                    *
 *************************************************/
int
Close_IO( int hpib_id )
{
   return( TRUE );
}

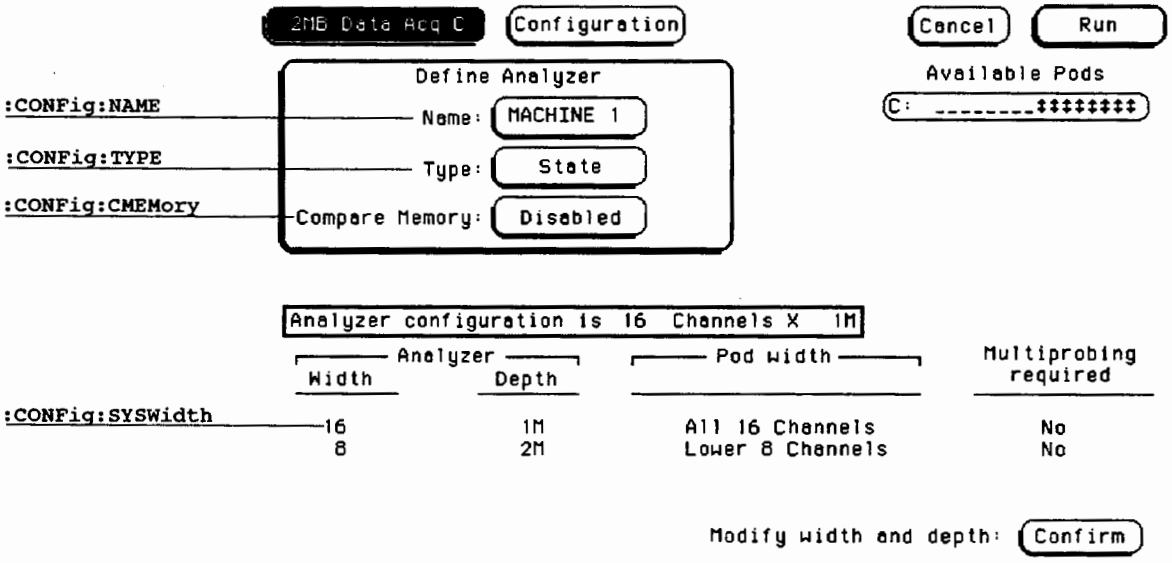
/*************************************************
 * Function Name: Set_EOI                        *
 * Passed Parameters:                            *
 *    value - integer value set to ON or OFF which *
 *            determines whether or not an EOI is  *
 *            sent as the last byte of information *
 *            is written during a Write_IO        *
 *            operation.                          *
 * Return value: None                            *
 * Description:                                  *
 *      This function sets the global variable eoi *
 * equal to the passed parameter value.  The value *
 * parameter should be set to ON or OFF as defined *
 * in hpib.h.  The value of eoi affects only the   *
 * Write_IO operation.                           *
 *************************************************/
void
Set_EOI( int value )
{
   eoi = value;
}
```



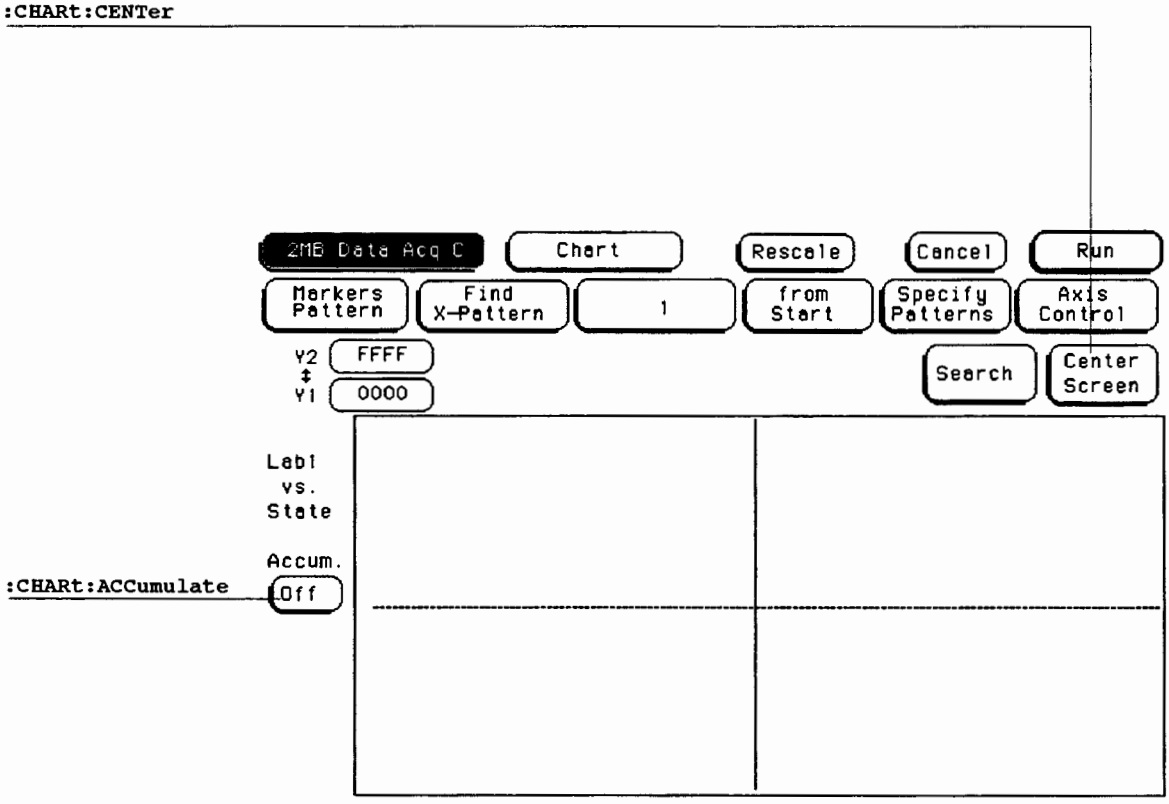
14

Menus and Commands

Configuration

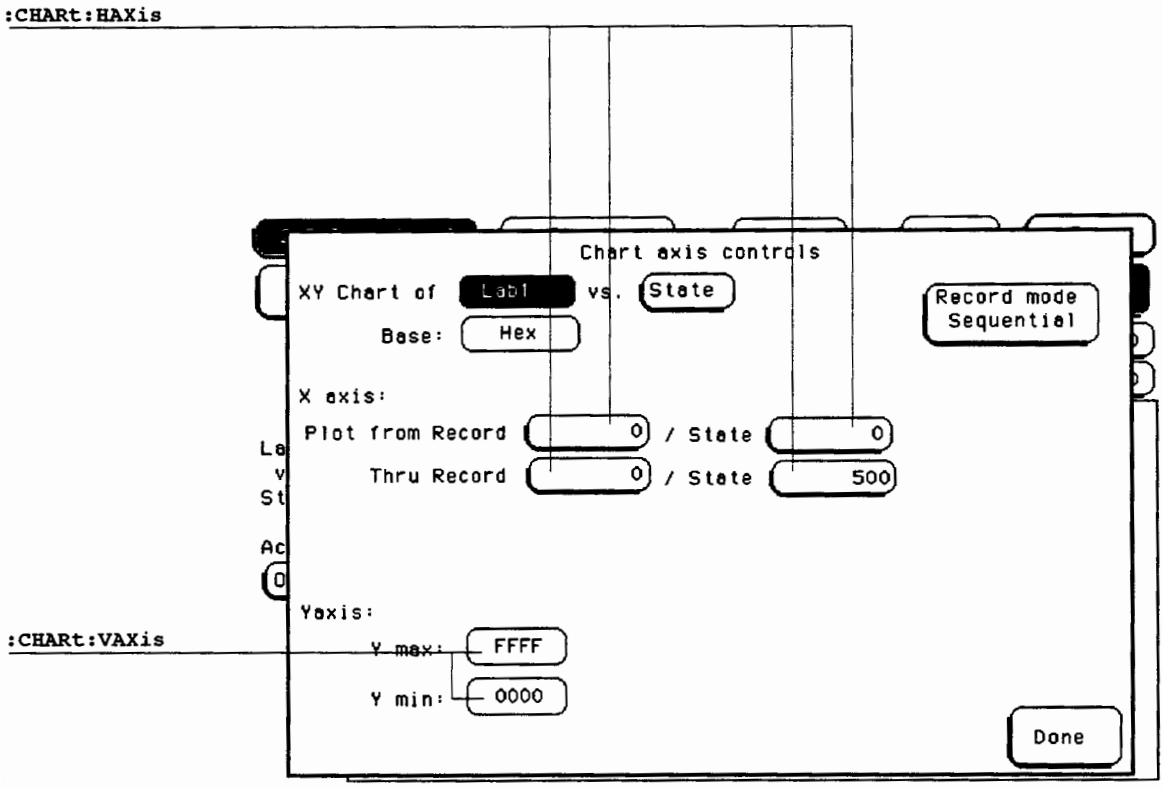


Chart


:CHARt:CENTer
2MB Data Acq C
Chart
Rescale
Cancel
Run
Markers Pattern
Find X-Pattern
1
from Start
Specify Patterns
Axis Control
Y2
Y1
FFFF
0000
Search
Center Screen
Labl vs. State
Accum.
