Programming Reference

HP 16540A,D and 16541A,D
100 MHz State Analyzer Module

for the HP 16500A Logic Analysis System

HEWLETT PACKARD

Manual Part Number 16540-90909

Printed in the U.S.A. October 1991

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 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. However, warranty service for products installed by Hewlett-Packard and certain other products designated by Hewlett-Packard will be performed at the Buyer's facility at no charge within the Hewlett-Packard service travel area. Outside Hewlett-Packard service travel areas, warranty service will be performed at the Buyer's facility only upon Hewlett-Packard's prior agreement and the Buyer shall pay Hewlett-Packard's round trip travel expenses.

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 and Service 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 Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Safety This product has been designed and tested according to International Safety Requirements. To ensure safe operation and to keep the product safe, the information, cautions, and warnings in this manual must be heeded.

Printing History

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 the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

A software code may be printed before the date; this indicates the version level of the software product at the time of the manual or update was issued. Many product updates and fixes 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.

Edition 1	January 1991	16540-90903
Edition 2	October 1991	16540-90909

List of Effective Pages

The List of Effective Pages gives the date of the current edition and of any pages changed in updates to that edition. Within the manual, any page changed since the last edition will have the date the changes were made printed on the bottom of the page. If an update is incorporated when a new edition of the manual is printed, the change dates are removed from the bottom of the pages and the new edition date is listed in Printing History and on the title page.

Pages	Effective Date
	October 1991

Contents

Chapter 3: WLISt Subsystem

Chapter 4: FORMat Subsystem

Chapter 5: TRACe Subsystem

Chapter 6: LIST Subsystem

Chapter 7: SWAVeform Subsystem

Chapter 8: TWAVeform Subsystem

Appendix A: DATA and SETup Commands

Index

Programming the State/Timing Analyzer 1

Introduction

This manual, combined with the *HP 16500A Programming Reference* manual, provides you with the information needed to program the 100 MHz State Analyzer module. Each module has its own manual to supplement the mainframe manual because not all mainframes will be configured with the same modules.

About This Manual

This manual is organized in nine chapters. The first chapter contains the following information:

General information and instructions to help you get started.

- Mainframe system commands that are frequently used with the state analyzer module.
- 100 MHz State Analyzer command tree.
- Alphabetic command-to-subsystem directory.

Chapters two through eleven contain the subsystem commands for the state analyzer.

Appendix A contains information on the SYSTem:DATA and SYSTem:SETup commands for this module.

Note 

If the analyzer module is configured with HP 16540,41A model cards, the memory depth per channel is 4,096 bytes deep. If the analyzer module is configured with HP 16540,41D model cards, the memory depth per channel is 16,384 bytes deep.

All references in this manual related to memory depth will be for the 16 kBytes deep HP 16540,41D model cards.

Programming the 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 used, 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 state analyzer, you must first "select" it. This directs your commands to the state analyzer.

To select the state analyzer module, use the system command :SELect followed by the numeric reference for the slot location of the state analyzer (1...5 refers to slot A...E respectively). For example, if the state analyzer 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.

Analyzer Program

A typical state 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 Program:

```
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
```

Note 

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 where 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 names analyzer to "STATETEST".
Line 30 specifies an External clock type.
Line 40 assigns a label called DATA, with positive polarity, and the first eight bits to the label.
Line 50 sets the threshold of the first pod 1 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 state 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 Programming Reference* manual.

CARDcage? Query

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 16540A,D master is 40. The identification number for the HP 16541A,D master is 41. 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,-1,40,41,2,2,0,2,5

A returned string of 12,11,-1,41,40,2,2,0,2,5 means that an oscilloscope time base card (ID number 11) is loaded in slot B and the oscilloscope acquisition card (ID number 12) is loaded in slot A. The next slot (C) is empty (-1). Slot D contains the state analyzer module's expander board (ID number 41). Slot E contains the state analyzer module's master board (ID number 40).

The next group of numbers (2,2,0,2,5) 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. The number 5 , located in slot E's position of the string, shows where the controlling software for the module is recognized.

MENU
Command/query

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

Menu identification for the state analyzer is listed below:

- X,0 - Configuration menu
- X,1 - Format menu
- X,2 - Trace menu
- X,3 - Timing Waveform menu
- X,4 - State Waveform menu
- X,5 - Listing menu
- X,6 - Mixed Display menu
- X,7 - Compare menu
- X,8 - Chart menu
- X,10 - Calibration

SELect
Command/query

The SELect command selects which module or intermodule will have parser control. SELect 0 selects the intermodule, SELect 1 through 5 selects modules A through E, respectively. Values -1 and -2 select software options 1 and 2. The SELect query returns the currently selected module.

STARt
Command

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

STOP
Command

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

Note 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 Programming Reference* manual.

RMODe Command/query 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 will be set by this command. The RMODe query returns the current setting.

SYSTem:ERRor? Query 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/query The SYSTem:PRINt command initiates a print of the screen or listing buffer over the current printer communication interface. 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 16540A,D and 16541A,D 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).
- Trace menu (TRACe subsystem).
- Listing menu (LIST subsystem).
- Timing Waveform menu (TWAVeform subsystem).
- State Waveform menu (SWAVeform subsystem).
- Mixed Display menu (WLISt 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 16540A,D and 16541A,D State Analyzer module. The /x/ by the SELect command at the top of the tree represents the slot number where the state analyzer module is installed. The number may range from 1 through 5, representing slots A through E, 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 lowercase 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 16540A,D and 16541A,D Command Tree

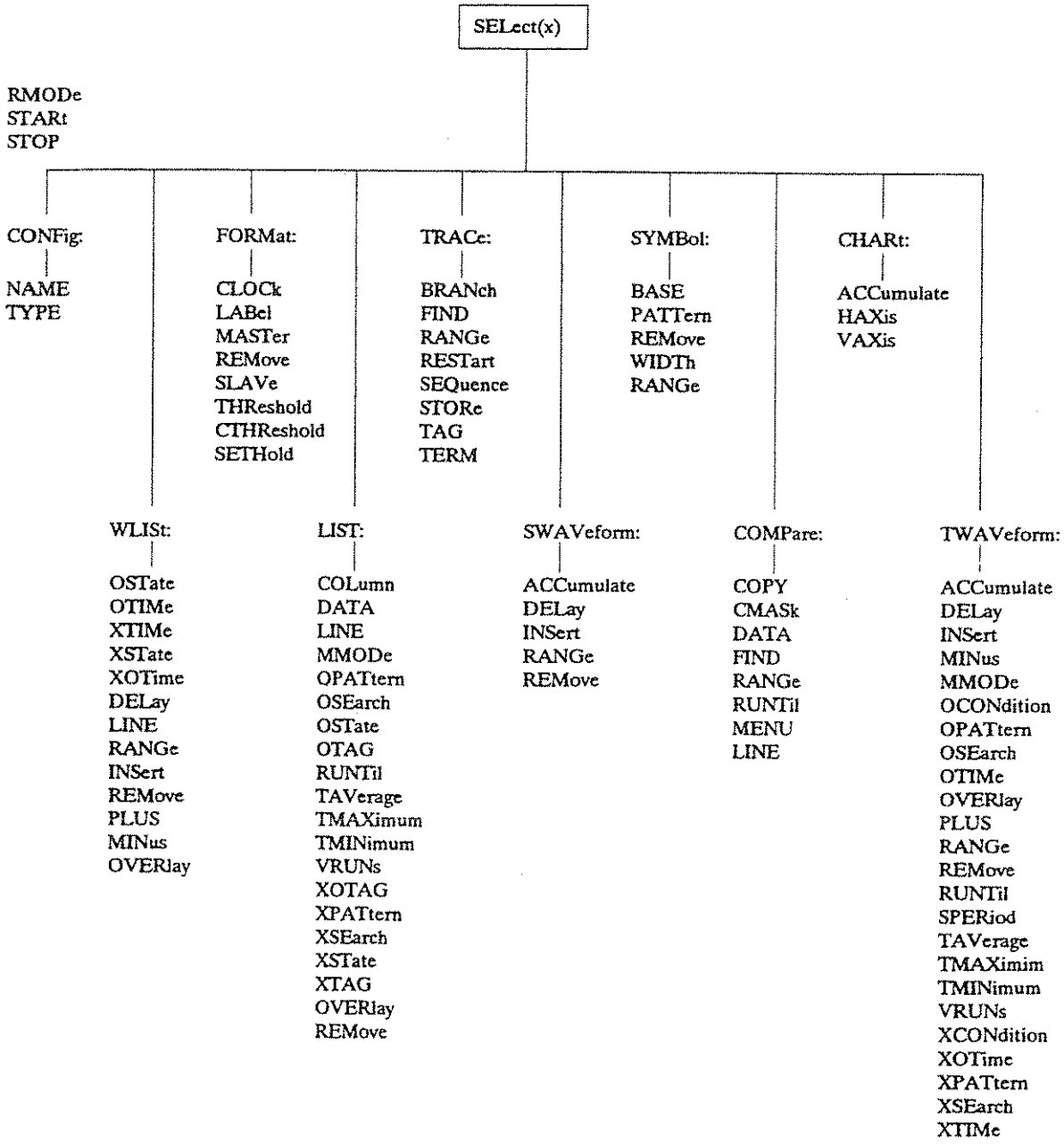


Table 1-2. Alphabetical Command-to-Subsystem Directory

Command	Where used
ACCumulate	CHARt, SWAVeform TWAVeform
BASE	SYMBol
BRANch	TRACe
CLOCk	SFORmat
COLumn	LIST
COPY	COMPare
CMASk	COMPare
CTHReshold	FORMat
DATA	COMPare, LIST
DELay	SWAVeform, TWAVeform, WLISt
FIND	COMPare, TRACe
HAXis	CHARt
INSert	SWAVeform, TWAVeform, WLISt
LABel	FORMat
LINE	LIST, WLISt
MASTer	FORMat
MINus	TWAVeform, WLISt
MMODe	LIST, TWAVeform
NAME	CONFig
OCONdition	TWAVeform
OPATtern	LIST, TWAVeform
OSEarch	LIST,TWAVeform
OSTate	LIST, WLISt
OTAG	LIST
OTIMe	TWAVeform, WLISt
OVERlay	TWAVeform, WLISt, LIST
PATTern	SYMBol
PLUS	TWAVeform, WLISt
RANGe	COMPare, TRACe, SWAVeform, SYMBol, TWAVeform, WLISt
REMove	FORMat, SWAVeform, SYMBol, TWAVeform, WLISt, LIST
RESTart	TRACe
RUNTil	COMPare, LIST, TWAVeform
SEQuence	TRACe
SETHold	FORMat
SLAVe	FORMat
SPERiod	TWAVeform
STORe	TRACe
TAG	TRACe
TAVerage	LIST, TWAVeform
TERM	TRACe
THReshold	FORMat
TMAXimum	LIST, TWAVeform
TMINimum	LIST, TWAVeform
TYPE	CONFig
VAXis	CHARt
VRUNs	LIST, TWAVeform
WIDTh	SYMBol
XCONdition	TWAVeform
XOTAG	LIST
XOTime	TWAVeform, WLISt
XPATtern	LIST, TWAVeform
XSEarch	LIST, TWAVeform
XSTate	WLISt, LIST
XTAG	LIST
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>).

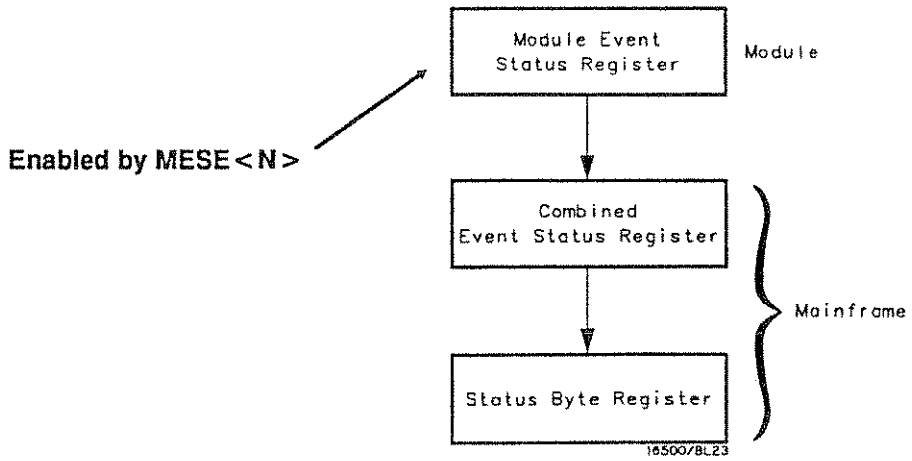


Figure 1-1. Module Status Reporting

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 16540A,D and 16541A,D.

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/query

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....5 refers to the module in slot A...E). 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.

Command Syntax: :MESE<N> <enable_mask>

where:

<N> ::= {1|2|3|4|5} number of slot in which the module resides
<enable_mask> ::= integer from 0 to 255

Example:

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

MESE<N>

The MESE query returns the current setting.

Query Syntax: :MESE<N>?

Returned Format: [:MESE<N>]<enable_mask><NL>

Example:
```
10 OUTPUT XXX;":MESE5?"
20 ENTER XXX; Mes
30 PRINT Mes
40 END
```

Table 1-3. Module Event Status Enable Register

Module Event Status Enable Register (A "1" enables the MESR bit)		
Bit	**Weight**	**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 satisified
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

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

Note 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...5 refers to the module in slot A...E, respectively.

Query Syntax: :MESR<N>?

Returned Format: [MESR<N>]<status><NL>

where:

<N> ::= {1|2|3|4|5} number of slot in which the module resides
<status> ::= integer from 0 to 255

Example:

```
10 OUTPUT XXX;":MESR5?"
20 ENTER XXX; Mer
30 PRINT Mer
40 END
```

MESR<N>

Table 1-4. Module Event Status Register

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	1 = Could not place X/O Markers 0 = Placed X/O Markers
2	4	1 = Real trigger found 0 = Real trigger not found
1	2	1 = Run until satisified 0 = Run until not satisified
0	1	1 = Measurement complete 0 = Measurement not complete

CONFig Subsystem 2

Introduction

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

- NAME
- TYPE

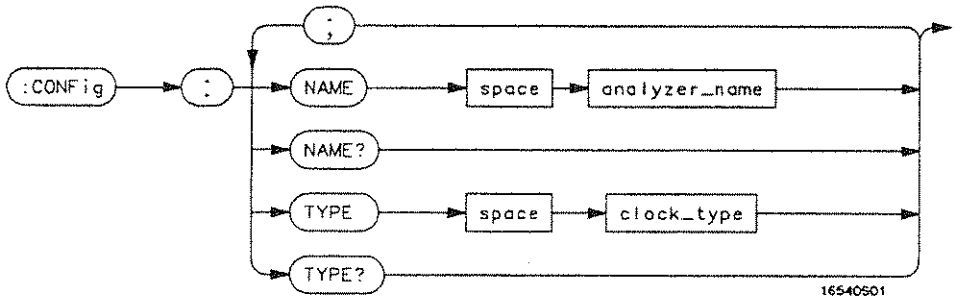


Figure 2-1. Config Subsystem Syntax Diagram

analyzer_name = *string of up to 10 alphanumeric characters*
clock_type = {TIMing|STATe}

CONFig

selector

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

Command Syntax: :CONFig

Example:

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

NAME command/query

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

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

Command Syntax: :CONFig:NAME <analyzer_name>

where:

<analyzer_name> ::= string of up to 10 alphanumeric characters

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

Query Syntax: :CONFig:NAME?

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

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":CONFig:NAME?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TYPE command/query

The TYPE command specifies what type of clock the analyzer will use. The analyzer clock types are external and internal. In addition to selecting the clock type, the TYPE command is used to select software options.

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

Command Syntax: :CONFig:TYPE <clock type>

where:

<clock type> ::= {TIMing | STATe}

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

Query Syntax: :CONFig:TYPE?