:CHARt:ACCumulate
Off

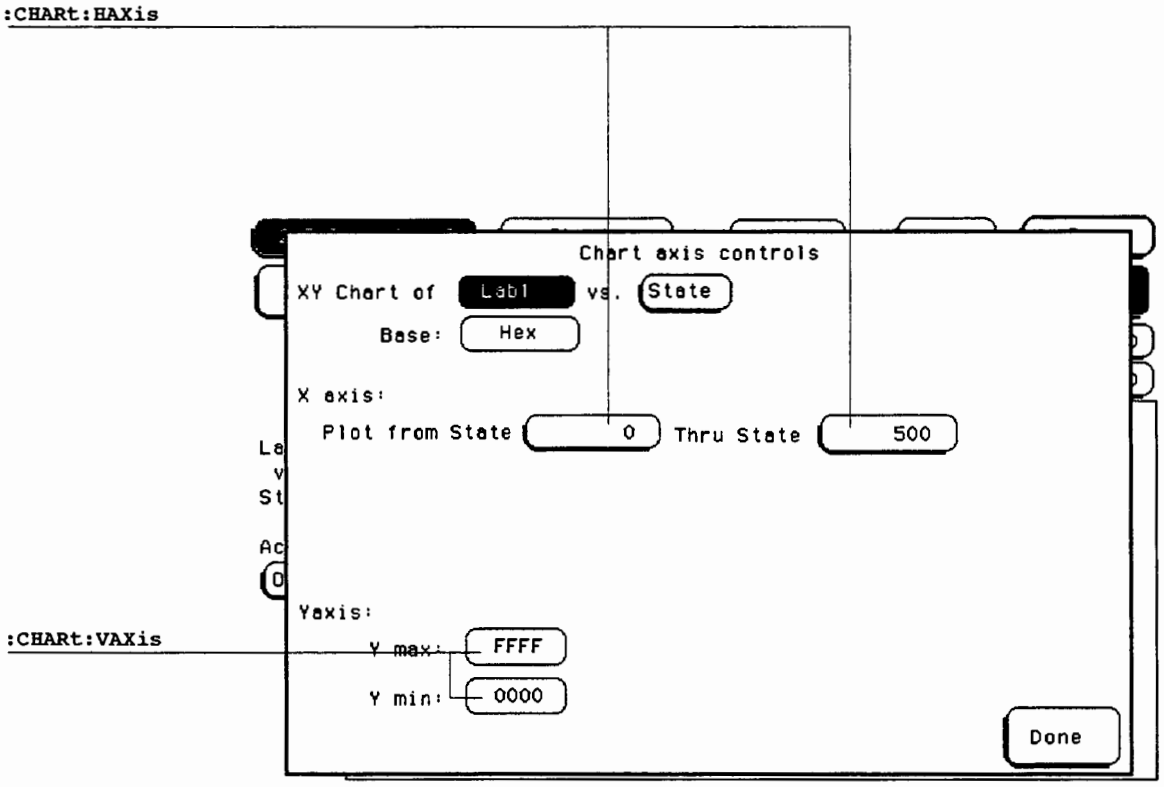
Chart

Record Mode On

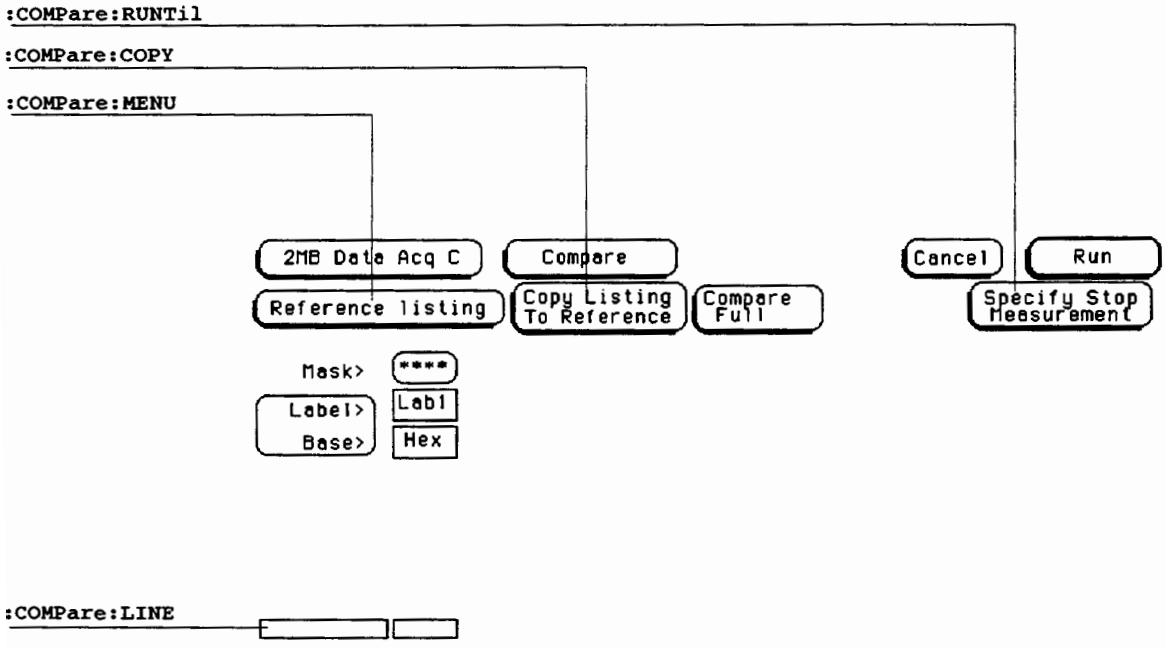


Chart

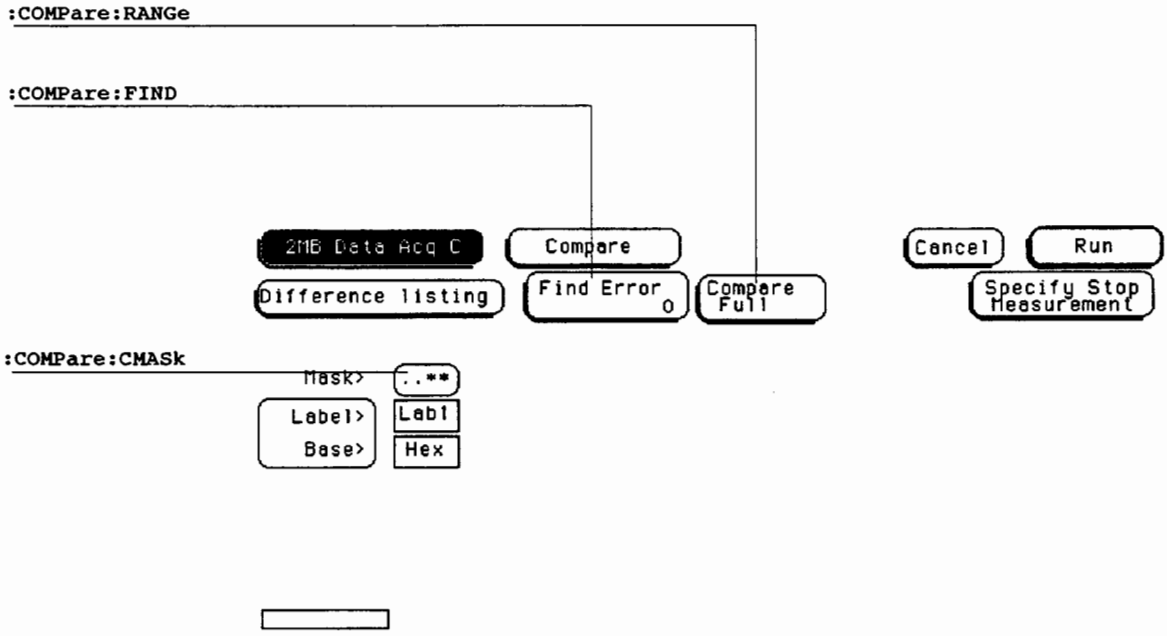
Record Mode Off



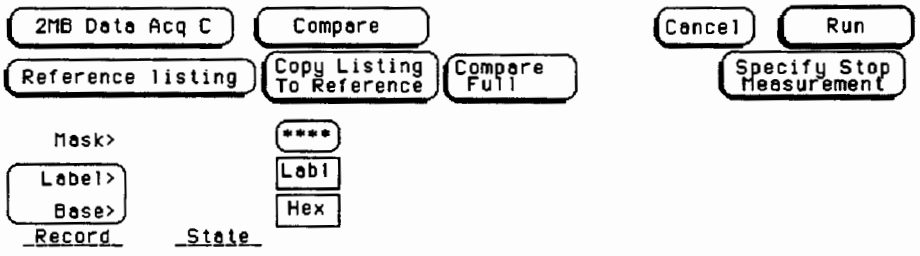
Compare


:COMPare:RUNTil
:COMPare:COPY
:COMPare:MENU
2MB Data Acq C
Compare
Cancel
Run
Reference listing
Copy Listing To Reference
Compare Full
Specify Stop Measurement
Mask>
****
Label>
Labl
Base>
Hex
:COMPare:LINE

Compare

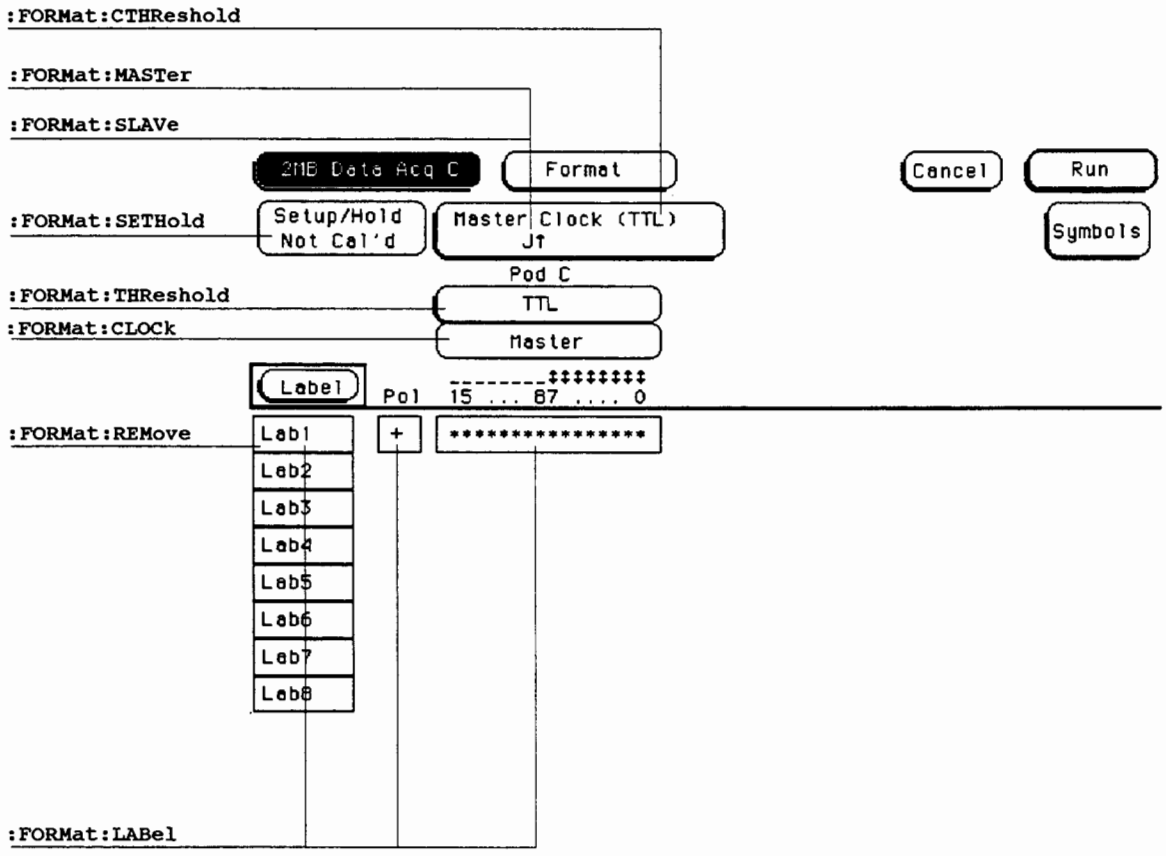

:COMPare:RANGe
:COMPare:FIND
2MB Data Acq C
Compare
Cancel
Run
Difference listing
Find Error
0
Compare Full
Specify Stop Measurement
:COMPare:CMASk
Mask>
..**
Label>
Labl
Base>
Hex

Compare

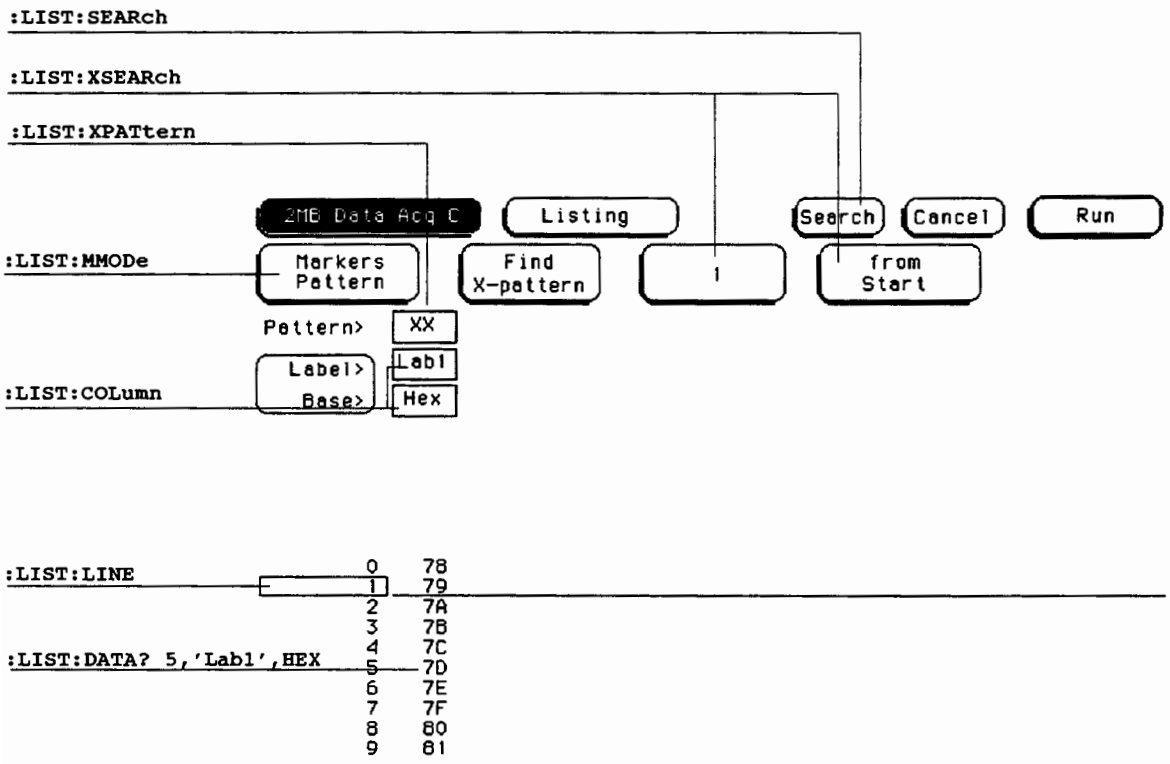


:COMPare:RECord

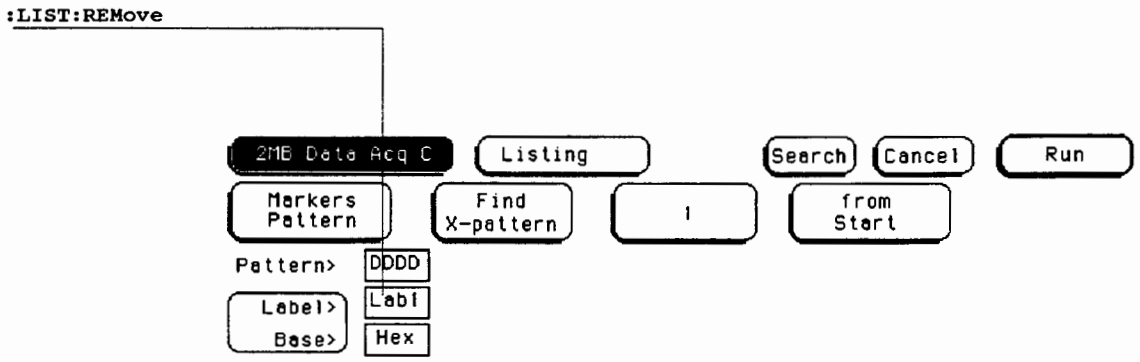
Format


:FORMat:CTHReshold
:FORMat:MASTer
:FORMat:SLAVe
2MB Data Acq C
Format
Cancel
Run
:FORMat:SETHold
Setup/Hold
Not Cal'd
Master Clock (TTL)
Jt
Symbols
Pod C
:FORMat:THReshold
TTL
:FORMat:CLOck
Master
Label
Pol
15 ... 87 0
:FORMat:REMove
Lab1
+
Lab2
Lab3
Lab4
Lab5
Lab6
Lab7
Lab8
:FORMat:LABel

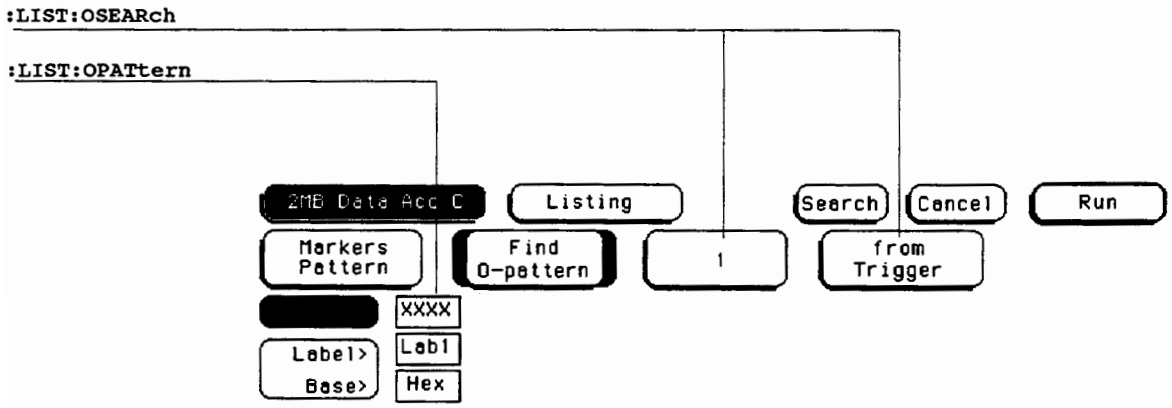