Returned Format: [:CONFig:TYPE] <clock type><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX;":CONFig:TYPE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

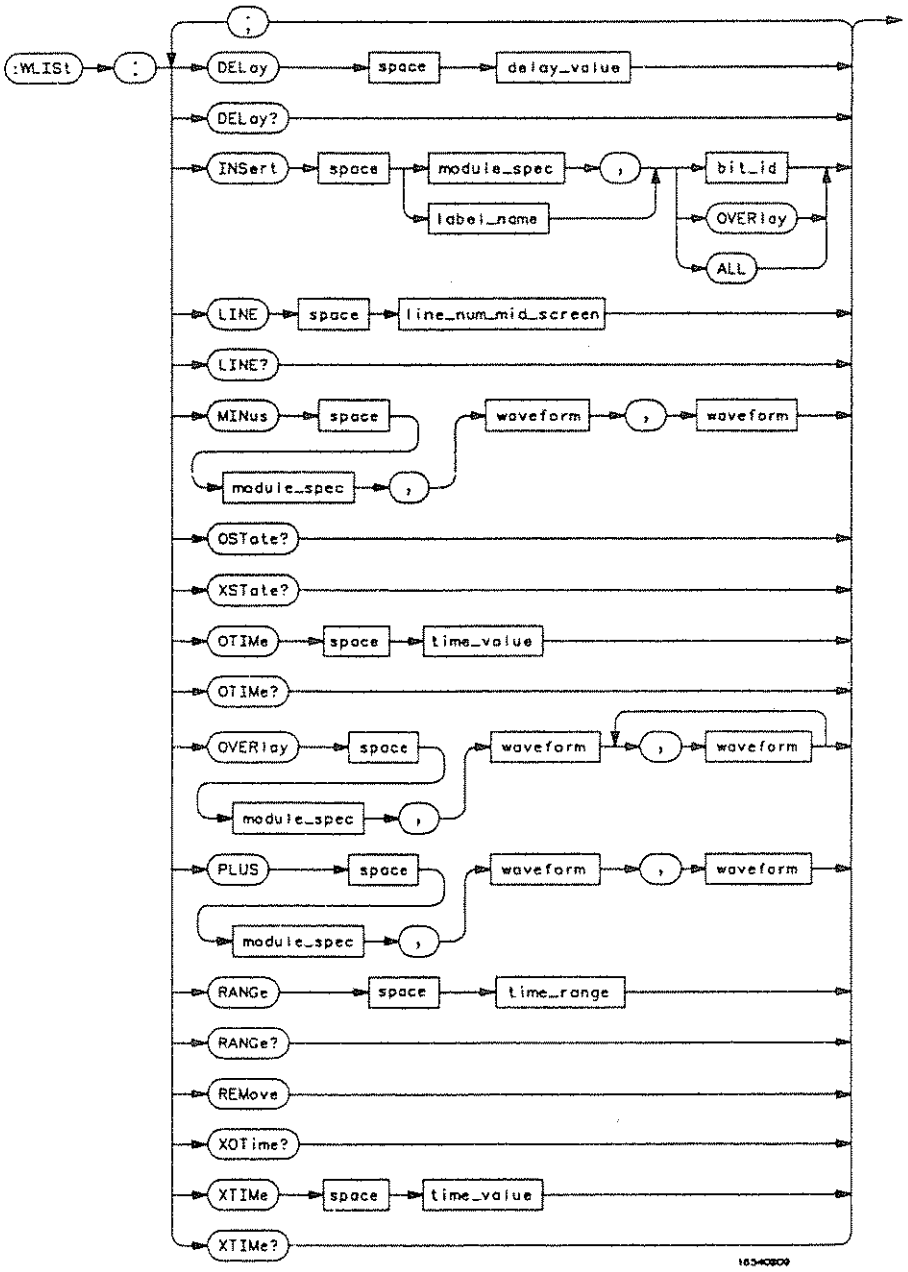
WLISt Subsystem 3

Introduction

Commands in the WLISt (Waveforms/Listing) subsystem control the X and O marker placement on the waveforms portion of the Mixed Display. The XSTate and OSTate queries return what states the X and O markers are on. Since the markers can only be placed on the timing waveforms, the queries return the state memory location for the marked pattern.

Note In order to have the Mixed Display menu available, the analyzer must be in the State mode with the **Count** field in the Trace menu set to **Time**. Waveforms can be brought in from other modules as long as they are time correlatable.

- DELay
- LINE
- RANGe
- OSTate
- XSTate
- OTIMe
- XTIMe
- XOTime
- INSert
- REMove
- PLUS
- MINus
- OVERlay



3-1. WLISt Subsystem Syntax Diagram

time_value = *real number*
line_num_mid_screen = *integer from -16384 to +16384*
delay_value = *real number between -2500 s and +2500 s*
time_range = *real number between 100 ns and 1 ks*
module_spec = *{1|2|3|4|5}(slot where timebase card is installed)*
bit_id = *integer from 0 to 31*
waveform = *string containing <acquisition_spec>{1|2}*
acquisition_spec = *{A|B|C|D|E}(slot where acquisition card is located)*
label_name = *string of up to 6 alphanumeric characters*

WLISt selector

The WLISt (Waveforms/Listing) selector is used as a part of a compound header to access the settings normally found in the Mixed Display menu. Because the WLISt command is a subsystem level command, it will always appear as the first element of a compound header.

Note In order to have the Mixed Display menu available, the analyzer must be in the State mode with the **Count** field in the Trace menu set to **Time**. Waveforms can be brought in from other modules as long as they are time correlatable.

Command Syntax: :WLISt

Example:
```
OUTPUT XXX;":WLIST:XTIME 40.0E-6"
```

OSTate

query

The OSTate query returns the state where the O Marker is positioned. If data is not valid, the query returns 32767.

Query Syntax: :WLISt:OSTate?

Returned Format: [:WLISt:OSTate] <state_num> <NL>

where:

<state_num> ::= integer

Example:

```
10 DIM So$[100]
20 OUTPUT XXX;":WLIST:OSTATE?"
30 ENTER XXX;So$
40 PRINT So$
50 END
```

XSTate query

The XSTate query returns the state where the X Marker is positioned. If data is not valid, the query returns 32767.

Query Syntax: :WLISt:XSTate?

Example:
```
OUTPUT XXX,":WLISt:XSTATE?
```

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

where:

<state_num> ::= integer

Example:
```
10 DIM Sx$[100]
20 OUTPUT XXX;":WLIST:XSTATE?"
30 ENTER XXX;Sx$
40 PRINT Sx$
50 END
```

OTIMe command/query

The OTIMe command positions the O Marker on the timing waveforms in the mixed display menu. If the data is not valid, the command performs no action.

The OTIMe query returns the O Marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :WLISt:OTIMe <time_value>

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX;":WLIST:OTIME 40.0E-6"
```

Query Syntax: :WLISt:OTIMe?

Returned Format: [:WLISt:OTIMe] <time_value><NL>

Example:
```
10 DIM To$[100]
20 OUTPUT XXX;":WLIST:OTIME?"
30 ENTER XXX;To$
40 PRINT To$
50 END
```

XTIMe command/query

The XTIMe command positions the X Marker on the timing waveforms in the mixed display. If the data is not valid, the command performs no action.

The XTIMe query returns the X Marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :WLISt:XTIMe <time_value>

where:

<time_value> ::= real number

Example:

```
OUTPUT XXX;":WLIST:XTIME 40.0E-6"
```

Query Syntax: :WLISt:XTIMe?

Returned Format: [:WLISt:XTIMe] <time_value><NL>

Example:

```
10 DIM Tx$[100]
20 OUTPUT XXX;":WLIST:XTIME?"
30 ENTER XXX;Tx$
40 PRINT Tx$
50 END
```

LINE

command/query

The LINE command allows you to scroll the state analyzer listing vertically. The command specifies the state line number relative to the trigger that the analyzer will be highlighted at center screen.

The LINE query returns the line number for the state currently in the box at center screen.

Command Syntax: :WLISt:LINE <line_num_mid_screen>

where:

<line_num_mid_screen> ::= integer from -16384 to +16384

Example:

```
OUTPUT XXX;":WLISt:LINE 0"
```

Query Syntax: :WLISt:LINE?

Returned Format: [:WLISt:LINE] <line_num_mid_screen><NL>

Example:

```
10 DIM Ln$[100]
20 OUTPUT XXX;":WLIST:LINE?"
30 ENTER XXX;Ln$
40 PRINT Ln$
50 END
```

DELay command/query

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 -2500 s to +2500 s.

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

Command Syntax: :WLISt:DELay <delay_value>

where:

<delay_value> ::= real number between -2500 s and +2500 s

Example:
```
OUTPUT XXX;":WLISt:DELAY 100E-6"
```

Query Syntax: :WLISt:DELay?

Returned Format: [:WLISt:DELay] <time_value><NL>

Example:
```
10 DIM D1$ [100]
20 OUTPUT XXX;":WLIST:DELAY?"
30 ENTER XXX; D1$
40 PRINT D1$
50 END
```

RANGe command/query

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 100 ns to 1 ks.

The RANGe query returns the current full-screen time.

Command Syntax: :WLISt:RANGe <time_value>

where:

<time_range> ::= real number between 100 ns and 1 ks

Example:
```
OUTPUT XXX;":WLISt:RANGE 100E-9"
```

Query Syntax: :WLISt:RANGe?

Returned Format: [:WLISt:RANGe] <time_value><NL>

Example:
```
10 DIM Rg$ [100]
20 OUTPUT XXX;":WLIST:RANGE?"
30 ENTER XXX; Rg$
40 PRINT Rg$
50 END
```

REMove command

The REMove command deletes all waveforms from the display.

Command Syntax: :WLISt:REMove

Example:

```
OUTPUT XXX;":WLIST:REMOVE"
```

PLUS

command

The PLUS command inserts time-correlated A+B oscilloscope waveforms on the screen. The first parameter is the module specifier where the oscilloscope module resides. 1...5 refers to slots A...E. The next two parameters specify which waveforms will be added to each other.

Note PLUS is only available for oscilloscope waveforms.

Command Syntax: :WLISt:PLUS <module_spec>,<waveform>,<waveform>

where:

<module_spec> ::= {1|2|3|4|5} (slot where timebase card is located)
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E} (slot where acquisition card is located)

Example:

```
OUTPUT XXX; ":WLIST:PLUS 2,'A1','A2'"
```

OVERlay command

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 syntax specifies which slot contains the oscilloscope time base card. The next parameters are the labels of the waveforms that are to be overlaid.

Note OVERlay is only available for oscilloscope waveforms.

Command Syntax: :WLISt:OVERlay <module_number>, <label>[, <label>]...

where:

<module_spec> ::= {1|2|3|4|5} (slot where timebase card is located)
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E} (slot where acquisition card is located)

Example:
```
OUTPUT XXX;":WLIST:OVERLAY 4, 'C1','C2'"
```

INSert command

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 high speed timing modules 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 the oscilloscope or high speed timing modules, the optional first parameter must be used. 1...5 corresponds to modules A...E. 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.

INSert

Command Syntax: :WLISt:INSert [<module_spec>,]<label_name>[,{<bit_id>|OVERlay|ALL}]

where:

<module_spec> ::= {1|2|3|4|5} (slot where timebase card is located)
<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= integer from 0 to 31

Example:

```
OUTPUT XXX;":WLIST:INSERT 3, 'WAVE',10"
```

Inserting Oscilloscope Waveforms

Inserting a waveform from an oscilloscope to the timing waveforms display:

Command Syntax: :WLISt:INSert <module_spec>,<label_name>

where:

<module_spec> ::= {1|2|3|4|5} (slot in which timebase card is installed)
<label_name> ::= string of one alpha and one numeric character

Example:

```
OUTPUT XXX;":WLIST:INSERT 5, 'C1'"
```

MINus command

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. 1...5 refers to slots A...E. The next two parameters specify which waveforms will be subtracted from each other.

Note MINus is only available for oscilloscope waveforms.

Command Syntax: :WLISt:MINus <module_spec>,<waveform>,<waveform>

where:

<module_spec> ::= {1|2|3|4|5} (slot where timebase card is located)
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E} (slot where acquisition card is located)

Example:
```
OUTPUT XXX; ":WLIST:MINUS 2,'A1','A2'"
```

XOTime query

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

Query Syntax: :WLISt:XOTime?

Returned Format: [:WLISt:XOTime] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Xot$ [100]
20 OUTPUT XXX;":WLIST:XOTIME?"
30 ENTER XXX; Xot$
40 PRINT Xot$
50 END
```

FORMat Subsystem 4

Introduction

The FORMat subsystem contains the commands available for the Format menu in the HP 16540A,D and 16541A,D logic analyzer module. These commands are listed below:

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

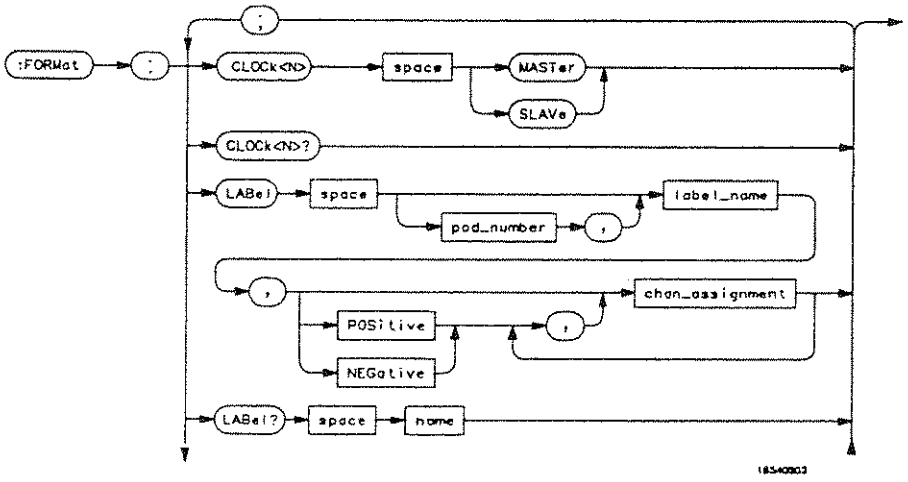


Figure 4-1. FORMat Subsystem Syntax Diagram

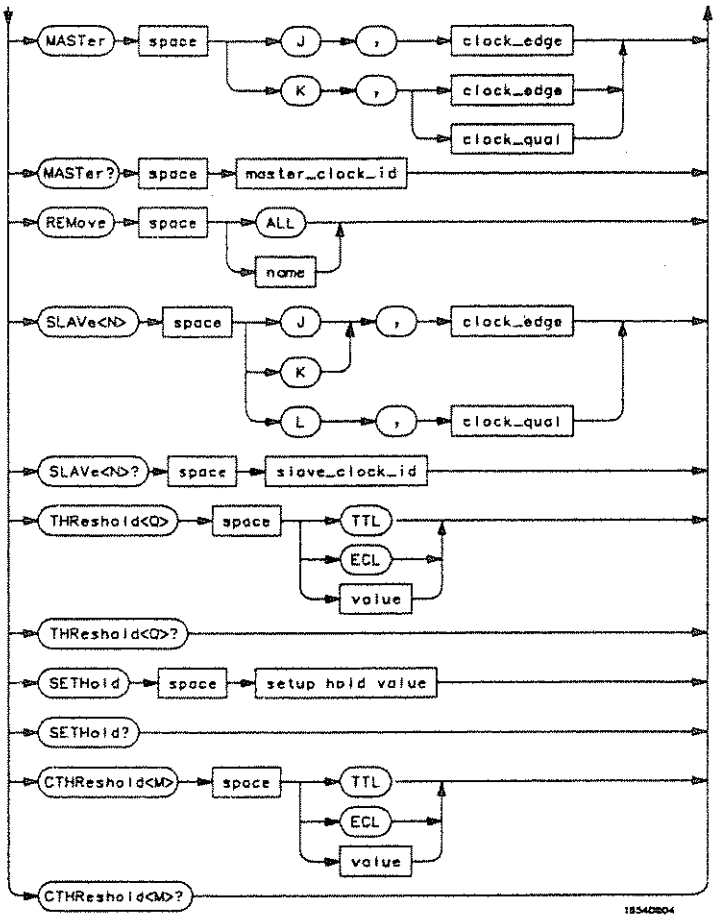


Figure 4-1. FORMat Subsystem Systax Diagram (cont.)

<N> = {*1* | *2* | *3* | *4*} **<M>** = {*0* | *1* | *2* | *3* | *4*} **<Q>** = {*0*|*1*|*2*|...|*12*}
setup hold value = *integer from 0 to 3*
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* | *L*}
clock_edge = {*OFF* | *RISing* | *FALLing* | *BOTH*}
clock_qual = {*OFF* | *LOW* | *HIGH*}
value = *voltage (real number) -3.5 V to +5.0 V*
pod number = *integer from 0 to 12*

FORMat selector

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 preceeds any Format subsystem commands.

Command Syntax: :FORMat

Example:

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

CLOCk command/query

The clock command selects the clocking mode (master or slave clocking) for a given expansion card. When the MASTer option is specified, all three pods on the expansion card sample data on the master clock. When the SLAVe option is specified, all three pods on the expansion card sample 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 query returns the current clocking mode for a given expansion card.

Command Syntax: :FORMat:CLOCk<N> <clock_mode>

where:

<N> ::= {1|2|3|4}
<clock_mode> ::= {MASTer | SLAVe}

Example:
```
OUTPUT XXX;":FORMAT:CLOCK2 SLAVE"
```

Query Syntax: :FORMat:CLOCk<N>?