State Listing



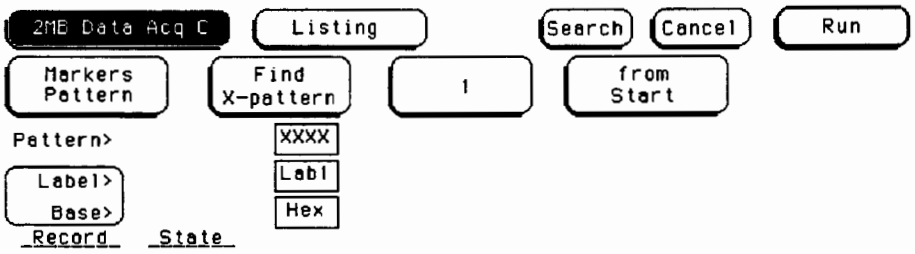
State Listing



State Listing

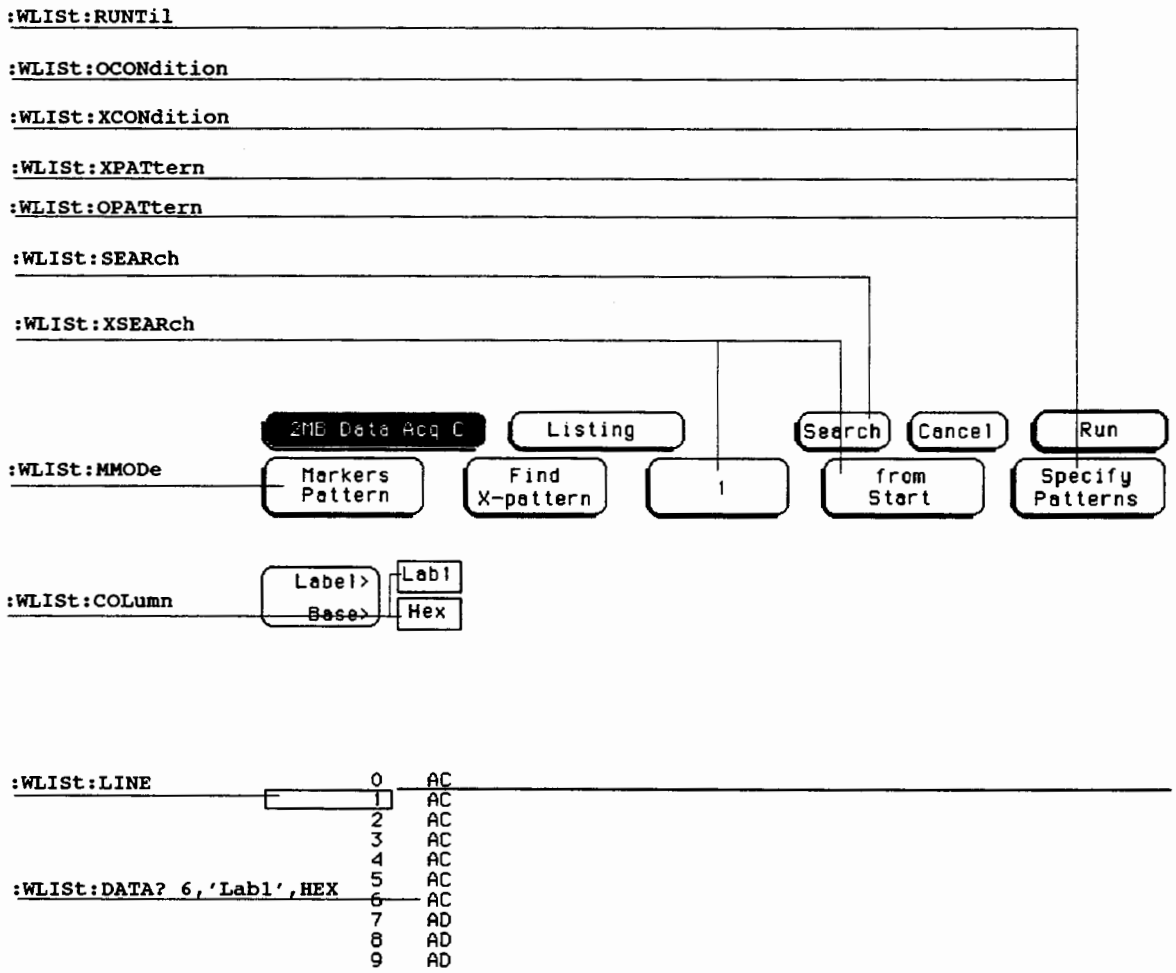


State Listing

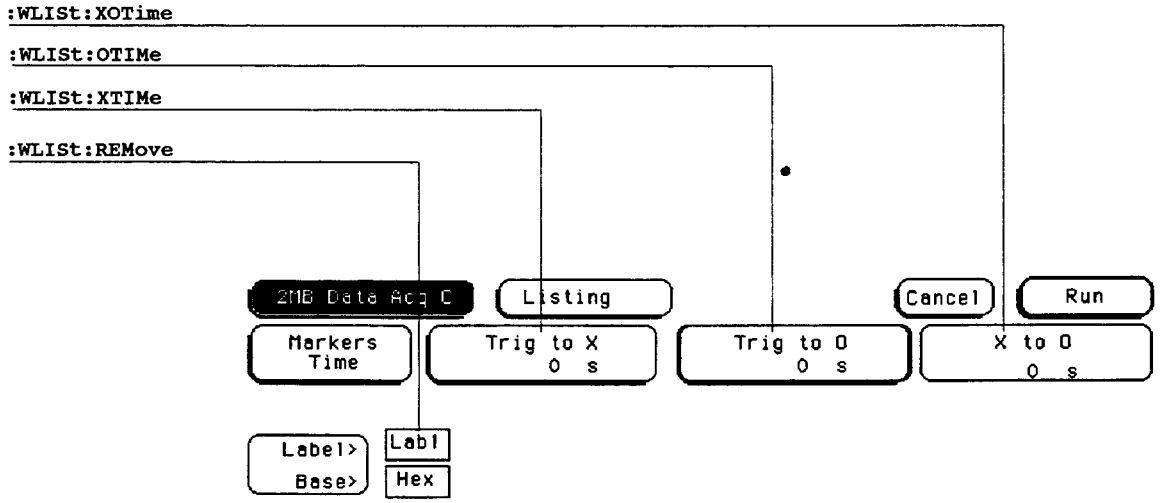


:LIST:RECord

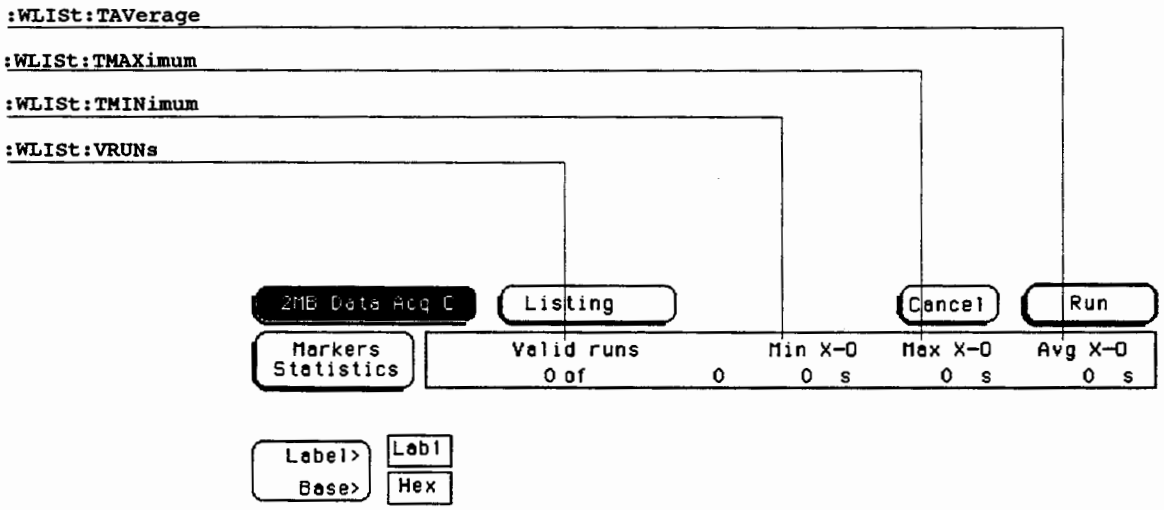
Timing Listing



Timing Listing

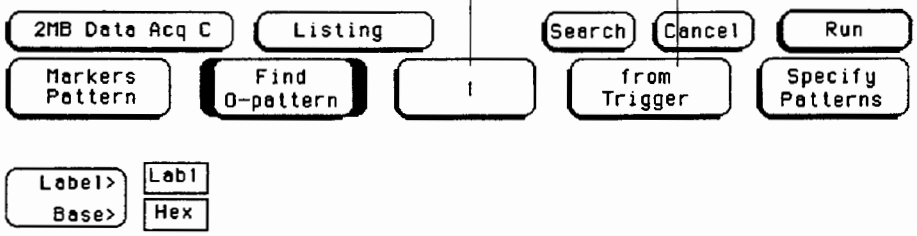


Timing Listing

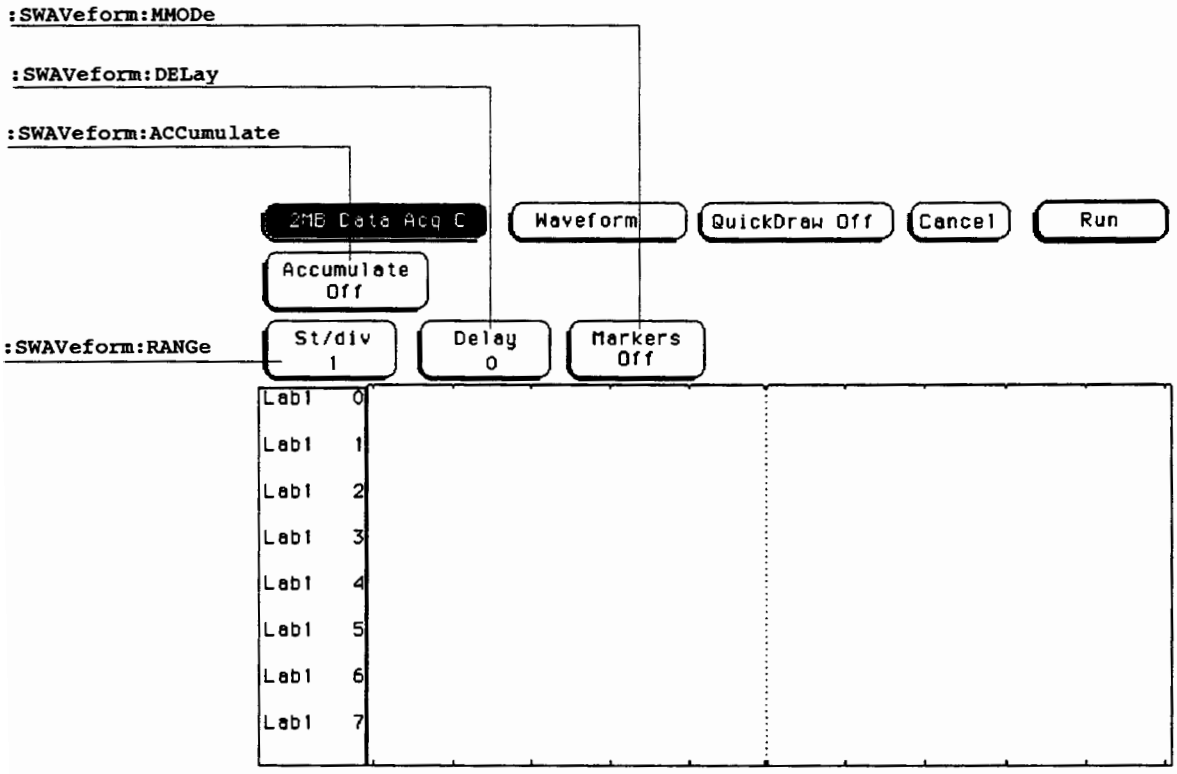


Timing Listing

:WLISt:OSEarch



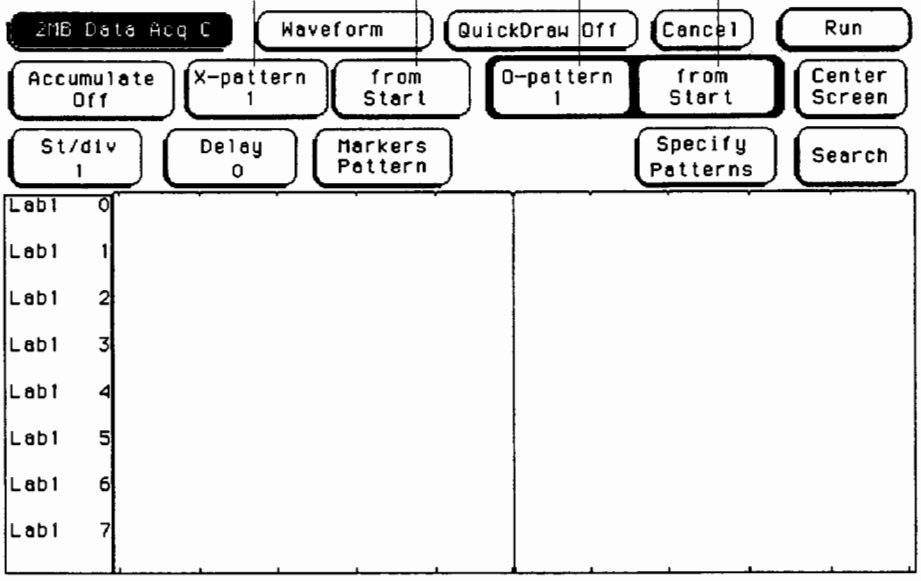
State Waveform



State Waveform

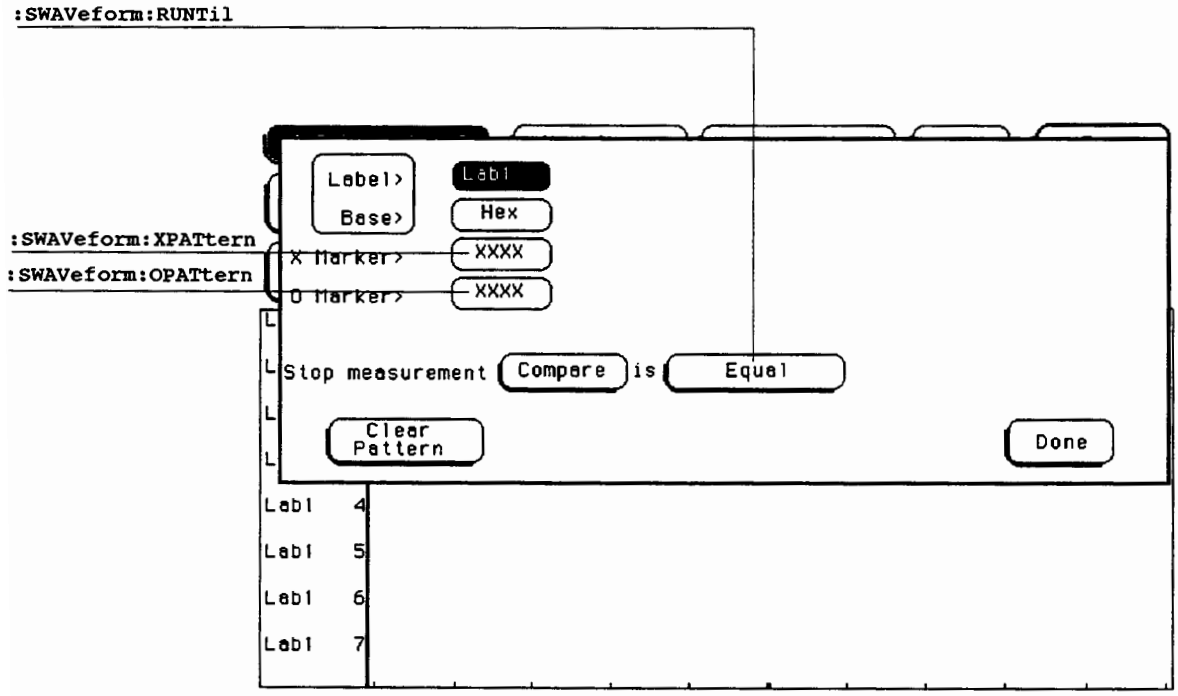
:SWAVeform:OSEarch

:SWAVeform:XSEarch

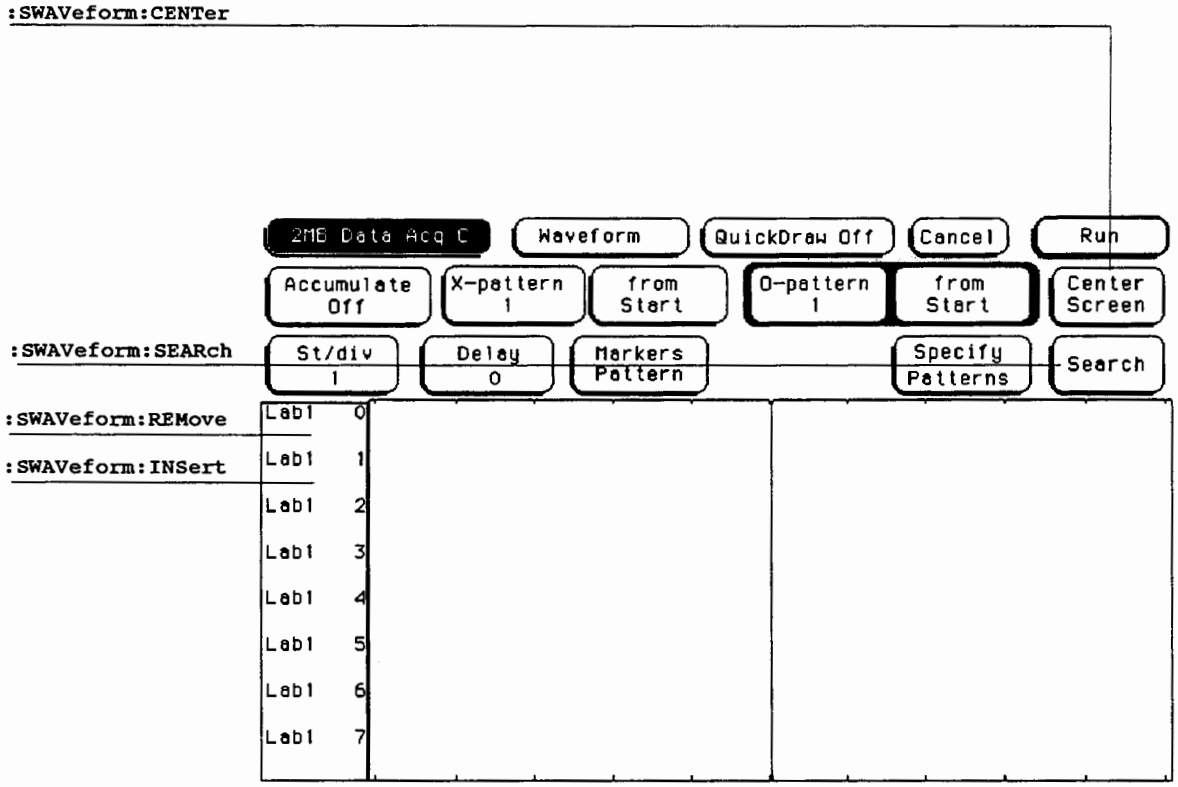


State Waveform

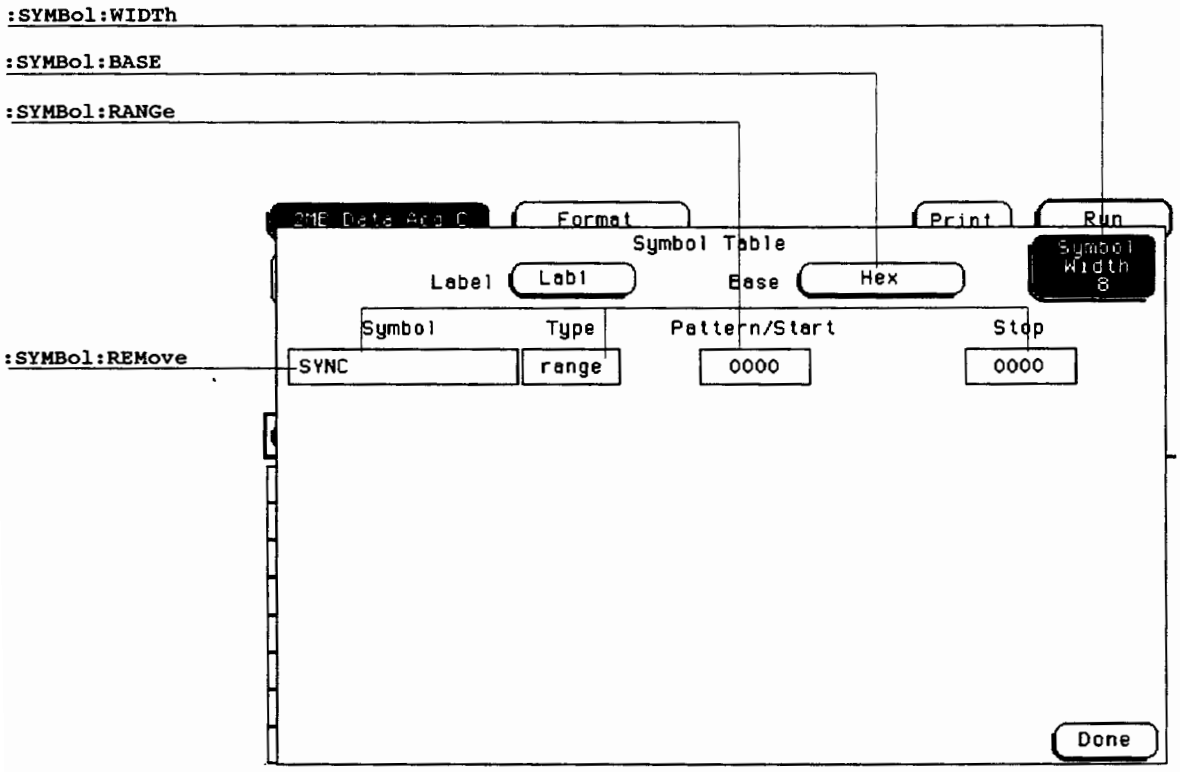
Compare Memory Enabled



State Waveform

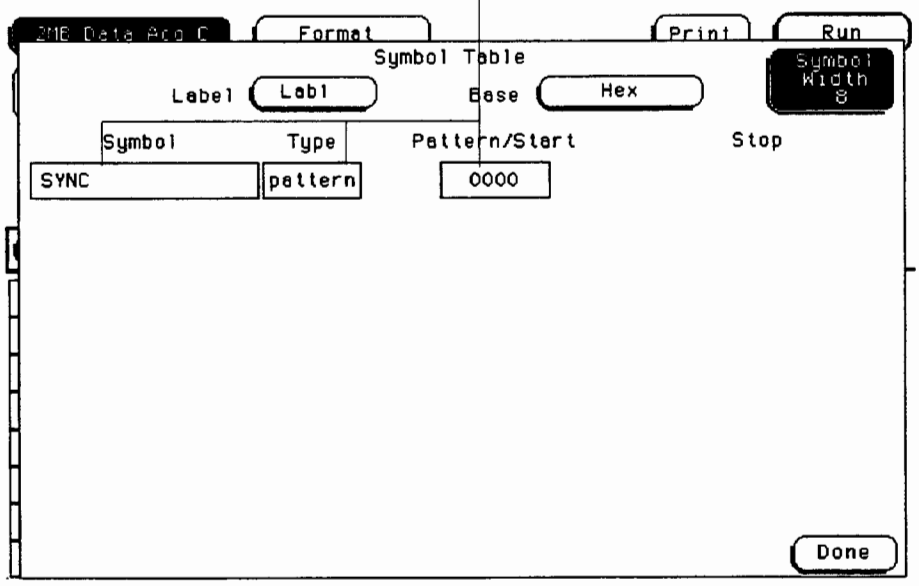


Symbol

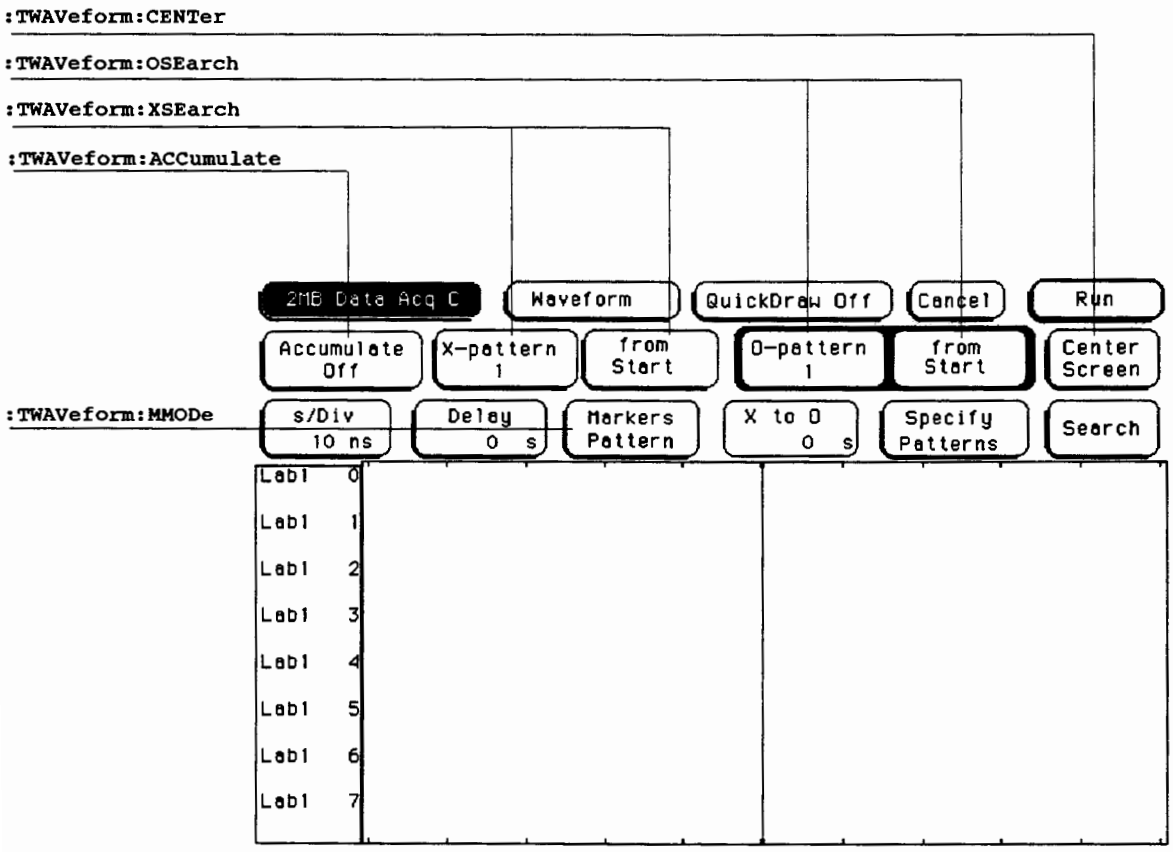

:SYMBol:WIDTh
:SYMBol:BASE
:SYMBol:RANGe
:SYMBol:REMove
Format
Print
Run
Symbol Table
Label
Labl
Base
Hex
Symbol Width 8
Symbol
Type
Pattern/Start
Stop
SYNC
range
0000
0000
Done

Symbol

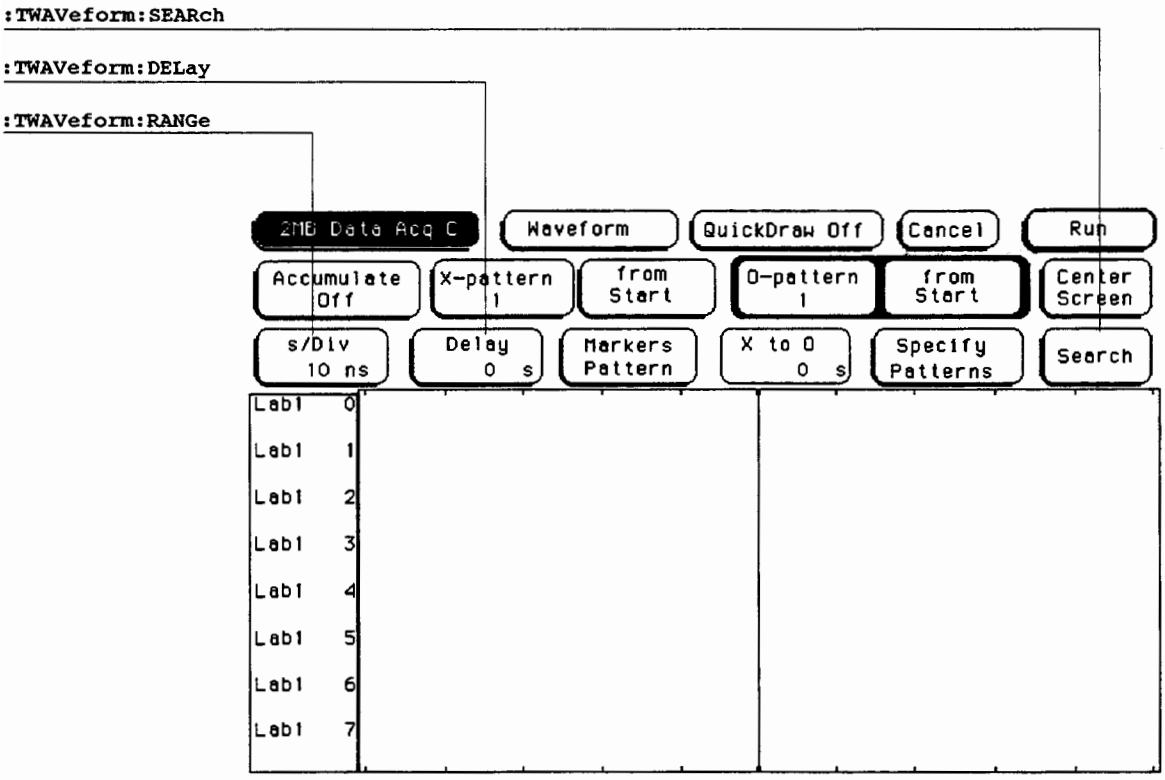
:SYMBol:PATTern



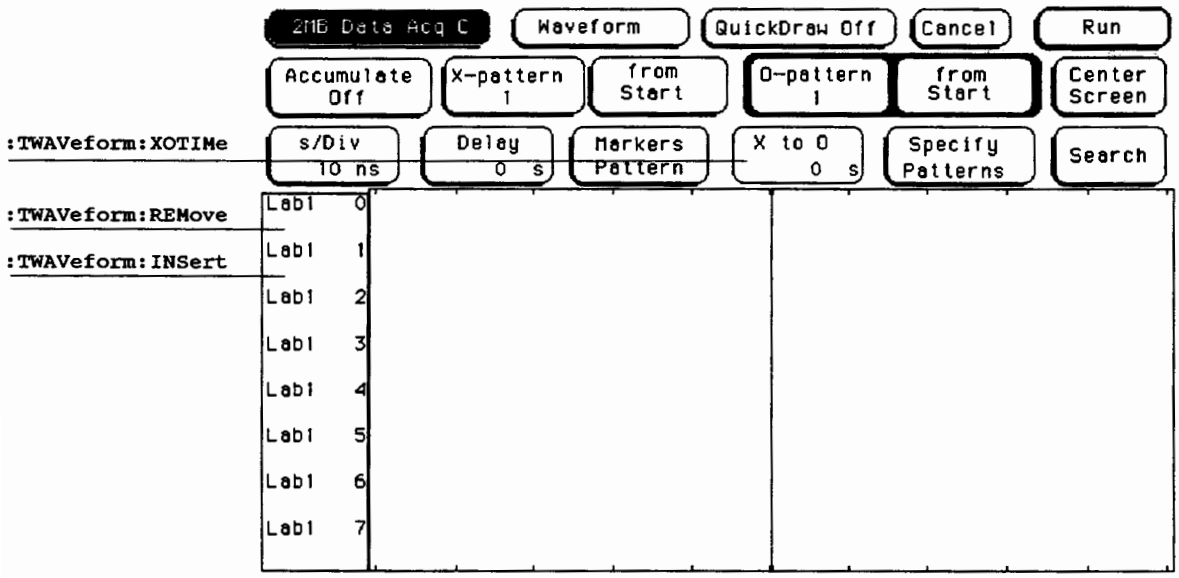