Returned Format: [:FORMat:CLOCK<N>] <clock_mode><NL>

Example:
```
10 DIM String$ [100]
20 OUTPUT XXX; ":FORMAT:CLOCK?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

LABel command/query

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 will be created. If more than 60 labels are specified, the last label (bottom label) in the Format menu will be 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.

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

Because pods contain 16 channels, the format 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 touchscreen.

A label can not have a total of more than 32 channels assigned to it.

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 first parameter. Numbers are always returned in decimal format.

LABel

Command Syntax: :FORMat:LABel [<pod number>,] 'name'[, {<polarity> | <assignment>}]...

where:

<pod number> ::= (integer from 0 - 12)
'name' ::= string of up to 6 case sensitive alphanumeric characters
<polarity> ::= {POSitive | NEGative}
<assignment> ::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples:

```
OUTPUT XXX;":FORMAT:LABEL 4,'STAT', POSITIVE, 65535,127,40312"
OUTPUT XXX;":FORMAT:LABEL 4, 'SIG 1', 64, 12, 0, 20, NEGATIVE"
OUTPUT XXX;":FORMAT:LABEL 4,'ADDR', NEG, #B0011110010101010"
```

Query Syntax: :FORMat:LABel? <name>

Returned Format: [:FORMat:LABel] <name>,<polarity>[, <assignment>]...<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":FORMAT:LABEL? 'DATA'"
30 ENTER XXX String$
40 PRINT String$
50 END
```

MASTer command/query

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 clock specification for the analyzer requires two commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed. Level specifications (LOW or HIGH) are ANDed. Only the K clock may contain a level specification.

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

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

Command Syntax: :FORMat:MASTer <clock_id>,<clock_spec>

where:

<clock_id> ::= {J or 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 Syntax: :FORMat:MASTer? <clock_id>

Returned Format: [:FORMat:MASTer] <clock_id>,<clock_spec><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":FORMAT:MASTER? J"
30 ENTER XXX String$
40 PRINT String$
50 END
```

REMove command

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

Command Syntax: :FORMat:REMove {<name>|ALL}

where:

<name> ::= string of up to 6 alphanumeric characters

Examples:
```
OUTPUT XXX;":FORMAT:REMOVE 'A'"
OUTPUT XXX;":FORMAT:REMOVE ALL"
```

SLAVe command/query

The SLAVe clock command allows you to specify a slave clock for a given expansion card. When the slave clock is selected, it becomes the clock for all three pods on the expansion card. Each command deals with only one clock (J, K or L), therefore, a complete clock specification for an expander card requires three commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed. Level specifications (LOW or HIGH) are ANDed. The J and K clocks may only contain edge specifications and the L clock can only contain a level specification

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

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).

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

where:

<N> ::= {1|2|3|4}
<clock_id> ::= {J|K|L}
<clock_spec> ::= {OFF|RISing|FALLing|BOTH} for J and K clocks
::= {OFF|LOW|HIGH} for L clock

Example:
```
OUTPUT XXX;":FORMAT:SLAVE2 J, RISING"
```

SLAVe

Query Syntax: FORMat:SLAVe<N>? <clock_id>

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

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":FORMAT:SLAVE<N>? J"
30 ENTER XXX String$
40 PRINT String$
50 END
```

THReshold

command/query

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 12" selects expansion pods as they appear in the Format menu from left to right.

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

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

where:

<M> ::= pod number {0|1|2|3|4 ——|12} where 0 = master and 1-12 = 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 Syntax: :FORMat:THReshold<M>?

Returned Format: [:FORMat:THReshold<M>] <value><NL>

Example:
```
10 DIM Value$ [100]
20 OUTPUT XXX;":FORMAT:THRESHOLD4?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

CTHReshold command/query

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 query returns the current threshold for a given clock.

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

where:

<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 ::= default value of +1.6V
ECL ::= default value of -1.3V

Example:
```
OUTPUT XXX;":FORMAT:CTHRESHOLD1 4.0"
```

Query Syntax: :FORMat:CTHReshold <M>?

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

Example:
```
10 DIM Value$ [100]
20 OUTPUT XXX;":FORMAT:CTHRESHOLD1?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

SETHold command/query

The SETHold command allows you to select one of three setup and holdtime combinations.

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

Command Syntax: :FORMat:SETHold <L>

where:

<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 (good for query only)

Example:

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

Query Syntax: :FORMat:SETHold?

Returned Format: [:FORMat:SETHold] <value> <NL>

Example:

```
10 DIM Value$ [100]
20 OUTPUT XXX;":FORMAT:SETHold?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

TRACe Subsystem 5

Introduction

The TRACe subsystem contains the commands available for the Trace menu in the HP 16540A,D and 16541A,D logic analyzer module. The TRACe subsystem commands are listed below:

- BRANch
- FIND
- RANGe
- RESTart
- SEQuence
- STORe
- TAG
- TERM

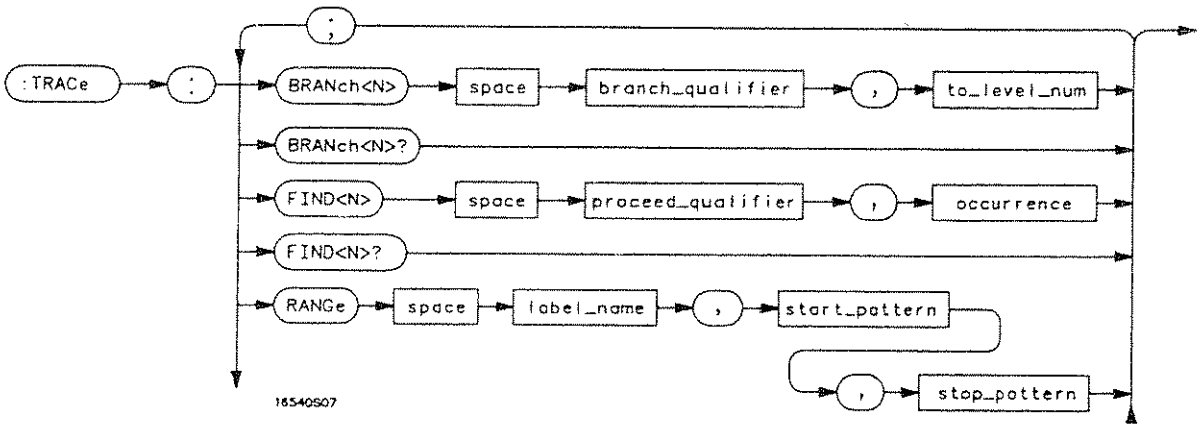


Figure 5-1. TRACe Subsystem Syntax Diagram

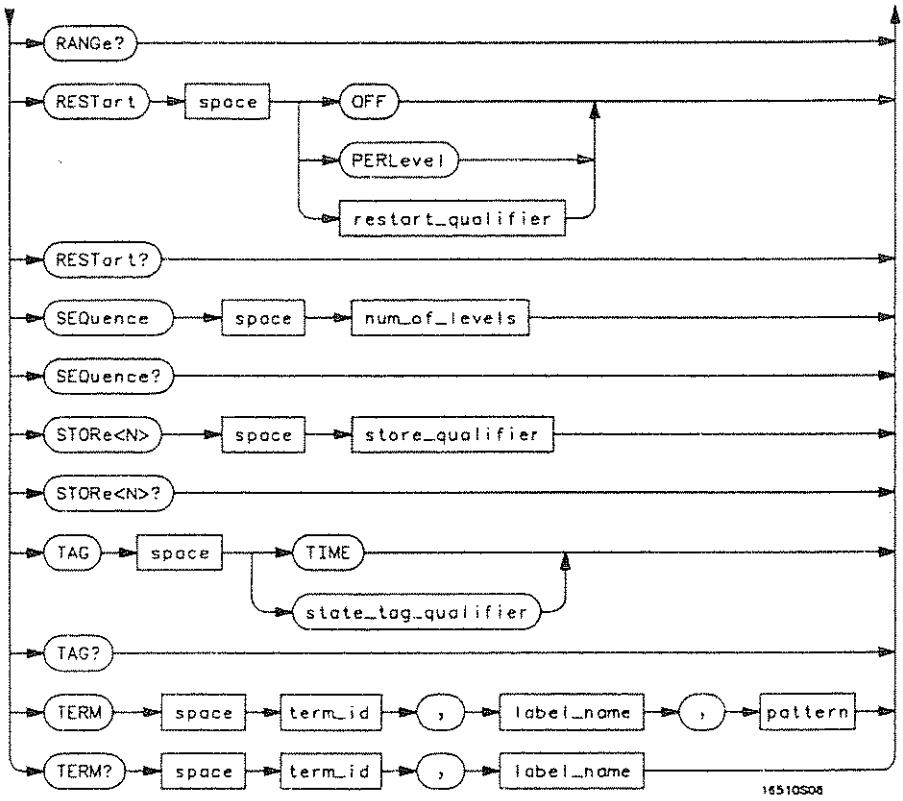


Figure 5-1. TRACe Subsystem Syntax Diagram (cont.)

N = *an integer from 1 to <num_of_levels>*
branch_qualifier = *<qualifier>*
to_level_num = *integer from 1 to <num_of_levels> -1*
proceed_qualifier = *<qualifier>*
occurrence = *number from 1 to 65535*
label_name = *string of up to 6 alphanumeric characters*
start_pattern = "{#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_pattern = "{#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}... }"
restart_qualifier = *<qualifier>*
num_of_levels = *integer from 2 to 5*
store_qualifier = *<qualifier>*
state_tag_qualifier = *<qualifier>*
term_id = {A|B|C|D}
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}... }"
qualifier = { *ANYState* | *NOSTate* | *<any_term>* | *(expression)*}
any_term = {A|B|C|D|*INRange*|*OUTRange*|*NOTA*|*NOTB*|*NOTC*|*NOTD*}
expression = {*<expression 1>[OR <expression1>]* | *<expression 1>[AND <expression1>]*}
expression1 = {*any_term[OR any_term]*... | *any_term[AND any_term]*... }

TRACe selector

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

Command Syntax: :TRACe

Example:
```
OUTPUT XXX;":TRACE:TAG TIME"
```

BRANch command/query

The BRANch command defines the branch qualifier for a given sequence level. When this branch qualifier is matched, it causes the sequencer to jump to the specified sequence level.

Note "RESTART PERLEVEL" must have been invoked for this command to have an effect (see the RESTart command in this chapter).

The terms used by the branch qualifier (A through D) are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Within the limitations shown by the syntax definitions, complex expressions may be formed using the AND and OR operators. Expressions are limited to what you can manually enter through the touch screen. Regarding parentheses, the syntax definitions on the next page show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. For example, the following two statements are both correct and have the same meaning. Notice that the conventional rules for precedence are not followed.

```
OUTPUT XXX;":TRACE:BRANCH1 (C OR D) AND (NOTA OR NOTC), 1"
OUTPUT XXX;":TRACE:BRANCH1 ((C OR D) AND (NOTA OR NOTC)), 1"
```

Figure 5-2 shows a complex expression as seen on the Format display.

Note It is illegal to branch across the trigger level. The values for <N> and <to_level_num> must both be on or before the trigger level.

The BRANch query returns the current branch qualifier specification for a given sequence level.

BRANch

Command Syntax: :TRACe:BRANch<N> <branch_qualifier>,<to_level_number>

where:

<N> ::= an integer from 1 to <number_of_levels> -1
<to_level_number> ::= integer from 1 to trigger level
<number_of_levels> ::= integer from 2 to the number of existing sequence levels (maximum 5)
<branch_qualifier> ::= { ANYState | NOSTate | <any_term>** |(<expression>**)}
<any_term> ::= {A|B|C|D|INRange*|OUTRange*|NOTA|NOTB|NOTC|NOTD}
<expression> ::= {<expression1>[OR <expression1>] | <expression1>[AND <expression1>]}
<expression 1> ::= {any_term[OR any_term] . . .| any_term[AND any_term] . . . }

* = Range choices only available if range label is available.
** = Only available in State mode.

Examples:
```
OUTPUT XXX;":TRACE:BRANCH1 ANYSTATE, 3"
OUTPUT XXX;":TRACE:BRANCH2 A, 3"
OUTPUT XXX;":TRACE:BRANCH3 ((A OR B) OR NOTC), 1"
```

Query Syntax :TRACe:BRANch<N>?

Returned Format: [:TRACe:BRANch<N>] <branch_qualifier>,<to_level_num><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:BRANCH3?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

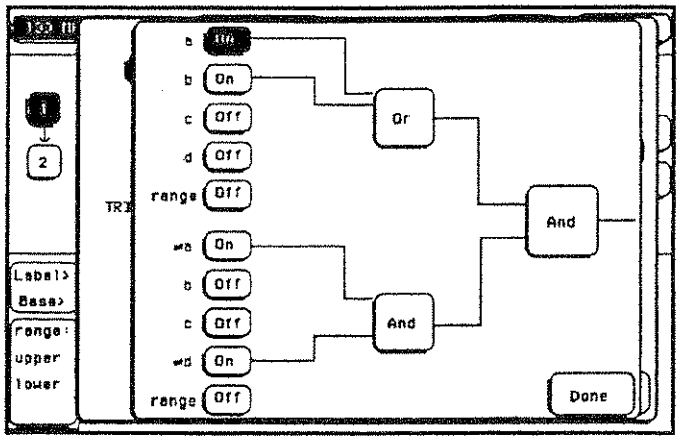


Figure 5-2. Complex Qualifier

Figure 5-2 is a front panel representation of the complex qualifier **(a Or b) And (≠ a And ≠ d)**. The following example would be used to specify this complex qualifier.

```
OUTPUT XXX;":TRACE:BRANCH1 ((A OR B) AND (NOTA AND NOTD)), 2"
```

FIND command/query

The FIND command defines the proceed qualifier for a given sequence level. The qualifier tells the state analyzer when to proceed to the next sequence level. When this proceed qualifier is matched the specified number of times, the sequencer proceeds to the next sequence level. The state that causes the sequencer to switch levels is automatically stored in memory whether it matches the associated store qualifier or not. In the sequence level where the trigger is specified, the FIND command specifies the trigger qualifier (see SEQuence command).

The terms A through D are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command. Expressions are limited to what you could manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed. See figure 5-2 for a detailed example.

The FIND query returns the current proceed qualifier specification for a given sequence level.

Command Syntax: :TRACe:FIND<N> <proceed_qualifier>,<occurrence>

where:

<N>	::= integer from 1 to the number of existing sequence levels -1 (maximum 4)
<occurrence>	::= integer from 1 to 65535
<proceed_qualifier>	::= { ANYState | NOSTate | <any_term>** |(<expression>**)}
<any_term>	::= {A|B|C|D|INRange*|OUTRange*|NOTA|NOTB|NOTC|NOTD}
<expression>	::= {<expression1>[OR <expression1>] | <expression1>[AND <expression1>]}
<expression 1>	::= {any_term[OR any_term] . . . | any_term[AND any_term] . . . }

* = Range choices only available if range label is available.

** = Only available in State mode.

Examples:

```
OUTPUT XXX;":TRACE:FIND1 ANYSTATE, 1"
OUTPUT XXX;":TRACE:FIND2 A, 512"
OUTPUT XXX;":TRACE:FIND3 ((NOTA AND NOTB) OR C), 1"
```

Query Syntax: :TRACe:FIND4?

Returned Format: [:TRACe:FIND<N>] <proceed_qualifier>,<occurrence><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:FIND<N>?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RANGe command/query

The RANGe command allows you to specify a range recognizer term. Because a range can only be defined across one label and, because a label must contain 32 or less bits, the value of the start pattern or stop pattern will be between (2^{32})-1 and 0.

Note Because a label can only be defined across a maximum of two pods, a range term is only available across a single label; therefore, the end points of the range cannot be split between labels.

When these values are expressed in binary, they represent the bit values for the label at the range recognizers' end points. Don't cares are not allowed in the end point pattern specifications.

The RANGe query returns the range recognizer end point specifications for the range.

RANGe

Command Syntax: :TRACe:RANGe <label_name>,<start_pattern>,<stop_pattern>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<start_pattern> ::= "{#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_pattern> ::= "{#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}...}"

Examples:

```
OUTPUT XXX;":TRACE:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":TRACE:RANGE 'ABC', '#B00001111', '#HCF' "
```

Query Syntax: :TRACe:RANGe?

Returned Format: [:TRACe:RANGe] <label_name>,<start_pattern>,<stop_pattern><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:RANGE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RESTart command/query

The RESTart command selects the type of restart to be enabled during the trace sequence. It also defines the global restart qualifier that restarts the sequence in global restart mode. The qualifier may be a single term or a complex expression. The terms A through D are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Expressions are limited to what you can manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

A detailed example is provided in figure 5-2.

The RESTart query returns the current restart specification.

Command Syntax: :TRACe:RESTart {OFF | PERLevel | <restart_qualifier>}

where:

<restart_qualifier> ::= { ANYState | NOSTate | <any_term> ** | (<expression> **)}
<any_term> ::= {A|B|C|D|INRange*|OUTRange*|NOTA|NOTB|NOTC|NOTD}
<expression> ::= {<expression1> [OR <expression1>] | <expression1> [AND <expression1>]}
<expression 1> ::= {any_term[OR any_term] . . . | any_term[AND any_term] . . . }

* = Range choices only available if range label is available.
** = Only available in State mode.

Examples:

```
OUTPUT XXX;":TRACE:RESTART OFF"
OUTPUT XXX;":TRACE:RESTART PERLEVEL"
OUTPUT XXX;":TRACE:RESTART (NOTA AND NOTB AND INRANGE)"
OUTPUT XXX;":TRACE:RESTART (B OR (NOTC AND NOTD))"
```

RESTart

Query Syntax: :TRACe:RESTart?

Returned Format: [:TRACe:RESTart] {OFF | PERLevel | <restart_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:RESTART?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SEQuence command/query

The SEQuence command redefines the state analyzer trace sequence. First, it deletes the current trace sequence. Next, it inserts the number of levels specified with default settings, and creates the desired number of levels. The number of levels can be between 2 and 5.

The SEQuence query returns the current sequence specification.

Command Syntax: :TRACe:SEQuence <number_of_levels>,<level_of_trigger>

where:

<number_of_levels> ::= integer from 2 to 5

Example:
```
OUTPUT XXX;":TRACE:SEQUENCE 4"
```

Query Syntax: :TRACe:SEQuence?

Returned Format: [:TRACe:SEQuence] <number_of_levels> <NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:SEQUENCE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

STORe command/query

The STORe command defines the store qualifier for a given sequence level. Any data matching the STORe qualifier is stored in memory as part of the current trace data. The qualifier may be a single term or a complex expression. The terms A through D are defined by the TERM command. The meaning of INRange and OUTRange is determined by the RANGe command.

Expressions are limited to what you can manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

A detailed example is provided in figure 5-2.

The STORe query returns the current store qualifier specification for a given sequence level <N>.

Command Syntax: :TRACe:STORe <N> <store_qualifier>

where:

<N> ::= an integer from 1 to the number of existing sequence levels (maximum 5)

<store_qualifier> ::= { ANYState | NOSTate | <any_term> ** |(<expression>**)}

<any_term> ::= {A|B|C|D|INRange*|OUTRange*|NOTA|NOTB|NOTC|NOTD}

<expression> ::= {<expression1> [OR <expression1>] | <expression1> [AND <expression1>]}

<expression 1> ::= {any_term[OR any_term] . . .| any_term[AND any_term] . . . }

* = Range choices only available if range label is available.

** = Only available in State mode.

STORe

Examples:

```
OUTPUT XXX;":TRACE:STORE1 ANYSTATE"
OUTPUT XXX;":TRACE:STORE2 OUTRANGE"
OUTPUT XXX;":TRACE:STORE3 (NOTA AND NOTC AND NOTD)"
```

Query Syntax: :TRACe:STORe <N>?

Returned Format: [:TRACe:STORe <N>] <store_qualifier> <NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:STORE4?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TAG command/query

The TAG command selects the type of count tagging (state or time) to be performed during data acquisition. State tagging is indicated when the parameter is the state tag qualifier, which will be counted in the qualified state mode. The qualifier may be a single term or a complex expression. The terms A through D are defined by the TERM command. The terms INRange and OUTRange are defined by the RANGe command.

Expressions are limited to what you can manually enter through the Format menu. Regarding parentheses, the syntax definitions below show only the required ones. Additional parentheses are allowed as long as the meaning of the expression is not changed.

The TAG query returns the current count tag specification.

Command Syntax: :TRACe:TAG {TIME | <state_tag_qualifier>}

where:

<state_tag_qualifier> ::= { ANYState | NOSTate | <any_term>** |(<expression>**)}
<any_term> ::= {A|B|C|D|INRange*|OUTRange*|NOTA|NOTB|NOTC|NOTD}
<expression> ::= {<expression1>[OR <expression1>] | <expression1>[AND <expression1>]}
<expression 1> ::= {any_term[OR any_term] . . . | any_term[AND any_term] . . . }

* = Range choices only available if range label is available.
** = Only available in State mode.

TAG

Examples:
```
OUTPUT XXX;":TRACE:TAG OFF"
OUTPUT XXX;":TRACE:TAG TIME"
OUTPUT XXX;":TRACE:TAG (INRANGE OR NOTB)"
OUTPUT XXX;":TRACE:TAG ((INRANGE OR A) AND D)"
```

Query Syntax: :TRACe:TAG?

Returned Format: [:TRACe:TAG] {OFF|TIME|<state_tag_qualifier>}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:TAG?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TERM

command/query

The TERM command allows you to a specify a pattern recognizer term. Each command deals with only one label in the given term; therefore, a complete specification could require several commands. Since a label can contain 32 or less bits, the range of the pattern value will be between 2^{32} - 1 and 0. When the value of a pattern is expressed in binary, it represents the bit values for the label inside the pattern recognizer term. 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.

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

Command Syntax: :TRACe:TERM <term_id>,<label_name>,<pattern>

where:

<term_id> ::= {A|B|C|D}

<label_name> ::= 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;":TRACE:TERM A,'DATA','255' "
OUTPUT XXX;":TRACE:TERM B,'ABC','#BXXXX1101' "
```

TERM

Query Syntax: :TRACe:TERM? <term_id>,<label_name>

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

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":TRACE:TERM? B,'DATA' "
30 ENTER XXX;String$
40 PRINT String$
50 END
```

LIST Subsystem 6

Introduction

The LIST subsystem contains the following commands available for the Listing menu in the HP 16540A,D and 16541A,D logic analyzer module:

- COLumn
- DATA
- OVERlay
- LINE
- MMODe
- OPATtern
- OSEarch
- OSTate
- OTAG
- RUNTil
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XOTAG
- XPATtern
- XSEarch
- XSTate
- XTAG

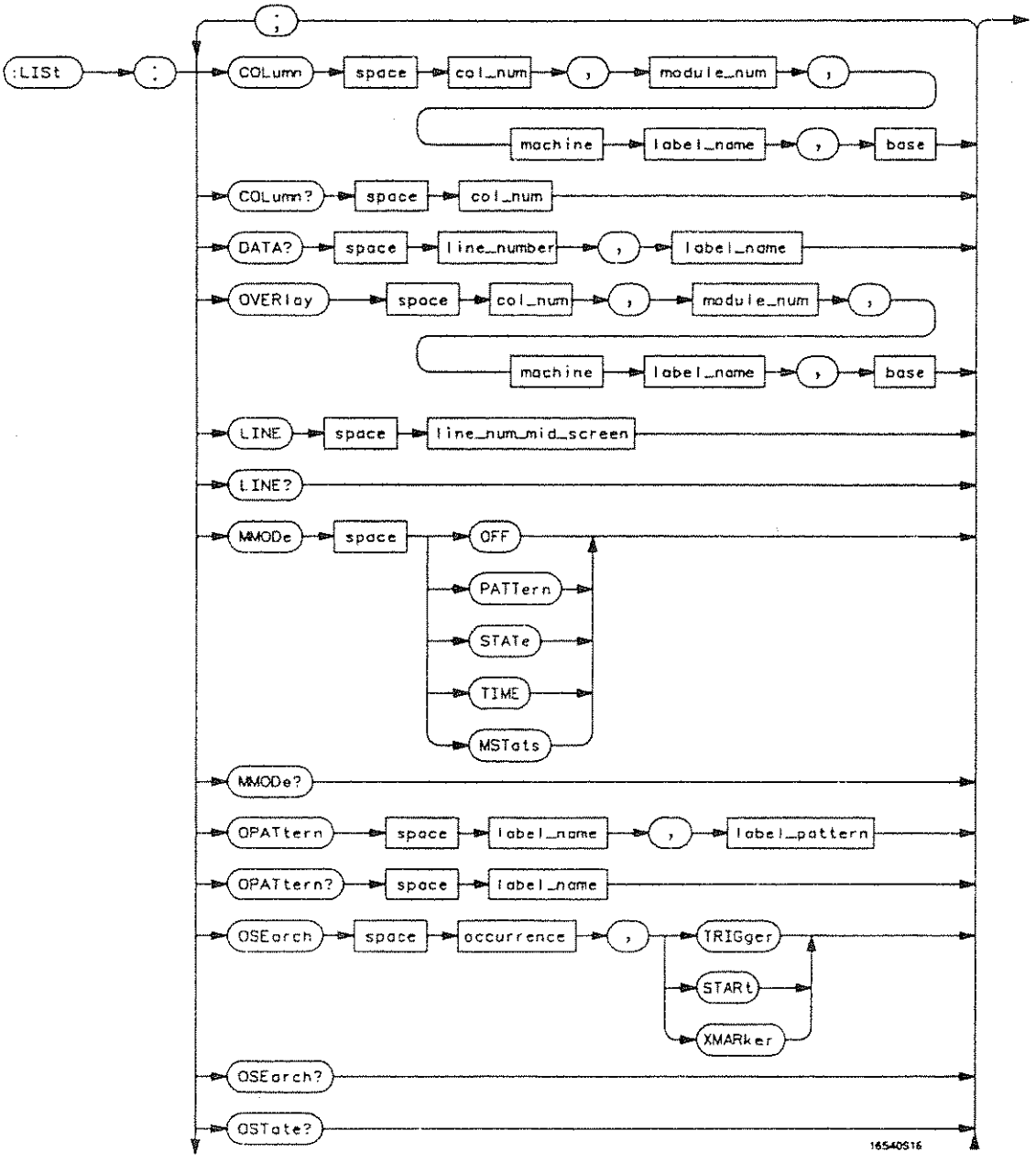


Figure 6-1. LIST Subsystem Syntax Diagram

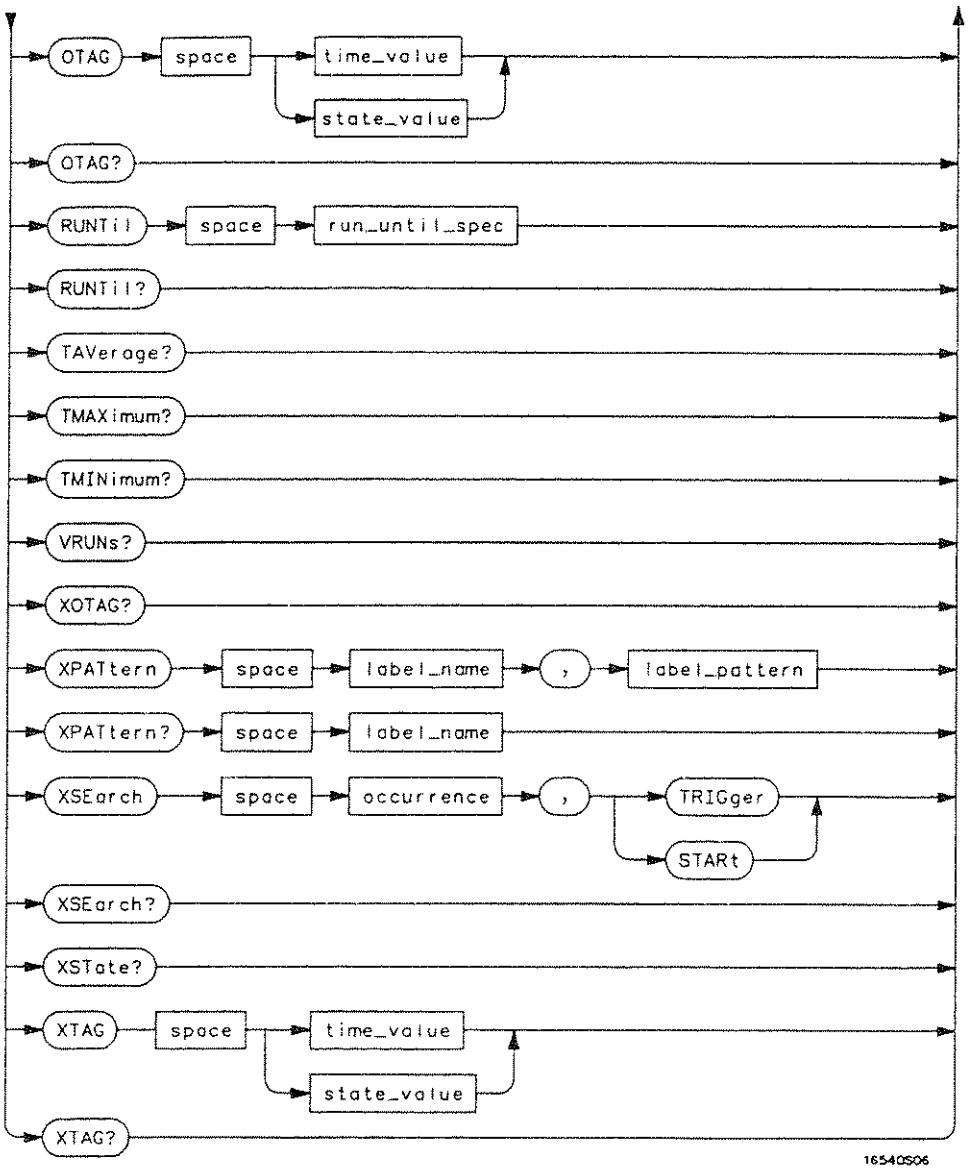


Figure 6-1. LIST Subsystem Syntax Diagram (cont.)

module_num = {*1*|*2*|*3*|*4*|*5*}
machine = {*1*|*2*}
col_num = {*1*|*2*|*3*. . . |*61*}
line_number = *integer from -16384 to +16384*
label_name = *a string of up to 6 alphanumeric characters*
base = {*BINary*|*HEXadecimal*|*OCTal*|*DECimal*|*ASCii*|*SYMBol*|*IASSembler*|*TWOS*} *for labels or*
{*ABSolute*|*RELative*} *for tags*
line_num_mid_screen = *integer from -16384 to +16384*
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*}. . . }"
occurrence = *integer from -16384 to +16384*
time_value = *real number*
state_value = *integer*
run_until_spec = {*LT,<value>*|*GT,<value>*|*INRange,<value>,<value>*|
OUTRange,<value>,<value>|*EQUal*|*NEQual*}
value = *real number*

LIST selector

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

Command Syntax: :LIST

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

COLumn command/query

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 (60 for timing) in the menu. A column number of 1 refers to the left most column. When a label is assigned to a column it replaces the original label in that column. The label originally in the specified column is placed in the column the specified label is moved from.

When the label name is "TAGS," the TAGS column is assumed and the next parameter must specify RELative or ABSolute (state only).

Note A label for tags must be assigned in order to use ABSolute or RELative state tagging.

The COLumn query returns the column number, label name, and base for the specified column.

COLumn

Command Syntax: :LIST:COLumn
<col_num>,[<module_num>,MACHine{1|2}],<label_name>,<base>

where:

<col_num> ::= {1|2|3|...|60}
<label_name> ::= a string of up to 6 alphanumeric characters
<base> ::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler|TWOS} for labels or
::= {ABSolute|RELative} for tags

Examples:
```
OUTPUT XXX;":LIST:COLUMN 4,2,MACHINE1,'A',HEX"
OUTPUT XXX;":LIST:COLUMN 1,'TAGS', ABSOLUTE"
```

Query Syntax: :LIST:COLumn? <col_num>

Returned Format: [:LIST:COLumn]
<col_num>,[<module_num>,MACHine{1|2}],<label_name>,<base><NL>

Example:
```
10 DIM C1$[100]
20 OUTPUT XXX;":LIST:COLUMN? 4"
30 ENTER XXX;C1$
40 PRINT C1$
50 END
```

DATA query

The DATA query returns the value at a specified line number for a given label. The format will be the same as the one shown in the Listing display.

Query Syntax: :LIST:DATA? <line_number>,<label_name>

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

where:

<line_number> ::= integer from -16384 to +16384

<label_name> ::= string of up to 6 alphanumeric characters

<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:

```
10 DIM Sd$[100]
20 OUTPUT XXX;":LIST:DATA? 512, 'RAS'"
30 ENTER XXX;Sd$
40 PRINT Sd$
50 END
```

OVERlay command

The OVERlay command allows you to interleave a label from another time-correlated machine into the state listing. Both machines from which the labels are assigned, must be configured in a Group Run, and the Count time must be selected in the Trace menu before the OVERlay option becomes available.

Command Syntax: :LIST:OVERlay
<col_num>,[<module_num>,MACHine{1|2}],<label_name>,<base>

where:

<col_num> ::= {1|2|3|...|60}
<label_name> ::= a string of up to 6 alphanumeric characters
<base> ::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler|TWOS} for labels or
::= {ABSolute|RELative} for tags

Example:
```
10 OUTPUT XXX;":LIST:OVERlay 4,3,MACHINE1,'A'
```

LINE

command/query

The LINE command allows you to scroll the analyzer listing vertically. The command specifies the line number relative to the trigger that the analyzer will highlight at the center of the screen.

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

Command Syntax: :LIST:LINE <line_num_mid_screen>

where:

<line_num_mid_screen> ::= integer from -16384 to +16384

Example:

```
OUTPUT XXX;":LIST:LINE 0"
```

Query Syntax: :LIST:LINE?

Returned Format: [:LIST:LINE] <line_num_mid_screen><NL>

Example:

```
10 DIM Ln$[100]
20 OUTPUT XXX;":LIST:LINE?"
30 ENTER XXX;Ln$
40 PRINT Ln$
50 END
```

MMODe

command/query

The MMODe command (Marker Mode) selects the mode controlling the marker movement and the display of marker readouts. When PATTern is selected, the markers are placed on patterns. When STATe is selected and state tagging is on, the markers move on qualified states counted between normally stored states. When TIME is selected and time tagging is enabled, the markers move on time between stored states. When MSTats is selected and time tagging is on, the markers are placed on patterns, but the readouts will be time statistics.

In timing mode, only TIME is allowed.

The MMODe query returns the current marker mode selected.

Command Syntax: :LIST:MMODe <marker_mode>

where:

<marker_mode> ::= {OFF|PATTern|STATe|TIME|MSTats}

Example:

```
OUTPUT XXX;":LIST:MMODE TIME"
```

Query Syntax: :LIST:MMODe?

Returned Format: [:LIST:MMODe] <marker_mode><NL>

Example:

```
10 DIM Mn$[100]
20 OUTPUT XXX;":LIST:MMODE?"
30 ENTER XXX;Mn$
40 PRINT Mn$
50 END
```

OPATtern command/query

The OPATtern command allows you to construct a pattern recognizer term for the O Marker which is then used with the OSEarch criteria 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. Since a label may not have more than 32 bits, the value must be between 0 and 2^{32} - 1, no matter which base is used. Because the <label_pattern> parameter may contain don't cares, it is handled as a string of characters rather than a number.

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.

Command Syntax: :LIST:OPATtern <label_name>,<label_pattern>

where:

<label_name> ::= 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' "
```

OPATtern

Query Syntax: :LIST:OPATtern? <label_name>

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

Example:

```
10 DIM Op$[100]
20 OUTPUT XXX;":LIST:OPATTERN? 'A'"
30 ENTER XXX;Op$
40 PRINT Op$
50 END
```

OSEarch command/query

The OSEarch command defines the search criteria for the O marker. The OSEarch command is used with the OPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the marker to begin a search with the trigger, the start of data, or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OPATtern recognizer specification, 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.

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

Command Syntax: :LIST:OSEarch <occurrence>,<origin>

where:

<occurrence> ::= integer from -16384 to +16384
<origin> ::= {TRIGger|STARt|XMARker}

Example:
```
OUTPUT XXX;":LIST:OSEARCH +10,TRIGGER"
```

Query Syntax: :LIST:OSEarch?

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

Example:
```
10 DIM Os$[100]
20 OUTPUT XXX;":LIST:OSEARCH?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OSTate query

The OSTate query returns the line number in the listing where the O marker resides (-16384 to +16384). If data is not valid , the query returns 32767.

Query Syntax: :LIST:OSTate?

Returned Format: [:LIST:OSTate] <state_num> <NL>

where:

<state_num> ::= an integer from -16384 to +16384, or 32767

Example:

```
10 DIM Os$[100]
20 OUTPUT XXX;":LIST:OSTATE?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OTAG command/query

The OTAG command specifies the tag value on which the O Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed. The OTAG command may only be used when the marker mode is Time or States (use MMODE command).

The OTAG query returns the O Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time or through a pattern search. If data is not valid, the query returns 9.9E37 for time tagging, 32767 for state tagging.

Command Syntax: :LIST:OTAG {<time_value> | <state_value>}

where:

<time_value> ::= real number
<state_value> ::= integer

Example:
```
:OUTPUT XXX;":LIST:OTAG 40.0E-6"
```

Query Syntax: :LIST:OTAG?

Returned Format: [:LIST:OTAG] {<time_value> | <state_value>}<NL>

Example:
```
10 DIM Ot$[100]
20 OUTPUT XXX;":LIST:OTAG?"
30 ENTER XXX;Ot$
40 PRINT Ot$
50 END
```

RUNTil command/query

The RUNTil (run until) command allows you to define a stop condition when the trace 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 TRACe 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 10 ns apart since this is the minimum time resolution of the time tag counter.

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).

The RUNTil query returns the current stop criteria.

Note The RUNTil instruction is available in both the LIST and COMPare subsystems.

RUNTil

Command Syntax: :LIST:RUNTil <run_until_spec>

where:

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

<value> ::= real number from -9E9 to +9E9

Example:

```
OUTPUT XXX;":LIST:RUNTIL GT,800.0E-6"
```

Query Syntax: :LIST:RUNTil?

Returned Format: [:LIST:RUNTil] <run_until_spec><NL>

Example:

```
10 DIM Ru$[100]
20 OUTPUT XXX;":LIST:RUNTIL?"
30 ENTER XXX;Ru$
40 PRINT Ru$
50 END
```

REMove command

The REMove command removes all labels except the first label in the Listing menu. There are no parameters to this command.

Command Syntax LIST:REMove

Example 20 OUTPUT XXX;":LIST:REMove"

TAVerage query

The TAVerage query returns the value of the average time between the X and O Markers. If the number of valid runs is zero, the query returns 9.9E37. Valid runs are those where the pattern search for both the X and O markers was successful, resulting in valid delta-time measurements.

Query Syntax: :LIST:TAVerage?

Returned Format: [:LIST:TAVerage] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$[100]
20 OUTPUT XXX;":LIST:TAVERAGE?"
30 ENTER XXX;Tv$
40 PRINT Tv$
50 END
```

TMAXimum query

The TMAXimum query returns the value of the maximum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Query Syntax: :LIST:TMAXimum?

Returned Format: [:LIST:TMAXimum] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tx$[100]
20 OUTPUT XXX;":LIST:TMAXIMUM?"
30 ENTER XXX;Tx$
40 PRINT Tx$
50 END
```

TMINimum query

The TMINimum query returns the value of the minimum time between the X and O Markers. If data is not valid, the query returns 9.9E37.

Query Syntax: :LIST:TMINimum?

Returned Format: [:LIST:TMINimum] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$[100]
20 OUTPUT XXX;":LIST:TMINIMUM?"
30 ENTER XXX;Tm$
40 PRINT Tm$
50 END
```

VRUNs

query

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.

Query Syntax: :LIST:VRUNs?

Returned Format: [:LIST:VRUNs] <valid_runs>,<total_runs><NL>

where:

<valid_runs> ::= zero or positive integer
<total_runs> ::= zero or positive integer

Example:

```
10 DIM Vr$[100]
20 OUTPUT XXX;":LIST:VRUNS?"
30 ENTER XXX;Vr$
40 PRINT Vr$
50 END
```

XOTAG query

The XOTAG query returns the time from the X to O markers when the marker mode is time or number of states from the X to O markers when the marker mode is state. If there is no data in the time mode the query returns 9.9E37. If there is no data in the state mode, the query returns 32767.

Query Syntax: :LIST:XOTAG?

Returned Format: [:LIST:XOTAG] {<XO_time> | <XO_states>}<NL>

where:

<XO_time> ::= real number
<XO_states> ::= integer

Example:

```
10 DIM Xot$[100]
20 OUTPUT XXX;":LIST:XOTAG?"
30 ENTER XXX;Xot$
40 PRINT Xot$
50 END
```

XPATtern

command/query

The XPATtern command allows you to construct a pattern recognizer term for the X Marker which is then used with the XSEarch criteria when moving the marker on patterns. Since this command deals with only one label at a time, a complete specification could require several iterations.

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.

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.

Command Syntax: :LIST:XPATtern <label_name>,<label_pattern>

where:

<label_name> ::= 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' "
```

XPATtern

Query Syntax: :LIST:XPATtern? <label_name>

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

Example:

```
10 DIM Xp$[100]
20 OUTPUT XXX;":LIST:XPATTERN? 'A'"
30 ENTER XXX;Xp$
40 PRINT Xp$
50 END
```

XSEarch command/query

The XSEarch command defines the search criteria for the X Marker. The XSEarch command is associated to the XPATtern recognizer specification when moving the markers on patterns. The origin parameter tells the Marker to begin a search with the trigger or with the start of data. 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.

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

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

where:

<occurrence> ::= integer from -16384 to +16384
<origin> ::= {TRIGger|STARt}

Example:
```
OUTPUT XXX;":LIST:XSEARCH +10,TRIGGER"
```

Query Syntax: :LIST:XSEarch?

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

Example:
```
10 DIM Xs$[100]
20 OUTPUT XXX;":LIST:XSEARCH?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XSTate

query

The XSTate query returns the line number in the listing where the X marker resides (-16384 to +16384). If data is not valid, the query returns 32767.

Query Syntax: :LIST:XSTate?

Returned Format: [:LIST:XSTate] <state_num> <NL>

where:

<state_num> ::= an integer from -16384 to +16384, or 32767

Example:

```
10 DIM Xs$[100]
20 OUTPUT XXX;":LIST:XSTATE?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XTAG

command/query

The XTAG command specifies the tag value on which the X Marker should be placed. The tag value is time when time tagging is on or states when state tagging is on. If the data is not valid tagged data, no action is performed. The XTAG command may only be used when the marker mode is Time or States (use MMODE command).

The XTAG query returns the X Marker position in time when time tagging is on or in states when state tagging is on, regardless of whether the marker was positioned in time or through a pattern search. If data is not valid, the query returns 9.9E37 for time tagging, 32767 for state tagging.

Command Syntax: :LIST:XTAG {<time_value> | <state_value>}

where:

<time_value> ::= real number
<state_value> ::= integer

Example:
```
:OUTPUT XXX;":LIST:XTAG 40.0E-6"
```

Query Syntax: :LIST:XTAG?

Returned Format: [:LIST:XTAG] {<time_value> | <state_value>}<NL>

Example:
```
10 DIM Xt$[100]
20 OUTPUT XXX;":LIST:XTAG?"
30 ENTER XXX;Xt$
40 PRINT Xt$
50 END
```

SWAVeform Subsystem 7

Introduction

The commands in the State Waveform subsystem allow you to configure the display to view up to 96 channels as state data waveforms at a time. The waveforms are identified by label name and bit number. The five commands are analogous to their counterparts in the Timing Waveform subsystem. However, in this subsystem the x-axis is restricted to representing only samples (states), regardless of the type of tagging. As a result, the only commands which can be used for scaling are DELay and RANGe.

The way to manipulate the X and O markers on the Waveform display is through the State Listing (LIST) subsystem. Using the marker commands from the LIST subsystem will affect the markers on the State Waveform display.

The commands in the SWAVeform subsystem are listed below:

- ACCumulate
- DELay
- INSert
- RANGe
- REMove

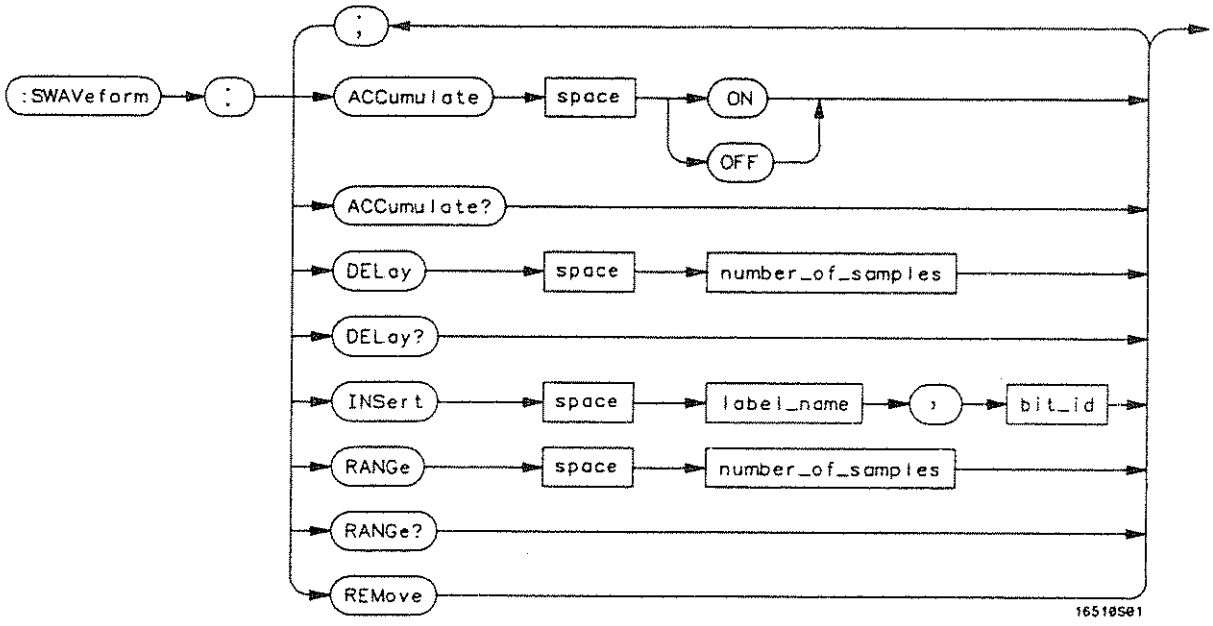


Figure 7-1. SWAVeform Subsystem Syntax Diagram

number_of_samples = *integer from -16384 to + 16384*
label_name = *string of up to 6 alphanumeric characters*
bit_id = {*OVERlay*|<*bit_num*|*ALL*>}
bit_num = *integer representing a label bit from 0 to 31*

SWAVeform selector

The SWAVeform (State Waveform) selector is used as part of a compound header to access the settings in the State Waveform menu.

Command Syntax: :SWAVeform

Example:

```
OUTPUT XXX;":SWAVEFORM:RANGE 4"
```

ACCumulate command/query

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

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

Command Syntax: :SWAVeform:ACCumulate {{ON | 1} | {OFF | 0}}

Example:
```
OUTPUT XXX;":SWAVEFORM:ACCUMULATE ON"
```

Query Syntax: :SWAVeform:ACCumulate?

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

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":SWAVEFORM:ACCUMULATE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

DELay command/query

The DELay command allows you to specify the number of samples between the timing trigger and the horizontal center of the screen for the waveform display. The allowed number of samples is from -16384 to +16384.

The DELay query returns the current sample offset value.

Command Syntax: :SWAVeform:DELay <number_of_samples>

where:

<number_of_samples> ::= integer from −16384 to +16384

Example:
```
OUTPUT XXX;":SWAVEFORM:DELAY 127"
```

Query Syntax: :SWAVeform:DELay?

Returned Format: [SWAVeform:DELay] <number_of_samples><NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":SWAVEFORM:DELAY?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

INSert command

The INSert command allows you to add waveforms to the state waveform display. Waveforms are added from top to bottom on the screen. When 96 waveforms are present, inserting an additional waveform replaces the last waveform. Bit numbers are zero based, so a label with 8 bits is referenced as bits 0-7. Specifying OVERlay causes a composite waveform display of all bits or channels for the specified label. If ALL is specified, all the bits are displayed sequentially.

Command Syntax: :SWAVeform:INSert <label_name>,<bit_id>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= {OVERlay|<bit_num>|ALL}
<bit_num> ::= integer representing a label bit from 0 to 31

Examples:

```
OUTPUT XXX;":SWAVEFORM:INSERT 'WAVE', 19"
OUTPUT XXX;":SWAVEFORM:INSERT 'ABC', OVERLAY"
OUTPUT XXX;":SWAV:INSERT 'POD1', #B1001"
```

RANGe

command/query

The RANGe command allows you to specify the number of samples across the screen on the State Waveform display. It is equivalent to ten times the states per division setting (st/div) on the front panel. A number between 10 and 5000 can be entered.

The RANGe query returns the current range value.

Command Syntax: :SWAVeform:RANGe <number_of_samples>

where:

<number_of_samples> ::= integer from 10 to 5000

Example:

```
OUTPUT XXX;":SWAVEFORM:RANGE 10"
```

Query Syntax: :SWAVeform:RANGe?

Returned Format: [SWAVeform:RANGe] <number_of_samples><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":SWAVEFORM:RANGE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

REMove command

The REMove command allows you to clear the waveform display before building a new display.

Command Syntax: :SWAVeform:REMove

Example:

```
OUTPUT XXX;":SWAVEFORM:REMOVE"
```

TWAVeform Subsystem 8

Introduction

The TWAVeform subsystem contains the commands available for the Timing Waveforms menu in the HP 16540A,D and 16541A,D. These commands are listed below:

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

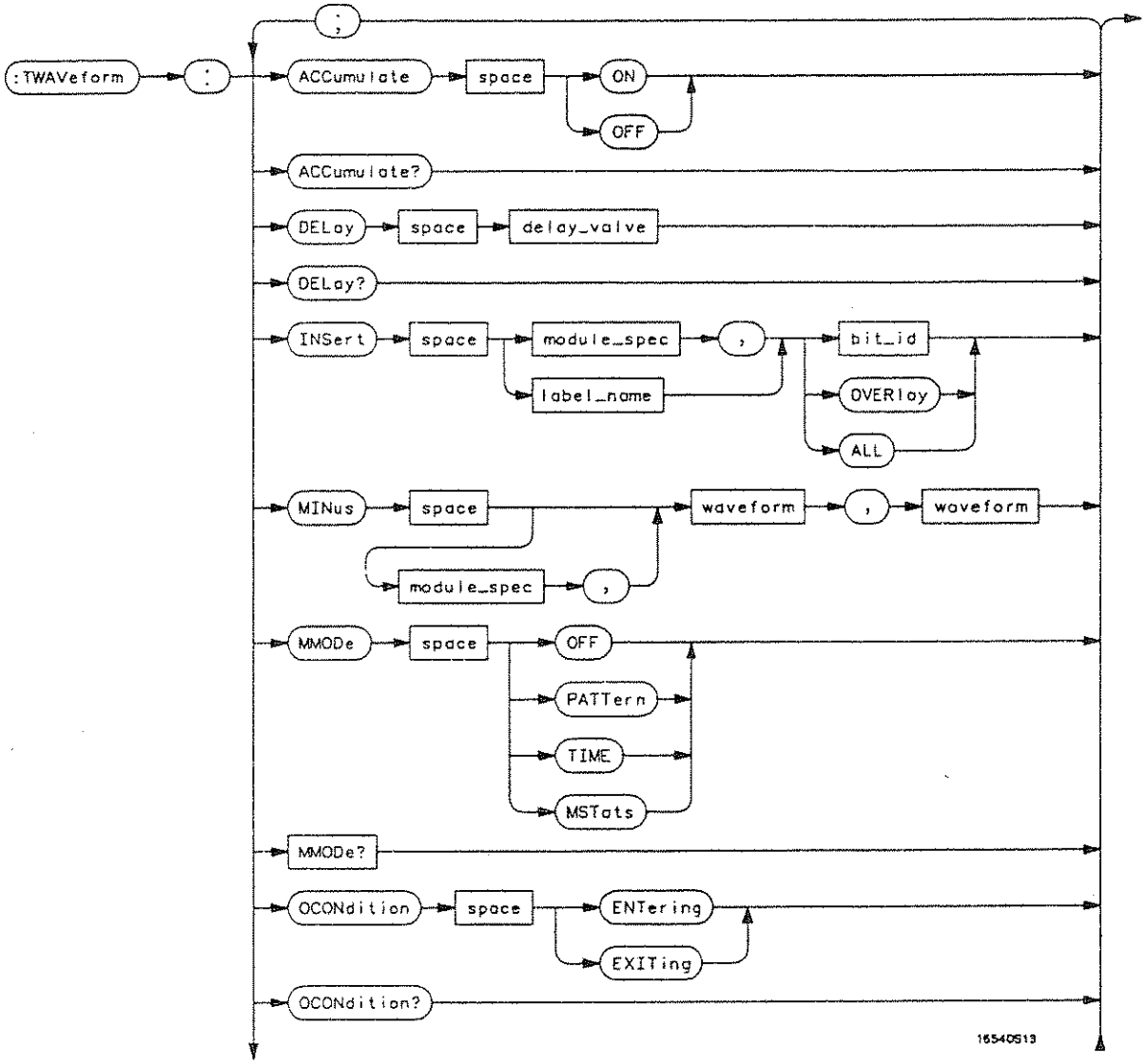


Figure 8-1. TWAVeform Subsystem Syntax Diagram

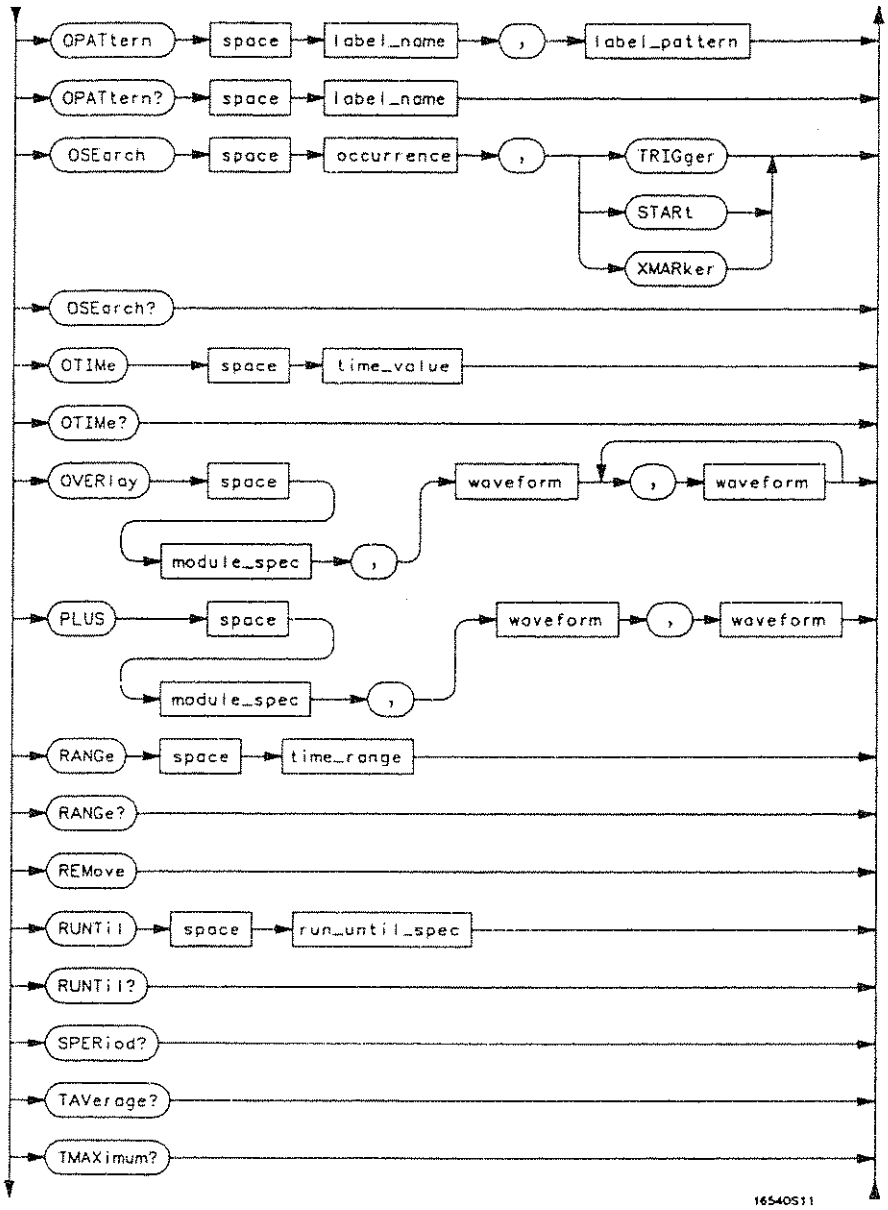


Figure 8-1. TWAVeform Subsystem Syntax Diagram (continued)

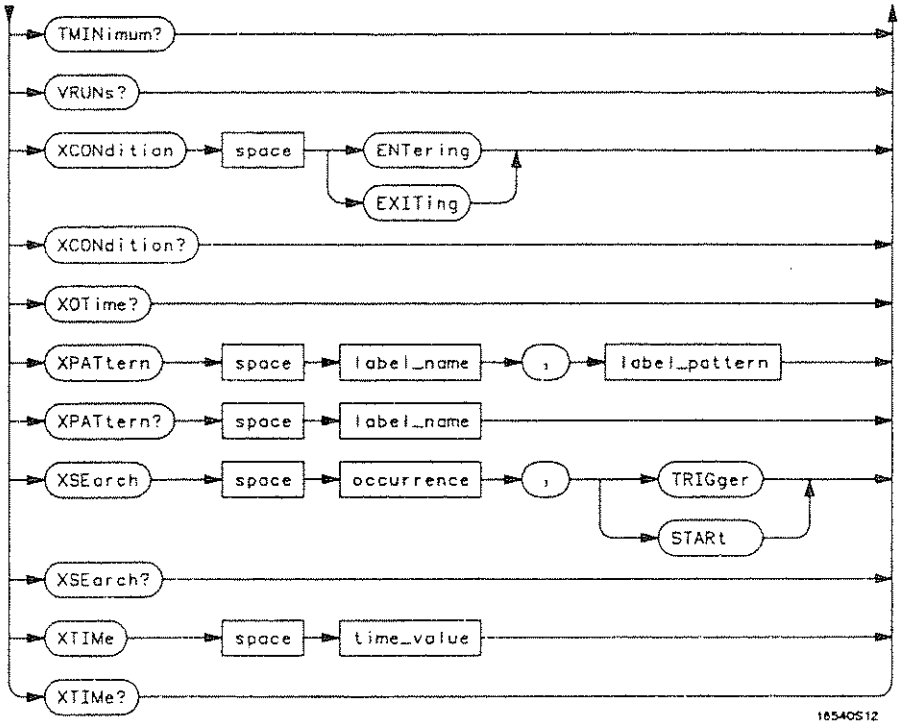


Figure 8-1. TWAVeform Subsystem Syntax Diagram (continued)

delay_value = *real number between -2500 s and +2500 s*
module_spec = *{1|2|3|4|5}*
bit_id = *integer from 0 to 31*
waveform = *string containing <acquisition_spec> {1|2}*
acquisition_spec = *{A|B|C|D|E} (slot where acquisition card is located)*
label_name = *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}... }"
occurrence = *integer*
time_value = *real number*
time_range = *real number between 100 ns and 1 ks*
run_until_spec = *{LT,<value>|GT,<value>|INRange,<value>,<value>|*
OUTRange,<value>,<value>|EQUal|NEQual}
value = *real number*

TWAVeform selector

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

Command Syntax: :TWAVeform

Example:

```
OUTPUT XXX;":TWAVEFORM:DELAY 100E-9"
```

ACCumulate command/query

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 ones.

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

Command Syntax: :TWAVeform:ACCumulate <setting>

where:

<setting> ::= {0|OFF} or {1|ON}

Example:
```
OUTPUT XXX;":TWAVEFORM:ACCUMULATE ON"
```

Query Syntax: :TWAVeform:ACCumulate?

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

Example:
```
10 DIM P$ [100]
20 OUTPUT XXX;":TWAVEFORM:ACCUMULATE?"
30 ENTER XXX; P$
40 PRINT P$
50 END
```

DELay command/query

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 -2500 s to +2500 s.

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

Command Syntax: :TWAVeform:DELay <delay_value>

where:

<delay_value> ::= real number between -2500 s and +2500 s

Example:

```
OUTPUT XXX;":TWAVEFORM:DELAY 100E-6"
```

Query Syntax: :TWAVeform:DELay?

Returned Format: [:TWAVeform:DELay] <time_value><NL>

Example:

```
10 DIM D1$ [100]
20 OUTPUT XXX;":TWAVEFORM:DELAY?"
30 ENTER XXX; D1$
40 PRINT D1$
50 END
```

INSert command

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 the oscilloscope or high speed timing modules, the optional first parameter must be used. 1...5 corresponds to modules A...E. 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.

INSert

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

where:

<module_spec> ::= {1|2|3|4|5}
<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= integer from 0 to 31

Example:
```
OUTPUT XXX;":TWAVEFORM:INSERT 3, 'WAVE',10"
```

Inserting Oscilloscope Waveforms

Inserting a waveform from an oscilloscope to the timing waveforms display:

Command Syntax: :TWAVeform:INSert <module_spec>,<label_name>

where:

<module_spec> ::= {1|2|3|4|5} slot in which timebase card is installed
<label_name> ::= string of one alpha and one numeric character

Example:
```
OUTPUT XXX;":TWAVEFORM:INSERT 5, 'C1'"
```

MINus command

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. 1...5 refers to slots A...E. The next two parameters specify which waveforms will be subtracted from each other.

Note MINus is only available for oscilloscope waveforms.

Command Syntax: :TWAVeform:MINus <module_spec>,<waveform>,<waveform>

where:

<module_spec> ::= {1|2|3|4|5}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E} (slot where acquisition card is located)

Example:

```
OUTPUT XXX; ":TWAVEFORM:MINUS 2,'A1','A2'"
```

MMODe command/query

The MMODe (Marker Mode) command selects the mode controlling marker movement and the display of the marker readouts. When PATTern is selected, the markers will be placed on patterns. When TIME is selected, the markers move on time. In MSTats, the markers are placed on patterns, but the readouts will be time statistics.

The MMODe query returns the current marker mode.

Command Syntax: :TWAVeform:MMODe {OFF|PATTern|TIME|MSTats}

Example:
```
OUTPUT XXX; ":TWAVEFORM:MMODE TIME"
```

Query Syntax: :TWAVeform:MMODe?

Returned Format: [:TWAVeform:MMODe] <marker_mode> <NL>

where

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

Example:
```
10 DIM M$ [100]
20 OUTPUT XXX;":TWAVEFORM:MMODE?"
30 ENTER XXX; M$
40 PRINT M$
50 END
```

OCONdition command/query

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.

The OCONdition query returns the current setting.

Command Syntax: :TWAVeform:OCONdition {ENTering|EXITing}

Example:
```
OUTPUT XXX; ":TWAVEFORM:OCONDITION ENTERING"
```

Query Syntax: :TWAVeform:OCONdition?

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

Example:
```
10 DIM Oc$ [100]
20 OUTPUT XXX;":TWAVEFORM:OCONDITION?"
30 ENTER XXX; Oc$
40 PRINT Oc$
50 END
```

OPATtern command/query

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.

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.

Command Syntax: :TWAVeform:OPATtern <label_name>,<label_pattern>

where:

<label_name> ::= 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'"
```

OPATtern

Query Syntax: :TWAVeform:OPATtern? <label_name>

Returned Format: [:TWAVeform:OPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Op$ [100]
20 OUTPUT XXX;":TWAVEFORM:OPATTERN? 'A'"
30 ENTER XXX; Op$
40 PRINT Op$
50 END
```

OSEarch command/query

The OSEarch command defines the search criteria for the O marker which is then 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 with the trigger, start, or with the X marker. The actual occurrence the marker searches for is determined by the occurrence parameter of the OPATtern recognizer specification, 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.

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

Command Syntax: :TWAVeform:OSEarch <occurrence>,<origin>

where:

<origin> ::= {TRIGger|STARt|XMARker}
<occurrence> ::= integer from -9999 to +9999

Example:
```
OUTPUT XXX; ":TWAVEFORM:OSEARCH +10,TRIGGER"
```

Query Syntax: :TWAVeform:OSEarch?

Returned Format: [:TWAVeform:OSEarch] <occurrence>,<origin><NL>

Example:
```
10 DIM Os$ [100]
20 OUTPUT XXX;":TWAVEFORM:OSEARCH?"
30 ENTER XXX; Os$
40 PRINT Os$
50 END
```

OTIMe command/query

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.

The OTIMe query returns the O marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :TWAVeform:OTIMe <time_value>

where:

<time_value> ::= real number -2.5Ks to +2.5Ks

Example:
```
OUTPUT XXX; ":TWAVEFORM:OTIME 30.0E-6"
```

Query Syntax: :TWAVeform:OTIMe?

Returned Format: [:TWAVeform:OTIMe] <time_value><NL>

Example:
```
10 DIM Ot$ [100]
20 OUTPUT XXX;":TWAVEFORM:OTIME?"
30 ENTER XXX; Ot$
40 PRINT Ot$
50 END
```

OVERlay command

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 syntax specifies which slot contains the oscilloscope time base card. The next parameters are the labels of the waveforms that are to be overlaid.

Note OVERlay is only available for oscilloscope waveforms.

Command Syntax: :TWAVeform:OVERlay <module_number>, <label>[, <label>]...

where:

<module_spec> ::= {1|2|3|4|5}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E} (slot where acquisition card is located)

Example:

```
OUTPUT XXX;":TWAVEFORM:OVERLAY 4, 'C1','C2'"
```

PLUS command

The PLUS command inserts time-correlated A + B oscilloscope waveforms on the screen. The first parameter is the module specifier where the oscilloscope module resides. 1...5 refers to slots A...E. The next two parameters specify which waveforms will be added to each other.

Note PLUS is only available for oscilloscope waveforms.

Command Syntax: :TWAVeform:PLUS <module_spec>,<waveform>,<waveform>

where:

<module_spec> ::= {1|2|3|4|5}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E} (slot where acquisition card is located)

Example:
```
OUTPUT XXX; ":TWAVEFORM:PLUS 2,'A1','A2'"
```

RANGe command/query

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 100 ns to 1 ks.

The RANGe query returns the current full-screen time.

Command Syntax: :TWAVeform:RANGe <time_value>

where:

<time_range> ::= real number between 100 ns and 1 ks

Example:
```
OUTPUT XXX;":TWAVEFORM:RANGE 100E-9"
```

Query Syntax: :TWAVeform:RANGe?

Returned Format: [:TWAVeform:RANGe] <time_value><NL>

Example:
```
10 DIM Rg$ [100]
20 OUTPUT XXX;":TWAVEFORM:RANGE?"
30 ENTER XXX; Rg$
40 PRINT Rg$
50 END
```

REMove command

The REMove command deletes all waveforms from the display.

Command Syntax: :TWAVeform:REMove

Example:

```
OUTPUT XXX;":TWAVEFORM:REMOVE"
```

RUNTil command/query

The RUNTil (run until) command defines stop criteria based on the time between the X and O markers when the trace 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).

The RUNTil query returns the current stop criteria.

Command Syntax: :TWAVeform:RUNTil <run_until_spec>

where:

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

<value> ::= real number

Examples:
```
OUTPUT XXX;":TWAVEFORM:RUNTIL GT, 800.0E-6"
OUTPUT XXX;":TWAVEFORM:RUNTIL INRANGE, 4.5, 5.5"
```

Query Syntax: :TWAVeform:RUNTil?

Returned Format: [:TWAVeform:RUNTil] <run_until_spec><NL>

Example:
```
10 DIM Ru$ [100]
20 OUTPUT XXX;":TWAVEFORM:RUNTIL?"
30 ENTER XXX; Ru$
40 PRINT Ru$
50 END
```

SPERiod command/query

The SPERiod command specifies the sample period of the analyzer in the Timing Waveform menu. The SPERiod query returns the sample period of the last run.

Command Syntax: :TWAVeform:SPERiod <time_value>

where:

<time_value> ::= real number

Query Syntax: :TWAVeform:SPERiod?

Returned Format: [:TWAVeform:SPERiod] <time_value><NL>

Example:

```
10 DIM Sp$ [100]
20 OUTPUT XXX;":TWAVEFORM:SPERIOD?"
30 ENTER XXX; Sp$
40 PRINT Sp$
50 END
```

TAVerage query

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.

Query Syntax: :TWAVeform:TAVerage?

Returned Format: [:TWAVeform:TAVerage] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$ [100]
20 OUTPUT XXX;":TWAVEFORM:TAVERAGE?"
30 ENTER XXX; Tv$
40 PRINT Tv$
50 END
```

TMAXimum query

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.

Query Syntax: :TWAVeform:TMAXimum?

Returned Format: [:TWAVeform:TMAXimum] <time_value> <NL>

where

<time_value> ::= real number

Example:

```
10 DIM Tx$ [100]
20 OUTPUT XXX;":TWAVEFORM:TMAXIMUM?"
30 ENTER XXX; Tx$
40 PRINT Tx$
50 END
```

TMINimum

query

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.

Query Syntax: :TWAVeform:TMINimum?

Returned Format: [:TWAVeform:TMINimum] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$ [100]
20 OUTPUT XXX;":TWAVEFORM:TMINIMUM?"
30 ENTER XXX; Tm$
40 PRINT Tm$
50 END
```

VRUNs query

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.

Query Syntax: :TWAVeform:VRUNs?

Returned Format: [:TWAVeform:VRUNs] <valid_runs>,<total_runs><NL>

where:

<valid_runs> ::= zero or positive integer
<total_runs> ::= zero or positive integer

Example:

```
10 DIM Vr$ [100]
20 OUTPUT XXX;":TWAVEFORM:VRUNS?"
30 ENTER XXX; Vr$
40 PRINT Vr$
50 END
```

XCONdition command/query

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.

The XCONdition query returns the current setting.

Command Syntax: :TWAVeform:XCONdition {ENTering|EXITing}

Example:
```
OUTPUT XXX; ":TWAVEFORM:XCONDITION ENTERING"
```

Query Syntax: :TWAVeform:XCONdition?

Returned Format: [:TWAVeform:XCONdition] {ENTering|EXITing}<NL>

Example:
```
10 DIM Xc$ [100]
20 OUTPUT XXX;":TWAVEFORM:XCONDITION?"
30 ENTER XXX; Xc$
40 PRINT Xc$
50 END
```

XOTime query

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

Query Syntax: :TWAVeform:XOTime?

Returned Format: [:TWAVeform:XOTime] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Xot$ [100]
20 OUTPUT XXX;":TWAVEFORM:XOTIME?"
30 ENTER XXX; Xot$
40 PRINT Xot$
50 END
```

XPATtern command/query

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.

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.

Command Syntax: :TWAVeform:XPATtern <label_name>,<label_pattern>

where:

<label_name> ::= 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'"
```

XPATtern

Query Syntax: :TWAVeform:XPATtern? <label_name>

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

Example:

```
10 DIM Xp$ [100]
20 OUTPUT XXX;":TWAVEFORM:XPATTERN? 'A'"
30 ENTER XXX; Xp$
40 PRINT Xp$
50 END
```

XSEarch command/query

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 with the trigger or start. 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.

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

Command Syntax: :TWAVeform:XSEarch <occurrence>,<origin>

where:

<origin> ::= {TRIGger|STARt}
<occurrence> ::= integer from -9999 to +9999

Example:
```
OUTPUT XXX; ":TWAVEFORM:XSEARCH,+10,TRIGGER"
```

Query Syntax: :TWAVeform:XSEarch?

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

Example:
```
10 DIM Xs$ [100]
20 OUTPUT XXX;":TWAVEFORM:XSEARCH?"
30 ENTER XXX; Xs$
40 PRINT Xs$
50 END
```

XTIMe command/query

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.

The XTIMe query returns the X marker position in time. If data is not valid, the query returns 9.9E37.

Command Syntax: :TWAVeform:XTIMe <time_value>

where:

<time_value> ::= real number from -2.5Ks to +2.5Ks

Example:
```
OUTPUT XXX; ":TWAVEFORM:XTIME 40.0E-6"
```

Query Syntax: :TWAVeform:XTIMe?

Returned Format: [:TWAVeform:XTIMe] <time_value><NL>

Example:
```
10 DIM Xt$ [100]
20 OUTPUT XXX;":TWAVEFORM:XTIME?"
30 ENTER XXX; Xt$
40 PRINT Xt$
50 END
```

CHARt Subsystem 9

Introduction

The CHARt subsystem provides the commands necessary for programming the HP 16540A,D and 16541A,D 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
- HAXis
- VAXis

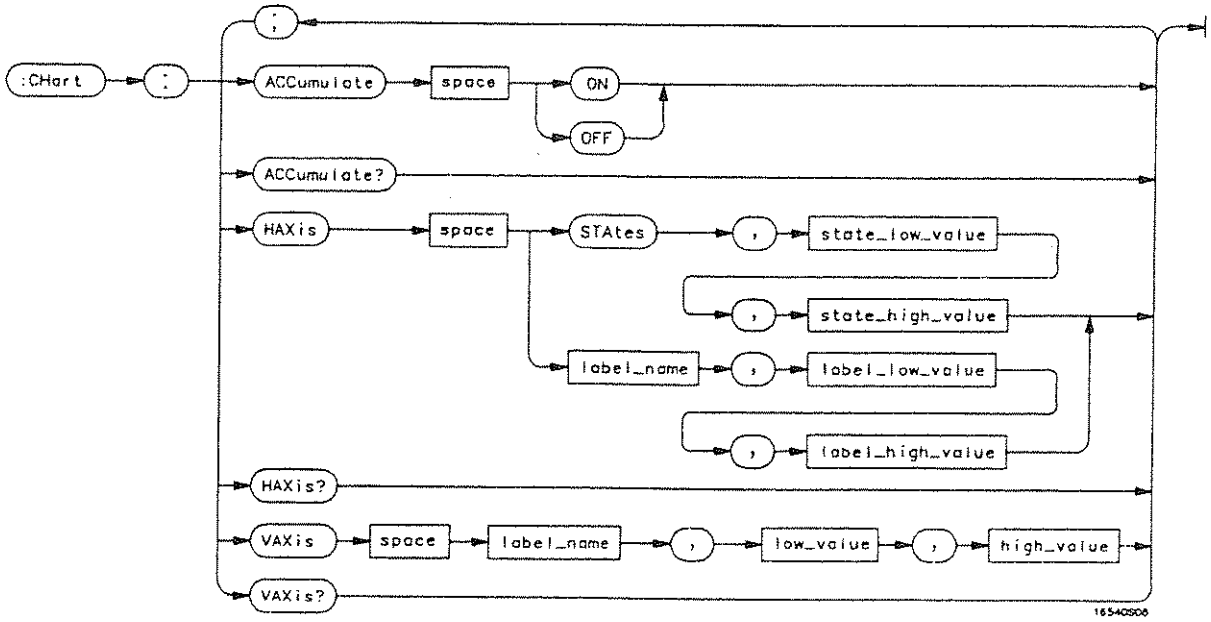


Figure 9-1. CHARt Subsystem Syntax Diagram

state_low_value = *integer from −16384 to + 16384*
state_high_value = *integer from <state_low_value> to + 16384*
label_name = *a string of up to 6 alphanumeric characters*
label_low_value = *string from 0 to* 2^{32} *- 1 (#HFFFF)*
label_high_value = *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)*

CHARt

selector

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

Command Syntax: :CHARt

Example:

```
OUTPUT XXX;":CHART:VAXIS 'A', '0', '9'"
```

ACCumulate command/query

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.

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

Command Syntax: :CHARt:ACCumulate {{ON | 1} | {OFF | 0}}

Example:
```
OUTPUT XXX;":CHART:ACCUMULATE OFF"
```

Query Syntax: CHARt:ACCumulate?

Returned Format: [CHARt:ACCumulate] {0 | 1}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":CHART:ACCUMULATE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

HAXis command/query

The HAXis command allows you to select whether states 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.

Note The shortform for STATES is STA. This is an intentional deviation from the normal truncation rule.

The HAXis query returns the current horizontal axis label and scaling.

Command Syntax: CHARt:HAXis {STAtes,<state_low_value>,<state_high_value> | <label_name>,<label_low_value>,<label_high_value>}

where:

<state_low_value> ::= integer from -16384 to 16384
<state_high_value> ::= integer from <state_low_value> to +16384
<label_name> ::= a string of up to 6 alphanumeric characters
<label_low_value> ::= string from 0 to 2^{32}-1 (#HFFFF)
<label_high_value> ::= string from <label_low_value> to 2^{32}-1 (#HFFFF)

Examples:

```
OUTPUT XXX;":CHART:HAXIS STATES, -100, 100"
OUTPUT XXX;":CHART:HAXIS 'NAUJ', '-511', '511'"
```

Query Syntax: :CHARt:HAXis?

Returned Format: [CHARt:HAXis] {STAtes,<state_low_value>,<state_high_value> | <label_name>,<label_low_value>,<label_high_value>}

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":CHART:HAXIS?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

VAXis command/query

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.

The VAXis query returns the current vertical axis label and scaling.

Command Syntax: CHARt:VAXis <label_name>,<low_value>,<high_value>

where:

<label_name> ::= a string of up to 6 alphanumeric characters
<low_value> ::= string from 0 to 2^{32}-1 (#HFFFF)
<high_value> ::= 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 Syntax: :CHARt:VAXis?

Returned Format: [CHARt:VAXis] <label_name>,<low_value>,<high_value><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":CHART:VAXIS?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

COMPare Subsystem 10

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:

- COPY
- DATA
- CMASk
- RANGe
- RUNTil
- FIND
- MENU
- LINE

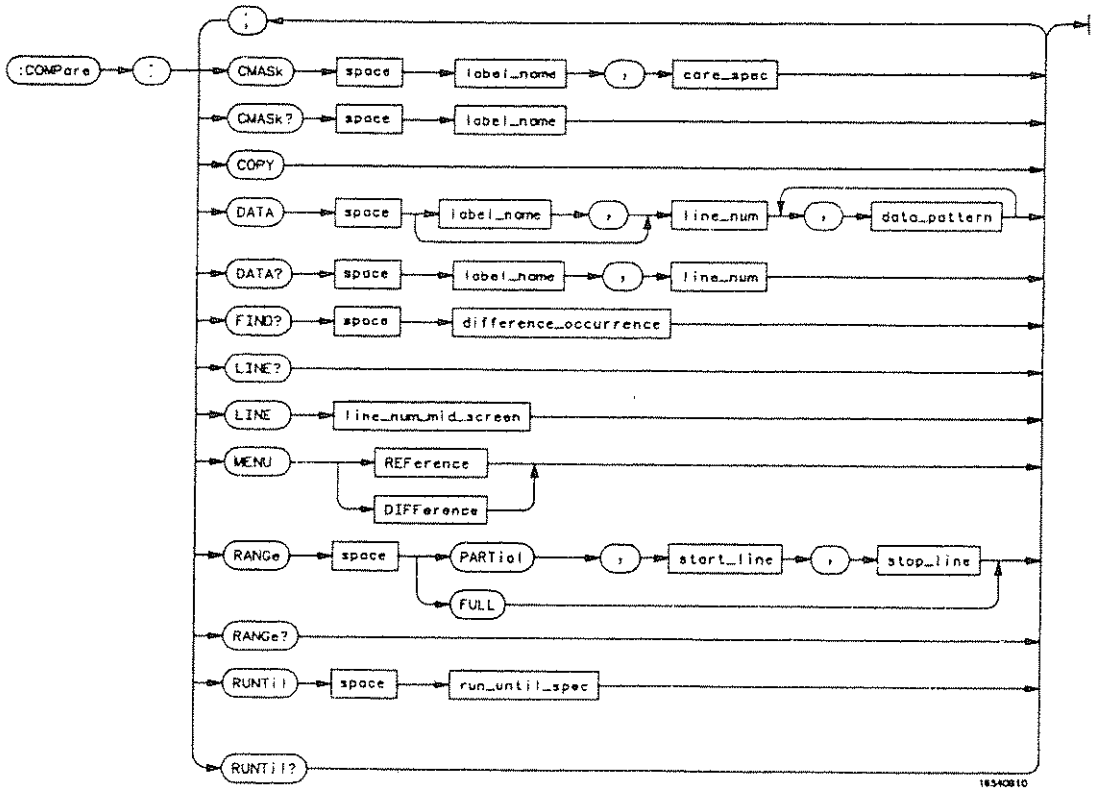


Figure 10-1. COMPare Subsystem Syntax Diagram

label_name = *string of up to 6 characters*
care_spec = *string of characters* "{*|.}..."
* = *care*
. = *don't care*
line_num = *integer from −16384 to + 16384*
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} . . . }"
difference_occurence = *integer from 1 to 16384*
start_line = *integer from −16384 to + 16384*
stop_line = *integer from* <*start_line*> *to + 16384*
run_until_spec = {*OFF|LT,* <*value*> *|GT,* <*value*> *|INRange,* <*value*>, <*value*> *|*
OUTRange, <*value*>, <value> *|EQual|NEQual*}
value = *real number*

COMPare selector

The COMPare selector is used as part of a compound header to access the settings found in the Compare menu.

Command Syntax: :COMPare

Example:

```
OUTPUT XXX;":COMPARE:FIND? 819"
```

CMASk command/query

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."

The CMASk query returns the state of the bits in the channel mask for a given label in the compare listing image.

Command Syntax: :COMPare:CMASk <label_name>,<care_spec>

where:

<label_name> ::= a string of up to 6 alphanumeric characters
<care_spec> ::= string of characters "{*|.}..." (32 characters maximum)
* ::= care
. ::= don't care

Example:

```
OUTPUT XXX;":COMPARE:CMASK 'EKIM', '*.**..**'"
```

Query Syntax: :COMPare:CMASk? <label_name>

Returned Format: [COMPare:CMASk] <label_name>,<care_spec><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":COMPARE:CMASK? 'POD5'"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

COPY command

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.

Command Syntax: :COMPare:COPY

Example:

```
OUTPUT XXX;":COMPARE:COPY"
```

DATA command/query

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 <label_name> 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.

The DATA query returns the value of the compare listing image for a given label and state row.

Command Syntax: :COMPare:DATA {[<label_name>,]<line_num>,<data_pattern>[,<data_pattern>]... }

where:

<label_name> ::= a string of up 6 alphanumeric characters
<line_num> ::= integer from −16384 to +16384
<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'"
```

DATA

Query Syntax: :COMPare:DATA? <label_name>,<line_num>

Returned Format: [COMPare:DATA] <label_name>,<line_num>,<data_pattern><NL>

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

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.

Query Syntax: :COMPare:FIND? <difference_occurrence>

Returned Format: [COMPare:FIND] <difference_occurrence>, <line_number><NL>

where:

<difference_occurrence> ::= integer from 1 to 16384

<line_number> ::= integer from −16384 to +16384

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":COMPARE:FIND? 26"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

RANGe command/query

The RANGe command allows you to define the boundaries for the comparison. The range entered must be a subset of the lines in the aquire memory.

The RANGe query returns the current boundaries for the comparison.

Command Syntax: :COMPare:RANGe {FULL | PARTial,<start_line>,<stop_line>}

where:

<start_line> ::= integer from −16384 to +16384
<stop_line> ::= integer from <start_line> to +16384

Examples:

```
OUTPUT XXX;":COMPARE:RANGE PARTIAL, -511, 512"
OUTPUT XXX;":COMPARE:RANGE FULL"
```

Query Syntax: :COMPare:RANGe?

Returned Format: [COMPare:RANGe] {FULL | PARTial,<start_line>, <stop_line>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":COMPARE:RANGE?"
30 ENTER XXX; String$
40 REM  See if substring "FULL" occurs in response string:
50 PRINT "Range is ";
60 IF POS(String$,"FULL") > 0 THEN PRINT "Full" ELSE PRINT "Partial"
70 END
```

RUNTil

command/query

The RUNTil (run until) command allows you to define a stop condition when the trace 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 TRACe 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 10 ns 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).

Note 

The RUNTil instruction is available in both the LIST and COMPare subsystems.

The RUNTil query returns the current stop criteria for the comparison when running in the repetitive trace mode.

RUNTil

Command Syntax: :COMPare:RUNTil {OFF| LT,<value>|GT,<value>|
INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}

where:

<value> ::= real number from -9E9 to +9E9

Example:
```
OUTPUT XXX;":COMPARE:RUNTIL EQUAL"
```

Query Syntax: :COMPare:RUNTil?

Returned Format: [COMPare:RUNTil] {OFF| LT,<value>|GT,<value>|
INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}<NL>

Example:
```
10 DIM String$[100]
20 OUTPUT XXX;":COMPARE:RUNTIL?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

LINE command/query

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 be highlighted at center screen.

The LINE query returns the line number for the state currently in the box at center screen.

Command Syntax: :COMPare:LINE <line_num_mid_screen>

where:

<line_num_mid_screen> ::= integer from -16384 to +16384

Example:
```
OUTPUT XXX;":COMPare:LINE 0"
```

Query Syntax: :COMPare:LINE?

Returned Format: [:COMPare:LINE] <line_num_mid_screen><NL>

Example:
```
10 DIM Ln$[100]
20 OUTPUT XXX;":COMPare:LINE?"
30 ENTER XXX;Ln$
40 PRINT Ln$
50 END
```

MENU command

The MENU command allows you to switch between the compare listing and the compare difference listing.

Command Syntax: :COMPare:MENU {REFerence|DIFFerence}

Example

```
OUTPUT XXX;":COMPare:MENU DIFFerence"
```

SYMBol Subsystem 11

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 16540A,D and 16541A,D logic analyzer module. The commands in this subsystem are listed below:

- BASE
- PATTern
- RANGe
- REMove
- WIDTh

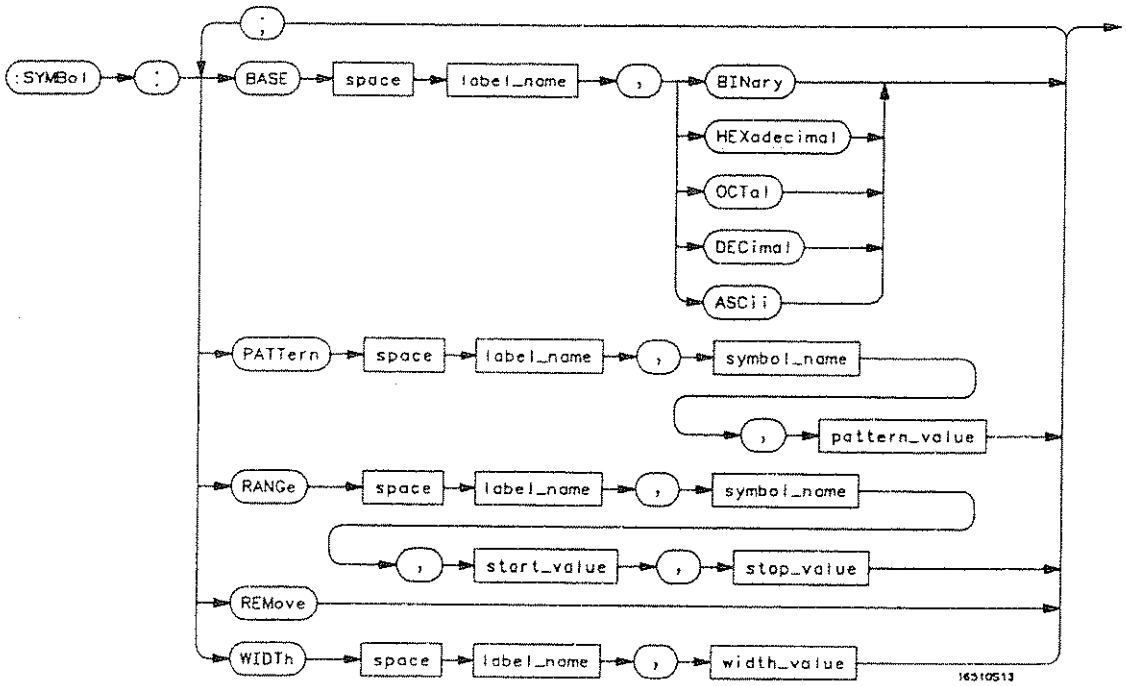


Figure 11-1. SYMBol Subsystem Syntax Diagram

label_name = *string of up to 6 alphanumeric characters*
symbol_name = *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}... }"
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}... }"
width_value = *integer from 1 to 16*

SYMBol selector

The SYMBol selector is used as a part of a compound header to access the commands used to create symbols.

Command Syntax: :SYMBol

Example:

```
OUTPUT XXX;":SYMBOL:BASE 'DATA', BINARY"
```

BASE

command

The BASE command sets the base in which symbols for the specified label will be displayed in the symbol menu. It also specifies the base in which the symbol offsets are displayed when symbols are used.

Note BINary is not available for labels with more than 20 bits assigned. In this case the base will default to HEXadecimal.

Command Syntax: :SYMBol:BASE <label_name>,<base_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<base_value> ::= {BINary | HEXadecimal | OCTal | DECimal | ASCii}

Example:

```
OUTPUT XXX;":SYMBOL:BASE 'DATA',HEXADECIMAL"
```

PATTern command

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; though don't cares cannot be used in a decimal number.

Command Syntax: :SYMBol:PATTern <label_name>,<symbol_name>,<pattern_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<symbol_name> ::= 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

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). You may not use "don't cares" in any base.

Command Syntax: :SYMBol:RANGe <label_name>,<symbol_name>,<start_value>,<stop_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters

<symbol_name> ::= 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

The REMove command deletes all symbols from a specified machine.

Command Syntax: :SYMBol:REMove

Example:

```
OUTPUT XXX;":SYMBOL:REMOVE"
```

WIDTh command

The WIDTh command specifies the width (number of characters) in which the symbol names will be displayed when symbols are used.

Note The WIDTh command does not affect the displayed length of the symbol offset value.

Command Syntax: :SYMBol:WIDTh <label_name>,<width_value>

where:

<label_name> ::= string of up to 6 alphanumeric characters
<width_value> ::= integer from 1 to 16

Example:

```
OUTPUT XXX;":SYMBOL:WIDTH 'DATA',9 "
```

DATA and SETup Commands A

Introduction

The DATA and SETup commands are SYSTem commands that allow you to send and receive block data between the HP 16540A,D and 16541A,D and a controller. Use the DATA instruction to transfer acquired waveform data, and the SETup instruction to transfer instrument configuration data. This appendix explains how to use these two commands.

The DATA and SETup commands are useful for the following tasks:

- Re-loading a configuration into the logic analyzer
- Post-processing data in the controller.

The format and length of the 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.

Note 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.

SYSTem:DATA command/query

The SYSTem:DATA command transmits configuration and acquisition memory data information from the controller to the 100 MHz State analyzer. The SYSTem:DATA query returns this information.

Note 

The data sent by the SYSTem:DATA query reflect the configuration of the analyzer when the last run was performed. Any changes made since then through either front-panel operations or programming commands do not affect the stored configuration.

The block data consists of bytes of information captured by the acquisition chips. The number of bytes depends on the analyzer configuration and the information captured. Care should be taken when transferring the data string into the logic analyzer. Since no parameter checking is performed, out-of-range values could cause instrument lockup.

The **<block data>** parameter can be broken down into a **<block length specifier>** and a variable number of **<section>**s. The **<block length specifier>** always takes the form #8DDDDDDDD. Each D represents a digit (ASCII characters "0" through "9"). The value of the eight digits represents the total length of the block (all sections). For example, if the total length of the block 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, which is composed of a **data preamble** followed by the **acquisition data**. This section will vary in byte length depending on the analyzer configuration.

SYSTem:DATA

Command Syntax: :SYSTem:DATA <block data>

Example: `OUTPUT XXX;":SYSTEM:DATA" <block data>`

where:

<block data>	::= <block length specifier> <section> ...
<block length specifier>	::= #8 <length>
<length>	::= the total length of all sections in byte format (must be represented with 8 digits)
<section>	::= <section header> <section data>
<section header>	::= 16 bytes, described on the following page
<section data>	::= format depends on the analyzer configuration

Note 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 Syntax: :SYSTem:DATA?

Returned Format: [:SYSTem:DATA] <block data> <NL>

HP-IB Example:

```
10 DIM Block$[32000]              ! allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":EOI ON"
40 OUTPUT XXX;":SYSTEM:HEADER OFF"
50 OUTPUT XXX;":SELECT 5"                ! select module
60 OUTPUT XXX;":SYSTEM:DATA?"            ! send data query
70 ENTER XXX USING "#,2A";Specifier$     ! read in #8
80 ENTER XXX USING "#,8D";Blocklength    ! read in block length
90 ENTER XXX USING "-K";Block$           ! read in data
100 END
```

Section Header Description The section header uses bytes 0 through 15. The 16 bytes of the section header are as follows:

0-9 10 bytes - section name. Example, "NAME ".

Note There must be 10 characters used in the section name. In the following example, 4 letters followed by 6 spaces are used for the name. Example, "NAME ".

10 1 byte - not used.

11 1 byte - module ID (40 for the HP 16540A,D).

12-15 4 bytes - number of bytes in <section data> + 16 bytes.

Section Data Description For the SYSTem:DATA command, the <section data> parameter consists of two parts. The first part is 44 bytes of data preamble information. The data preamble documents the analyzer configuration. The second part is a variable number (depending on analyzer configuration) of bytes of acquisition data.

The data preamble and the acquisition data are described in the following two sections.

Data Preamble Description The data preamble gives information describing the analyzer configuration including the type of data captured and where the trace point occurred in the data.

The preamble consists of the following **44 bytes:**

0 - 1 **2 bytes** - Instrument ID.

2 - 3 **2 bytes** - Revision Code.

Note The values stored in the preamble represent the captured data currently stored in this structure and not what the current configuration of the analyzer is. For example, the mode of the data (byte 4) may be STATE, while the current setup of the analyzer is TIMING.

4 **1 byte** - data format mode is represented with the following values:
1 = timing
2 = state
3 = SPA

5 **1 byte** - number of pods in the analyzer configuration. This number will be between 1 and 13.

6 - 7 **2 bytes** - number of valid states. This number will be between 0 and 16384.

8 **1 byte** - real trace point seen:
1 = Yes, a trace point was seen.
0 = No, a trace point was forced.

9 **1 byte** - not used.

10 - 11 **2 bytes** - trace point location. This number will be between 0 and 16384.

12 - 19 **8 bytes** - not used.

20 **1 byte** - tagging mode:
1 = time tags.
0 = state tags.

21 - 33 13 bytes - pod clocking identification. The next 13 bytes will hold a value of either 0 or 1. A "0" identifies a master clock is assigned to the pod, while a "1" identifies a slave clock. If a slave clock is assigned to an expander card, all three pods are assigned a "1".

Bytes 21 through 33 are assigned values from the top according to the number of pods in the configuration. As an example, if there are two expanders (a total of 7 pods with the master), bytes 21 - 27 would have a value of 1 or 0. Bytes 28 - 33 would be don't cares which become "0".

The chart below will help illustrate how bytes 21 - 33 are assigned. The expander location descriptions such as "top" or "2nd bottom" are with respect to their relative position within the mainframe.

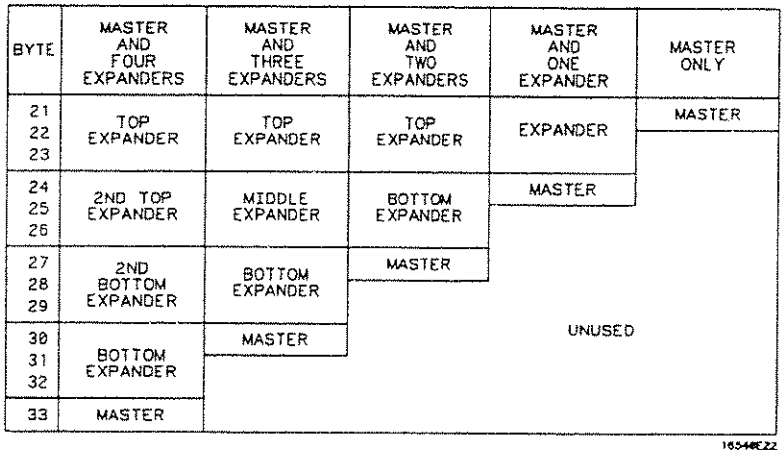


Figure A-1. Clock Identification

34 - 41 8 bytes - sample period. This is a 64 bit binary number that represents the ps/sample. For an example:

The binary number
(0000000000000000000000000000000000000000000000000010011100010000)

Is equal to $(10,000)_{10}$ which is equal to 10 ns/sample.

42 - 43 2 bytes - unused.

Acquisition data Description

The number of bytes in the acquisition data will vary depending on the number of pods in the configuration. However, the total number of bytes can be calculated as:

Byte 44 through (*number of valid states* X *number of pods* X *2*) + *44* - *1*

The order in which bytes of acquisition data is sent to the controller from the analyzer is represented below. MS = most significant and LS = least significant.

```
For  (first valid state) to (last valid state)
        For  (top  expander card) to (bottom expander card)
             MS data byte from pod 3
             LS data byte from pod 3
             MS data byte from pod 2
             LS data byte from pod 2
             MS data byte from pod 1
             LS data byte from pod 1
        Next expander card
        MS data byte from master pod
        LS data byte from master pod
Next valid state
```

Time tag data Description

Immediately following the acquisition data will be the beginning of time tag data. The number of bytes will vary depending on the number of pods in the configuration. However, the total number of bytes can be calculated as:

Byte (*number of valid states* X *number of pods* X *2*) + *44*

Through

Byte (*number of valid states* X *number of pods* X *2*) + *44* + (*number of valid states* X *8*) - *1*

The order in which bytes of time tag data is sent to the controller from the analyzer is represented below. MS = most significant and LS = least significant.

For (first valid state) to (last valid state)
- MS time tagged byte
- 2nd MS time tagged byte
- 3rd MS time tagged byte
- 4th MS time tagged byte
- 4th LS time tagged byte
- 3rd LS time tagged byte
- 2nd LS time tagged byte
- LS time tagged byte

Next valid state

SYSTem:SETup command/query

The SYStem:SETup command configures the logic analyzer module as defined by the block data sent by the controller.

The SYStem:SETup query returns a block of data that contains the current configuration to the controller.

There are three data sections which are always returned:
(These are the strings which would be included in the section header.)

- "CONFIG "
- "DISPLAY1 "
- "BIG_ATTRIB"

Additionally, the following sections may also be included, depending on the user's application:

- "SYMBOLS "
- "SPA_DATA "
- "INVASM "
- "COMPARE "

Note There must be 10 characters in the section name. In the following example, 7 letters with 1 underscore followed by 2 spaces are used for the section name.
Example, "SPA_DATA ".

SYSTem:SETup

Command syntax: :SYStem:SETup <block data>

where:

<block data>	::= <block length specifier> <section> ...
<block length specifier	::= #8 <length>
<length>	::= the total length of all sections in byte format (must be represented with 8 digits)
<section>	::= <section header> <section data>
<section header>	::= 16 bytes in the following format: 10 bytes for the section name 1 byte reserved 1 byte for the module ID code (40 for the logic analyzer) 4 bytes for the length of the section data in bytes
<section data>	::= format depends on analyzer configuration

Note 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;"SETUP"` <block data>

Query Syntax: :SYStem:SETup?

Returned Format: [:SYStem:SETup] <block data> <NL>

HP-IB Example:

```
10 DIM Block$[32000]              ! allocate enough memory for block data
20 DIM Specifier$[2]
30 OUTPUT XXX;":EOI ON"
40 OUTPUT XXX;":SYSTEM:HEADER OFF"
50 OUTPUT XXX;":SELECT 4"                ! select module
60 OUTPUT XXX;":SYSTEM:SETUP?"           ! send setup query
70 ENTER XXX USING "#,2A";Specifier$     ! read in #8
80 ENTER XXX USING "#,8D";Blocklength          ! read in block length
90 ENTER XXX USING "-K";Block$           ! read in data
100 END
```

Index

D

F

H

I

L

M

N

O

P

Q

R

S

T

V

W

X