Timing Waveform



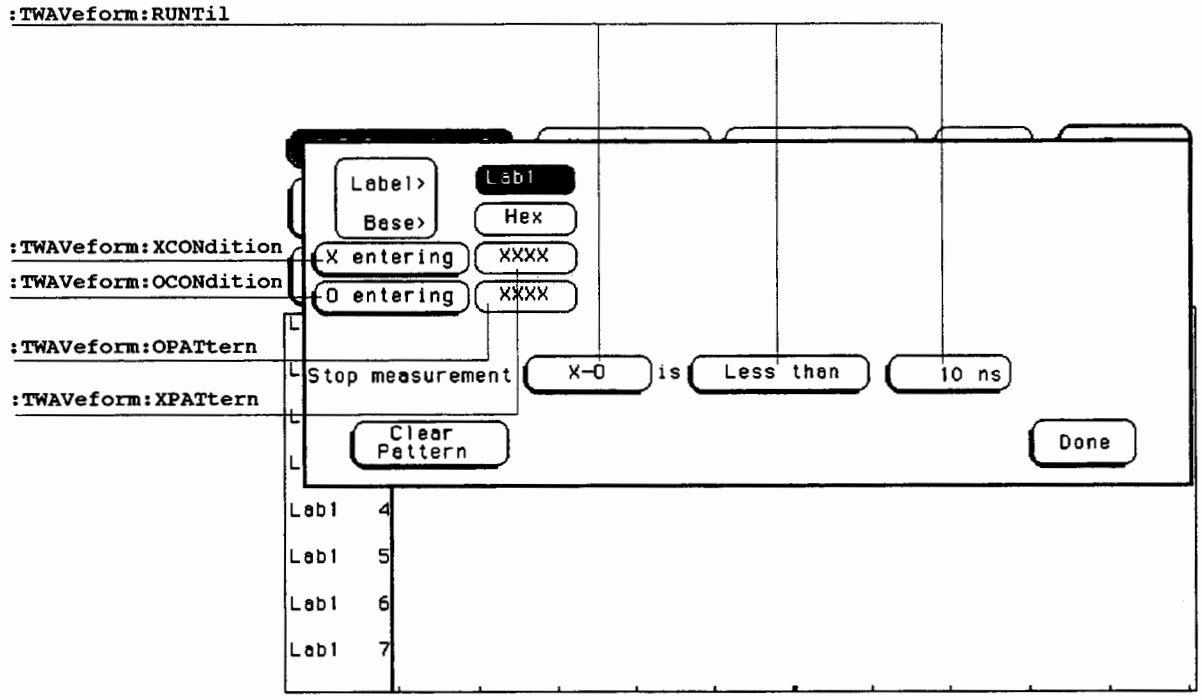
Timing Waveform



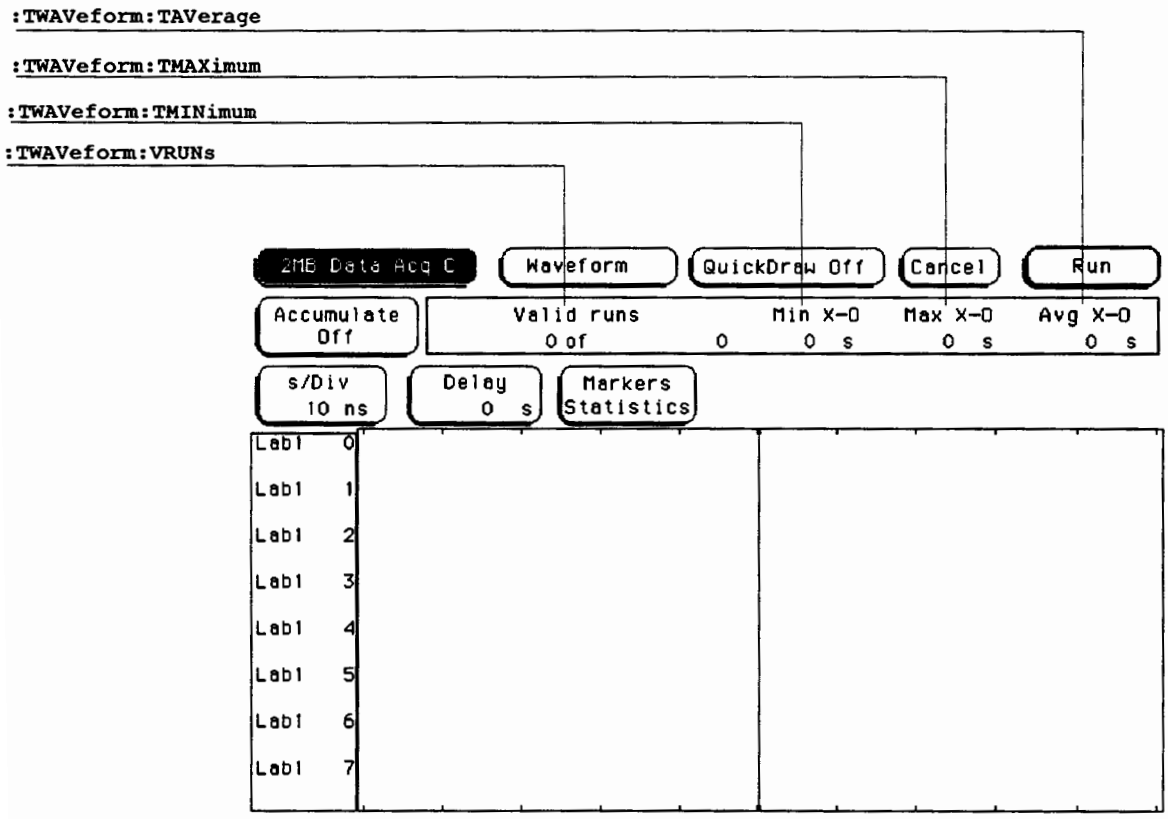
Timing Waveform



Timing Waveform



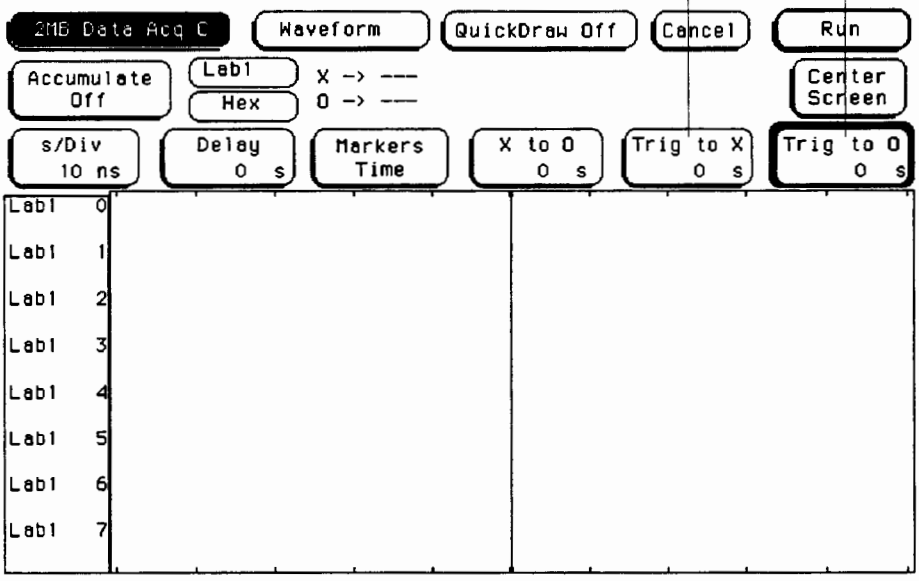
Timing Waveform



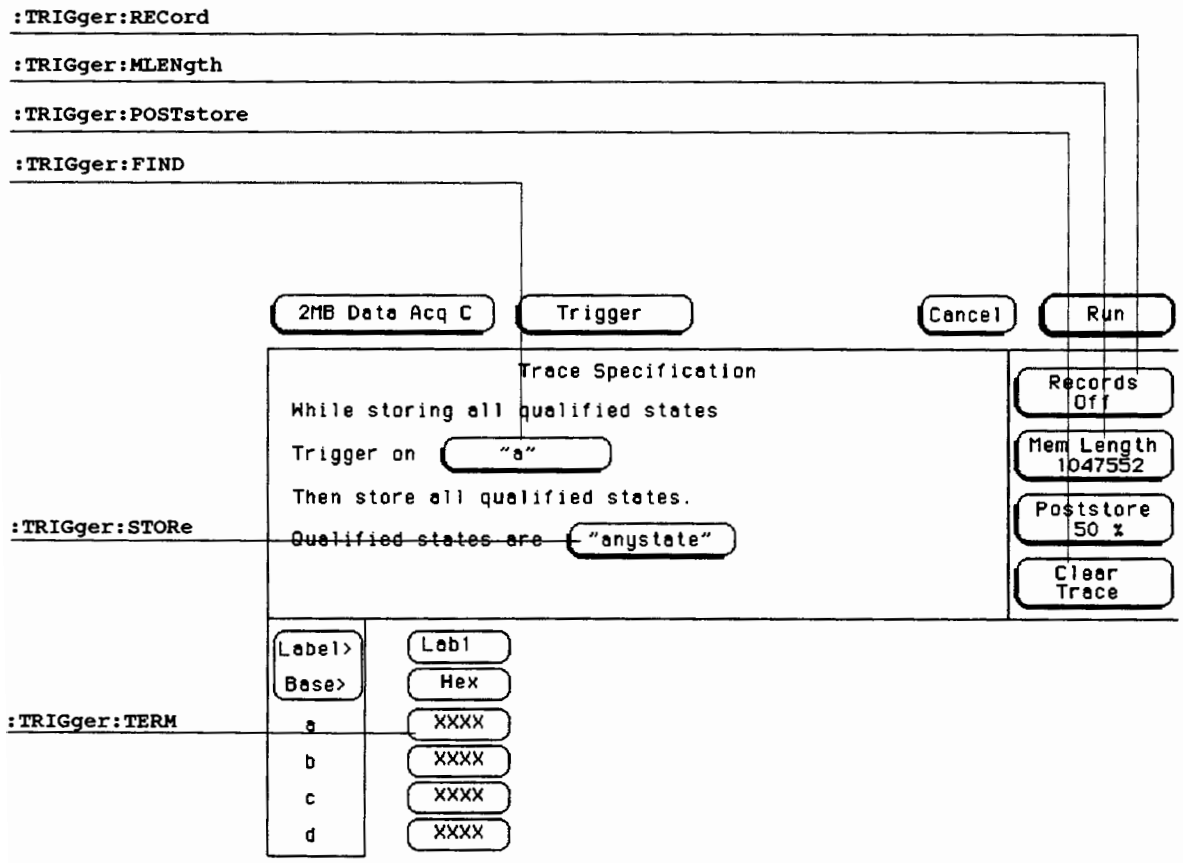
Timing Waveform

:TWAVeform:OTIMe

:TWAVeform:XTIMe



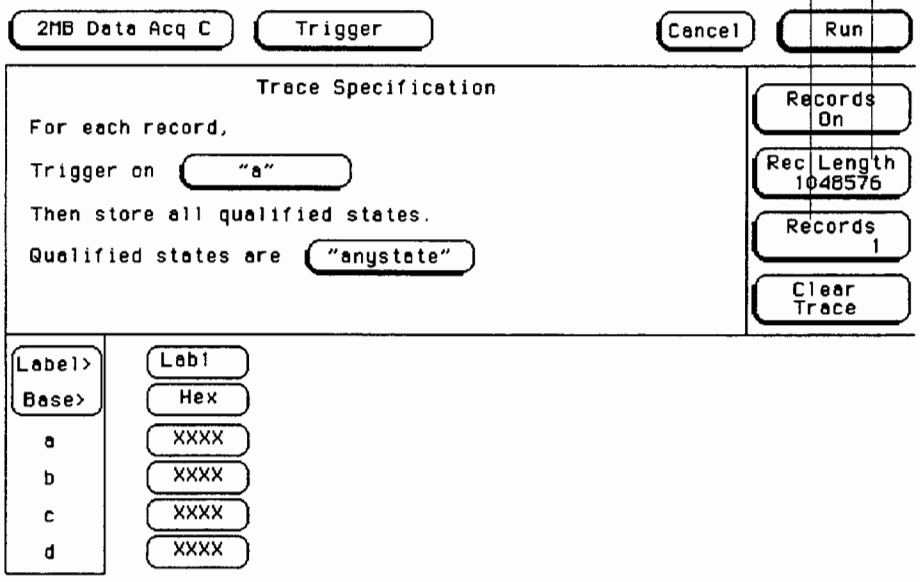
Trigger


:TRIGger:RECord
:TRIGger:MLENgth
:TRIGger:POSTstore
:TRIGger:FIND
2MB Data Acq C
Trigger
Cancel
Run
Trace Specification
While storing all qualified states
Trigger on
"a"
Then store all qualified states.
:TRIGger:STORe
Qualified states are
"anystate"
Records Off
Mem Length 1047552
Poststore 50 %
Clear Trace
Label>
Base>
Lab1
Hex
:TRIGger:TERM
a
b
c
d
XXXX
XXXX
XXXX
XXXX

Trigger

:TRIGger:RLENgth

:TRIGger:NUMRec



Index

A
ACCumulate, 7–6, 8–7, 9–5, 14–18, 14–24
ACCumulate command/query, 9–5
ACCumulate?, 7–6, 8–7, 9–5

B
BASE, 11–5, 14–22

C
CARDcage query, 1–4
CENTer, 7–6, 8–8, 9–5, 14–21, 14–24
channel
  assignment, 3–7
CHARt, 9–4
  ACCumulate, 9–5
  accumulate mode, 9–5
  ACCumulate?, 9–5
  CENTer, 9–5
  HAXis, 9–6 to 9–7
  HAXis?, 9–7
  VAXis, 9–8
  VAXis?, 9–8
chart display, 9–2
Chart Menu, 14–3
CLEar, 10–6
clock, 2–7 to 2–8, 3–5
  master, 3–9
  slave, 3–11
  voltage threshold, 3–6
clock mode, 3–5
CLOCk?, 3–5
CMASk, 10–6
CMASk?, 10–6
CMEMory, 2–3, 14–2
CMEMory?, 2–4
COLumn, 5–6, 6–8, 14–10, 14–14
COLumn?, 5–6, 6–8
command
  ACCumulate, 7–6, 8–7, 9–5
  BASE, 11–5
  CENTer, 7–6, 8–8, 9–5
  CLEar, 10–6
  CLOCk, 3–5
  CMASk, 10–6
  CMEMory, 2–3
  COLumn, 5–6, 6–8
  COPY, 10–7
  CTHReshold, 3–6
  DATA, 10–7
  DELay, 7–7, 8–8
  FIND, 4–5
  HAXis, 9–6
  INSert, 7–8, 8–9 to 8–10, 8–16
  LABel, 3–7
  LINE, 5–8, 6–10, 10–10, 10–12
  MASTer, 3–9
  MENU, 1–5, 10–11
  MESE, 1–13
  MLENgth, 4–6
  MMODe, 5–9, 6–10, 7–8, 8–11
  NAME, 2–4
  NUMRec, 4–13
  OCONdition, 6–11, 8–11
  OPATtern, 5–10, 6–12, 6–21, 7–9, 8–12
  OSEarch, 5–11, 6–13, 6–22, 7–10, 8–13
  OTIMe, 6–15, 8–14 to 8–15
  PATTern, 11–5
  POSTstore, 4–14
  PRINt, 1–8
  RANGe, 7–12, 8–16, 10–11, 11–6
  RECord, 4–15, 5–12
  REMove, 3–10, 5–13, 6–15, 7–12, 8–17, 11–7
  RLENgth, 4–16
  RMODe, 1–7
  RUNTil, 6–16, 7–13, 8–18, 10–13
  SEARch, 5–13, 6–17, 7–14, 8–19
  SELect, 1–2, 1–6
  SETHold, 3–10
  SLAVe, 3–11
  SLISt, 5–5, 6–8
  STARt, 1–6
  STOP, 1–7
  STORe, 4–17
  SYMBol, 11–4
  SYSTem:PRINt, 1–8
  SYSWidth, 2–5
  SYSYem:SETup, 12–14
  TERM, 4–18
  THReshold, 3–10, 3–13
  TYPE, 2–3, 2–7
  VAXis, 9–8
  WIDTh, 11–7
  XPATtern, 5–14, 6–20 to 6–21, 7–14, 8–21 to 8–22
  XSEarch, 5–15, 6–22, 7–15, 8–24
  XTIMe, 6–24, 8–25
Command Set Organization, 1–9 to 1–11
COMPare, 10–5
  channel mask, 10–6
  CLEar, 10–6
  CMASk, 10–6
  CMASk?, 10–6
  COPY, 10–7
  DATA, 10–7 to 10–8
  DATA?, 10–8
  difference, 10–9
  FIND?, 10–9
  LINE, 10–10
  LINE?, 10–10
  MENU, 10–11
  RANGe, 10–11
  RANGe?, 10–12
  RECord, 10–12
  RECord?, 10–12
  RUNTil, 10–13
  RUNTil?, 10–14
  stop condition, 10–13
compare listing
  edit, 10–7
  scroll, 10–10, 10–12
  switching to, 10–11
compare memory, 2–3 to 2–5
  copy, 10–7
Compare Menu, 14–6 to 14–8
COMPare Subsystem, 10–1, 10–3 to 10–15
CONFig, 2–3
  CMEMory, 2–3
  CMEMory?, 2–4
  NAME, 2–4
  NAME?, 2–4
  SYSWidth, 2–5 to 2–6
  SYSWidth?, 2–7
  TYPE, 2–7 to 2–8
  TYPE?, 2–8
Configuration Menu, 14–2
COPY, 10–7
CTHReshold, 3–6, 14–9
CTHReshold?, 3–6

D
DATA, 10–7 to 10–8
Data and Setup Commands, 12–1, 12–3 to 12–15
DATA?, 5–7, 6–9, 10–8

DELay, 7–7, 8–8, 14–18, 14–25
DELay?, 7–7, 8–8

E
edit
  compare listing, 10–7
example
  ASIC testing, 13–12 to 13–14
  Imaging, 13–7 to 13–8
  Lotus Spreadsheet, 13–15 to 13–16
  Telecommunications, 13–9 to 13–11
examples
  programming HP16542A, 13–17 to 13–44

F
FIND, 4–5, 14–30
FIND?, 10–9
FORmat, 3–4
  CLOCk, 3–5
  CLOCk?, 3–5
  CTHReshold, 3–6
  CTHReshold?, 3–6
  LABel, 3–7 to 3–8
  LABel?, 3–8
  MASTer, 3–9
  MASTer?, 3–9
  REMove, 3–10
  SETHold, 3–10
  SETHold?, 3–11
  SLAVe, 3–11 to 3–12
  SLAVe?, 3–12
  THReshold, 3–13
  THReshold?, 3–13
Format Menu, 14–9
FORmat Subsystem, 3–1, 3–3 to 3–13

G
GPIB example
  National Instruments, 13–60 to 13–65

H
HAXis, 9–6 to 9–7
HAXis command/query, 9–6 to 9–7
HAXis?, 9–7
HP-IB example
  Hewlett-Packard, 13–54 to 13–59
HP-IB Interface, 13–45
HP-UX example programs, 13–46 to 13–53

I
INSert, 7–8, 8–9, 14–21, 14–26

L
LABel, 3–7 to 3–8, 14–9
  delete, 3–10, 5–13, 6–15
  name assignment, 3–7, 5–6
  state waveforms, 7–8
LABel?, 3–8
line, 5–8, 6–10, 10–10, 14–10, 14–14 to 14–15
  data value, 5–7
  number, 5–12, 6–9 to 6–10, 6–14, 6–23, 7–11
LINE?, 5–8, 6–10, 10–10
LISt, 5–5
  COLumn, 5–6
  COLumn?, 5–6
  DATA?, 5–7
  LINE, 5–8
  LINE?, 5–8
  MMODe, 5–9
  MMODe?, 5–9
  OPATtern, 5–10
  OPATtern?, 5–10
  OSEarch, 5–11
  OSEarch?, 5–11
  OSTate?, 5–12, 7–11
  RECord, 5–12
  REMove, 5–13
  SEARch, 5–13
  XPATtern, 5–14
  XPATtern?, 5–14
  XSEarch, 5–15
  XSEarch?, 5–16
  XSTate?, 5–16, 7–16
listing
  scrolling, 5–8
Listing Menu, 14–10 to 14–13

M
MACHine Subsystem, 2–1, 2–3 to 2–8
marker
  mode, 5–9, 6–10, 7–8, 8–10
  O, 6–15
  pattern, 6–12
  X pattern, 6–21
MASTer, 3–9, 14–9
MASTer?, 3–9
maximum time
  X-O markers, 6–18, 8–20
memory
  depth, 4–6, 4–8
  width and depth, 2–5 to 2–6
MENU, 1–5, 10–11
MESE command/query, 1–13
MESR query, 1–14
minimum time
  X-O markers, 6–19, 8–20
MLENgth, 4–6 to 4–12, 14–30
MLENgth?, 4–7
MMEMory Subsystem, 1–8
MMODe, 5–9, 6–10, 7–8, 8–10, 14–18, 14–24
MMODe?, 5–9, 6–11, 7–9
Module Status Reporting, 1–12

N
NAME, 2–4, 14–2 to 14–3
NAME?, 2–4
NUMRec, 4–13, 14–31
NUMRec?, 4–13

O
OCONdition, 6–11, 8–11
OCONdition?, 6–12, 8–11
OPATtern, 5–10, 6–12, 7–9, 8–12, 14–20, 14–27
OPATtern?, 5–10, 6–13, 7–10, 8–12
OSEarch, 5–11, 6–13, 7–10, 8–13, 14–19, 14–24
OSEarch?, 5–11, 6–14, 8–14
OSTate?, 5–12, 6–14, 7–11
OTIMe, 6–15, 8–14
OTIMe?, 6–15, 8–14

P
pattern, 6–12, 11–5
  O marker, 7–9
  recognizer term, 4–18, 5–10, 5–14, 7–9, 7–14, 8–12, 8–22
plot
  state versus label, 9–6

polarity, 3–7
POSTstore, 4–14, 14–30
  percentage, 4–14
POSTstore?, 4–15
Programing, 1–1, 1–3 to 1–15
programming examples
  using C, 13–2

Q

Query
  ACCumulate?, 7–6, 8–7, 9–5
  CARDcage, 1–4
  CLOCk, 3–5
  CLOCk?, 3–5
  CMASk, 10–6
  CMASk?, 10–6
  CMEMory?, 2–4
  COLumn, 5–6, 6–8
  COLumn?, 5–6, 6–8
  CTHReshold?, 3–6
  DATA?, 5–7, 6–9, 10–8
  DELay, 7–7
  DELay?, 7–7, 8–8
  ERRor, 1–7
  FIND?, 4–6
  HAXis?, 9–7
  LABel, 3–8
  LABel?, 3–8
  LINE, 5–8
  LINE?, 5–8, 6–10, 10–10
  MASTer?, 3–9
  MENU, 1–5
  MESE, 1–13
  MESR, 1–14
  MLENgth?, 4–7
  MMODe, 5–9
  MMODe?, 5–9, 6–11, 7–8, 8–11
  NAME, 2–4
  NAME?, 2–4
  NUMRec?, 4–13
  OCONdition?, 6–12, 8–11
  OPATtern, 5–10, 6–13, 7–10
  OPATtern?, 5–10, 6–13, 7–10, 8–12
  OSEarch, 5–11
  OSEarch?, 5–11, 6–14, 7–11, 8–14
  OSTate, 6–14
  OSTate?, 5–12, 6–14, 7–11
  OTIMe?, 6–15, 8–14
  POSTstore?, 4–15
  PRINt, 1–8
  RANGe, 7–12
  RANGe?, 7–12, 8–17, 10–11
  RECord?, 4–15, 5–13, 10–12
  RLENgth?, 4–16
  RMODe, 1–7
  RUNTil?, 6–17, 7–13, 8–18, 10–14
  SETHold?, 3–11
  SLAVe, 3–12
  SLAVe?, 3–12
  SPERiod?, 8–19
  STORe, 4–18
  STORe?, 4–18
  SYSTem:DATA?, 12–5
  SYSTem:ERRor, 1–7
  SYSTem:PRINt, 1–8
  SYSWidth?, 2–7
  TAVerage?, 6–18, 8–20
  TERM, 4–19
  TERM?, 4–19
  THReshold, 3–6, 3–11
  THReshold?, 3–13
  TMAXimum?, 6–18, 8–20
  TMINimum?, 6–19, 8–20
  TYPE, 2–8
  TYPE?, 2–8
  VAXis?, 9–8
  VRUNs?, 6–19, 8–21
  XCONdition, 6–20, 8–22
  XCONdition?, 6–20, 8–22
  XOTime?, 6–20, 8–22
  XPATtern?, 5–14, 6–21, 7–15, 8–23
  XSEarch, 6–23
  XSEarch?, 5–16, 6–23, 7–16, 8–24
  XSTate, 5–16, 6–23, 7–16
  XSTate?, 5–16, 6–23, 7–16
  XTIMe, 8–25
  XTIMe?, 6–24

R

RANGe, 7–12, 8–17, 10–11, 11–6, 14–18, 14–22, 14–25
RANGe?, 7–12, 8–17, 10–12
record, 4–15, 5–12, 10–12, 14–30
  length, 4–16
  mode, 4–15
  number defined, 4–13
  scrolling, 5–12
RECord?, 4–15, 5–13, 10–12
REMove, 3–10, 5–13, 6–15, 7–12, 8–17, 11–7, 14–11, 14–21, 14–26 to 14–27
RLENgth, 4–16, 14–31
RLENgth?, 4–16
RMODe, 1–7
run until, 6–16, 7–13, 8–18, 10–13
RUNTil, 6–16, 7–13, 8–18, 10–13
RUNTil?, 6–17, 7–13, 8–18, 10–14

S

SCHart selector, 9–4
SCHart Subsystem, 9–1, 9–3 to 9–8
search, 5–13, 6–17, 7–14, 8–19, 14–10, 14–14, 14–25
  O marker, 5–11, 6–13, 7–10, 8–13
  X marker, 5–15, 6–22, 7–15
search criteria
  X marker, 8–24
SELect, 1–6
SELect command, 1–2
selector
  CHARt, 9–4
  COMPare, 10–5
  CONFig, 2–3
  FORMat, 3–4
  SWAVeform, 7–5
  SYMBol, 11–4
  TRIGger, 4–5
  TWAVeform, 8–7
  WLISt, 6–8
SETHold, 3–10, 14–9
SETHold?, 3–11
SFORmat Subsystem, 13–1, 13–3 to 13–65
SLAVe, 3–11 to 3–12
SLAVe?, 3–12
SLISt Subsystem, 5–1, 5–3 to 5–16, 6–1, 6–3 to 6–24
SPERiod, 8–19
SPERiod query, 8–19
SPERiod?, 8–19
STARt, 1–6
STATe, 2–7
  delay, 7–7
  insert waveforms, 7–8
State Waveform Menu, 14–18 to 14–21
states
  per division, 7–12
STOP, 1–7
stop critera, 6–16, 7–13, 8–18

STORe, 4–17, 14–30
  qualifier, 4–17
STORe command/query, 4–17
STORe?, 4–18
Subsystem
  COMPare, 10–2
  FORmat, 3–1, 3–3 to 3–13
  MACHine, 2–2
  SCHart, 9–2
  SLISt, 5–1, 5–3 to 5–16, 6–1, 6–3 to 6–24
  SWAVeform, 7–1, 7–3 to 7–17
  SYMBol, 11–1, 11–3 to 11–7
  TRIGger, 4–1, 4–3 to 4–19
  TWAVeform, 8–1, 8–3 to 8–25
SWAVeform, 7–5
  ACCumulate, 7–6
  ACCumulate?, 7–6
  CENTer, 7–6
  DELay, 7–7
  DELay?, 7–7
  INSert, 7–8
  MMODe, 7–8
  MMODe?, 7–9
  OPATtern, 7–9
  OPATtern?, 7–10
  OSEarch, 7–10
  OSEarch?, 7–11
  RANGe, 7–12
  RANGe?, 7–12
  REMove, 7–12
  RUNTil, 7–13
  RUNTil?, 7–13
  SEARch, 7–14
  XPATtern, 7–14
  XPATtern?, 7–15
  XSEarch, 7–15
  XSEarch?, 7–16
SWAVeform Subsystem, 7–1, 7–3 to 7–17
SYMBol, 11–4
  BASE, 11–5
  delete, 11–7
  PATTern, 11–5
  RANGe, 11–6
  REMove, 11–7
  WIDTh, 11–7
Symbol Menu, 14–22 to 14–23
SYMBol Subsystem, 11–1, 11–3 to 11–7
Syntax Diagram
  TWAVeform Subsystem, 8–3
SYSTem:DATA, 12–4 to 12–5
SYSTem:DATA?, 12–5
SYSTem:PRINt, 1–8
SYStem:SETup, 12–14
SYSWidth, 2–5 to 2–6, 14–2
SYSWidth?, 2–7

T

TAVerage?, 6–18, 8–20
TERM, 4–18 to 4–19, 14–30
TERM?, 4–19
THReshold, 3–13, 14–9
  channel, 3–13
THReshold command/query, 3–6
THReshold?, 3–13
time
  delay, 8–8
  O marker, 6–15
  seconds per division, 8–16
  setup and hold, 3–10
  X marker, 6–24
  X-O markers, 6–20, 8–22
time average
  X-O markers, 6–18, 8–20
Time Listing Menu, 14–15 to 14–17
TIMing, 2–7
  O marker placement, 6–11, 8–11
  O marker position, 8–14
  O marker search, 8–13
timing waveform
  delete, 8–17
  insert, 8–9
Timing Waveform Menu, 14–24 to 14–29
TMAXimum?, 6–18, 8–20
TMINimum?, 6–19, 8–20
TRIGger, 4–5
  FIND, 4–5
  MLENgth, 4–6 to 4–12
  MLENgth?, 4–7
  NUMRec, 4–13
  NUMRec?, 4–13
  POSTstore, 4–14
  POSTstore?, 4–15
  qualifier, 4–5
  RECord, 4–15
  RECord?, 4–15
  RLENgth, 4–16
  RLENgth?, 4–16
  STORe, 4–17
  STORe?, 4–18
  TERM, 4–18 to 4–19
  TERM?, 4–19
Trigger Menu, 14–30 to 14–31
TRIGger Subsystem, 4–1, 4–3 to 4–19
TWAVeform, 8–7
  ACCumulate, 8–7
  ACCumulate?, 8–7
  CENTer, 8–8
  DELay, 8–8
  DELay?, 8–8
  INSert, 8–9
  MMODe, 8–10
  MMODe?, 8–11
  OCONdition, 8–11
  OCONdition?, 8–11
  OPATtern, 8–12
  OPATtern?, 8–12
  OSEarch, 8–13
  OSEarch?, 8–14
  OTIMe, 8–14
  OTIMe?, 8–14
  RANGe, 8–16
  RANGe?, 8–17
  REMove, 8–17
  RUNTil, 8–18
  RUNTil?, 8–18
  SEARch, 8–19
  SPERiod, 8–19
  SPERiod?, 8–19
  TAVerage?, 8–20
  TMAXimum?, 8–20
  TMINimum?, 8–20
  VRUNs?, 8–21
  XCONdition?, 8–22
  XOTime?, 8–22
  XPATtern, 8–21 to 8–23
  XPATtern?, 8–23
  XSEarch, 8–24
  XSEarch?, 8–24
TWAVeform Subsystem, 8–1, 8–3 to 8–25
TYPE, 2–7 to 2–8, 14–2
TYPE?, 2–8

V

valid runs, 6–19, 8–21
VAXis, 9–8
VAXis command/query, 9–8

VAXis?, 9–8
vertical axis
  plot, 9–8
voltage
  channel threshold, 3–13
VRUNs?, 6–19, 8–21

W
waveform
  accumulate, 8–7
  delete, 7–12
WIDTh, 11–7, 14–22
WLISt, 6–8
  COLumn, 6–8
  COLumn?, 6–8
  DATA?, 6–9
  LINE, 6–10
  LINE?, 6–10
  MMODe, 6–10
  MMODe?, 6–11
  OCONdition, 6–11
  OCONdition?, 6–12
  OPATtern?, 6–13
  OSEarch, 6–13
  OSEarch?, 6–14
  OSTate?, 6–14
  OTIMe, 6–15
  OTIMe?, 6–15
  REMove, 6–15
  RUNTil, 6–16
  RUNTil?, 6–17
  SEARch, 6–17
  TAVerage?, 6–18
  TMAXimum?, 6–18
  TMINimum?, 6–19
  VRUNs?, 6–19
  XCONdition?, 6–20
  XOTime?, 6–20
  XPATtern, 6–20 to 6–21
  XPATtern?, 6–21
  XSEarch, 6–22
  XSTate?, 6–23
  XTIMe, 6–24
  XTIMe?, 6–24

X
X marker
  placement, 6–20, 8–21
XCONdition, 6–20, 8–21
XCONdition?, 6–20, 8–22
XOTime?, 6–20, 8–22
XPATtern, 5–14, 6–21, 7–14, 8–22 to 8–23, 14–10, 14–12, 14–14, 14–20, 14–27
XPATtern?, 5–14, 6–21, 7–15, 8–23
XSEarch, 5–15, 6–22, 7–15, 8–24, 14–10, 14–12, 14–14, 14–19, 14–24
XSEarch?, 5–16, 7–16, 8–24
XSTate?, 5–16, 6–23, 7–16 to 7–17
XTIMe, 6–24
XTIMe command/query, 8–25
XTIMe?, 6–24

© Copyright Hewlett-Packard Company 1992.

All Rights Reserved.

Microsoft® is a US registered trademark of Microsoft Corporation.

Reproduction, adaptation, or translation without prior written permission is prohibited, except as allowed under the copyright laws.

Document Warranty

The information contained in this document is subject to change without notice.

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties or merchantability and fitness for a particular purpose.

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Complete product warranty information is given at the end of this manual.

Safety

This is a Safety Class I instrument (provided with terminal for protective earthing). Before applying power, verify that the correct safety precautions are taken (see the following warnings). In addition, note the external markings on the instrument that are described under "Safety Symbols."

Safety Symbols



Instruction manual symbol: the product is marked with this symbol when it is necessary for you to refer to the instruction manual in order to protect against damage to the product.



Hazardous voltage symbol.



Earth terminal symbol: Used to indicate a circuit common connected to grounded chassis.

WARNING

The Warning sign denotes a hazard. It calls attention to a procedure, practice, or the like, which, if not correctly performed or adhered to, could result in personal injury. Do not proceed beyond a Warning sign until the indicated conditions are fully understood and met.

CAUTION

The Caution sign denotes a hazard. It calls attention to an operating procedure, practice, or the like, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product. Do not proceed beyond a Caution symbol until the indicated conditions are fully understood or met.

Hewlett-Packard
P.O. Box 2197
1900 Garden of the Gods Road
Colorado Springs, CO 80901

Product Warranty

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of one year from date of shipment. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard.

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Hewlett-Packard warrants that its software and firmware designated by Hewlett-Packard for use with an instrument will execute its programming instructions when properly installed on that instrument. Hewlett-Packard does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. Hewlett-Packard specifically disclaims the implied warranties or merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are the buyer's sole and exclusive remedies. Hewlett-Packard shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Assistance

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales Office.

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Institute of Standards and Technology, to the extent allowed by the Institute's calibration facility, and to the calibration facilities of other International Standards Organization members.

About this edition

This is the first edition of the *HP 16542A 2-Mbyte, 100-MHz State/Timing Logic Analyzer Programmer's Reference.*

Publication number
16542-90902

Printed in USA.

Edition dates are as follows:

First edition, September 1992

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by you. The dates on the title page change only when a new edition is published.

A software or firmware code may be printed before the date. This code indicates the version level of the software or firmware of this product at the time the manual or update was issued. Many product updates do not require manual changes; and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

The following list of pages gives the date of the current edition and of any changed pages to that edition.

All pages original edition

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP