Programming Reference

HP 16510B
Logic Analyzer Module

for the HP 16500A Logic Analysis System

HEWLETT
PACKARD

Manual Part Number 16510-90914

Printed in the U.S.A. June 1989

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.

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	June 1989	16510-90914

List of Effective Pages

The List of Effective Pages gives the data 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**
All	June 1989

Contents

Chapter 6

STRace Subsystem

Chapter 7

SLISt Subsystem

Programming the HP 16510B 1

Introduction

This manual combined with the *HP 16500A Programming Reference* manual provides you with the information needed to program the HP 16510B logic analyzer module. Each module has its own manual to supplement the mainframe manual since not all mainframes will be configured with the same modules.

About This Manual

This manual is organized in fourteen chapters. The first chapter contains:

- General information and instructions to help you get started
- Mainframe system commands that are frequently used with the logic analyzer module
- HP 16510B Logic Analyzer command tree
- Alphabetic command-to-subsystem directory.

Chapter two contains module level commands. Chapters three through fourteen contain the subsystem commands for the logic analyzer.

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

Error messages for the HP 16510B are included in generic system error messages and are in the *HP 16500A Programming Reference* manual.

Programming the HP 16510B Logic Analyzer

This chapter introduces you to the basic command structure used to program the logic analyzer. Also included is an example program that sets up the timing analyzer for the measurement described in chapter 7 of the *Front-panel Operating Reference* manual for the HP 16510B.

Selecting the Module

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

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

```
:SELect 5
```

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

Programming the Logic Analyzer

A typical logic analyzer program would:

- select the appropriate module
- name a specified analyzer
- specify the analyzer type
- assign pods
- assign labels
- sets pod thresholds
- specify a trigger condition
- set up the display
- specify acquisition type
- start acquisition

The following example program sets up the logic analyzer to make the simple timing measurement described in chapter 7 of the *HP 16510B Front-panel Operation Reference* manual.

Example Program:

```
10  OUTPUT XXX;":SELECT 5"
20  OUTPUT XXX;":MACH1:NAME 'DRAMTEST'"
30  OUTPUT XXX;":MACH1:TYPE TIMING"
40  OUTPUT XXX;":MACH1:ASSIGN 1"
50  OUTPUT XXX;":MACH1:TFOR:LABEL 'RAS',POS,#B0001"
60  OUTPUT XXX;":MACH1:TFOR:LABEL 'CAS',POS,#B0010"
70  OUTPUT XXX;":MACH1:TFOR:THRESHOLD1 TTL"
80  OUTPUT XXX;":MACH1:TTRACE:EDGE 'RAS','F'"
90  OUTPUT XXX;":MACH1:TWAVEFORM:RANGE 2E-6"
100 OUTPUT XXX;":MENU 5,3"
110 OUTPUT XXX;":RMODE SINGLE"
120 OUTPUT XXX;":START"
130 END
```

Note

The three Xs (XXX) after the "OUTPUT" statements in the previous examples 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.

Program Comments

Line 10 selects the logic analyzer in slot E.
Line 20 names machine (analyzer) 1 "DRAMTEST".
Line 30 specifies machine 1 is a timing analyzer.
Line 40 assigns pod 1 to machine 1.
Lines 50 and **60** set up the Timing Format menu by assigning two labels (RAS and CAS), assigning a polarity and channels to the labels.
Line 70 sets the threshold of pod 1 to TTL.
Line 80 selects the trigger condition for the timing analyzer.
Line 90 sets the range to 2 us (10 times s/div).
Line 100 changes the on-screen display to the Timing Waveforms menu.
Line 110 specifies the Single run mode.
Line 120 starts data acquisition.

For more information on the specific logic analyzer commands, refer to chapters 2 through 14 of this manual.

Mainframe Commands

These commands are part of the HP 16500A mainframe system and are mentioned here only for reference. For more 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 16510B logic analyzer is 31. 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,-1,31,2,2,0,0,5
```

A returned string of 12,11,-1,-1,31,2,2,0,0,5 means that an oscilloscope timebase card (ID number 11) is loaded in slot B and the oscilloscope acquisition card (ID number 12) is loaded in slot A. The next two slots (C and D) are empty (-1). Slot E contains a logic analyzer module (ID number 31).

The next group of numbers (2,2,0,0,5) indicate that a two 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 digit (5) in this group indicates a single module card is loaded in slot E. Complete information for the CARDcage query is in the *HP 16500A Programming Reference* manual.

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.

For the HP 16510B Logic Analyzer:

- X,0 - State/Timing Configuration
- X,1 - Format 1
- X,2 - Format 2
- X,3 - Trace 1
- X,4 - Trace 2
- X,5 - Listing/Waveform 1
- X,6 - Listing/Waveform 2
- X,7 - Mixed Display

Note

The menus of an "OFF" machine and Mixed Display are not available when only one analyzer is turned on.

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 will return 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 disc 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 16510B is divided into module level commands and subsystem commands. Module level commands are listed in Chapter 2 and each of the subsystem commands are covered in their individual chapters starting with Chapter 3.

Each of these chapters contains a description of the subsystem, 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.

Figure 1-1 on the following page shows the command tree for the HP 16510B logic analyzer module. The (x) by the SELect command at the top of the tree represents the slot number where the logic analyzer module is installed. The number may range from 1 through 5, representing slots A through E, respectively.

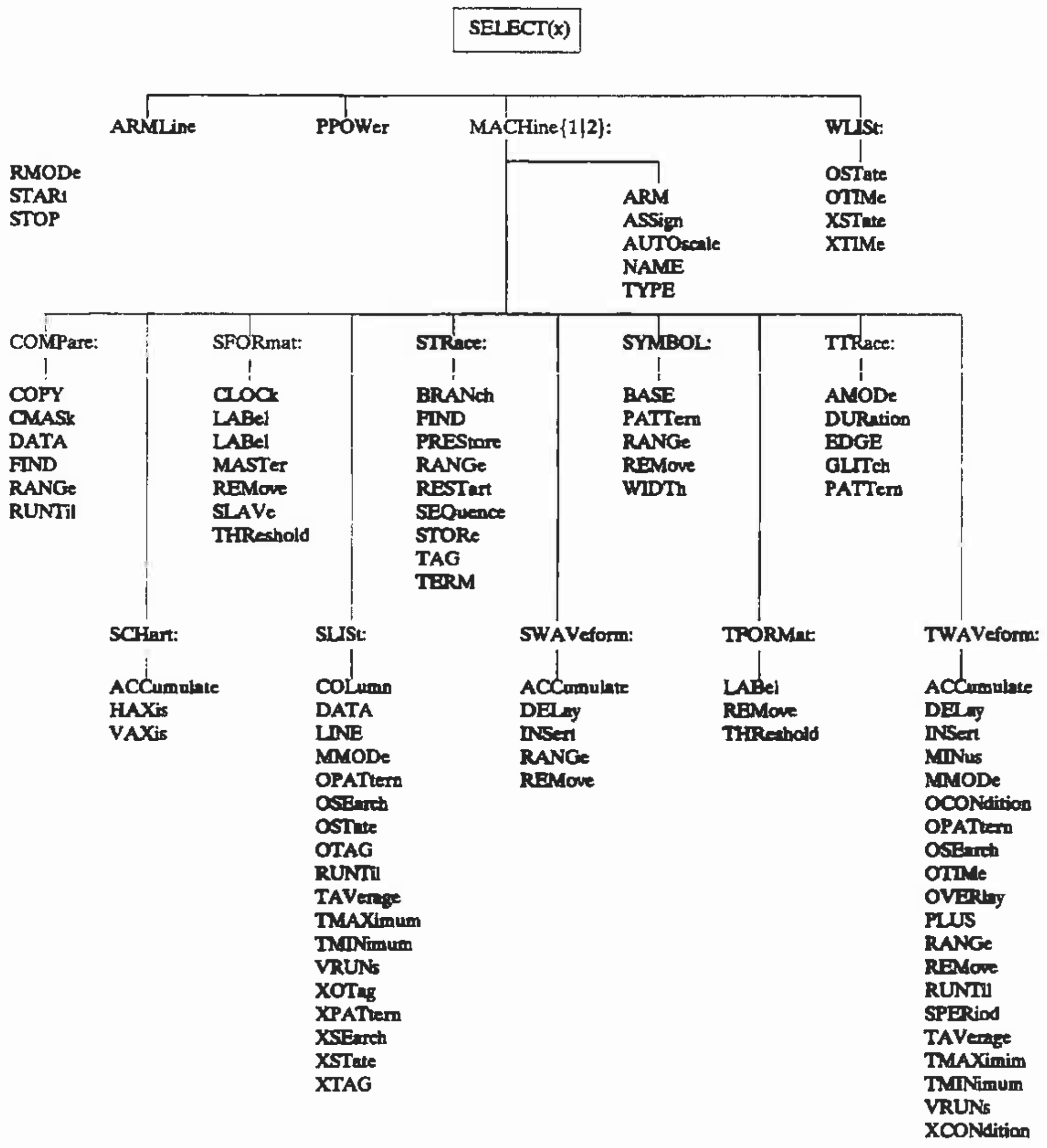


Figure 1-1. HP 16510B Command Tree

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

Command	Where used	Command	Where used
ACCumulate	SCHart, SWAVeform, TWAVeform	PATTern	SYMBol, TTRace
AMODe	TTRace	PLUS	TWAVeform
ARM	MACHine	PPOWer	Module Level
ARMLine	Module Level	PREStore	STRace
ASSign	MACHine	RANGe	COMPare, STRace, SWAVeform, SYMBol, TWAVeform
AUToscale	MACHine	REMove	SFORmat, SWAVeform, SYMBol, TFORmat, TWAVeform
BASE	SYMBol	RESTart	STRace
BRANch	STRace	RUNTil	COMPare, SLISt, TWAVeform
CLOCk	SFORmat	SEQuence	STRace
COLumn	SLISt	SLAVe	SFORmat
COPY	COMPare	SLISt	Module Level
CMASk	COMPare	SPERiod	TWAVeform
CPERiod	SFORmat	STORe	STRace
DATA	COMPare, SLISt	TAG	STRace
DELay	SWAVeform, TWAVeform	TAVerage	SLISt, TWAVeform
DURation	TTRace	TERM	STRace
EDGE	TTRace	THReshold	SFORmat, TFORmat
FIND	COMPare, STRace	TMAXimum	SLISt, TWAVeform
GLITch	TTRace	TMINimum	SLISt, TWAVeform
HAXis	SCHart	TYPE	MACHine
INSert	SWAVeform, TWAVeform	VAXis	SCHart
LABel	SFORmat, TFORmat	VRUNs	SLISt, TWAVeform
LINE	SLISt	WIDTh	SYMBol
MACHine	Module level	WLISt	Module level
MASTer	SFORmat	XCONdition	TWAVeform
MINus	TWAVeform	XOTag	SLISt
MMODe	SLISt, TWAVeform	XOTime	TWAVeform
NAME	MACHine	XPATtern	SLISt, TWAVeform
OCONdition	TWAVeform	XSEarch	SLISt, TWAVeform
OPATtern	SLISt, TWAVeform	XSTate	WLISt, SLISt
OSEarch	SLISt,TWAVeform	XTAG	SLISt
OSTate	SLISt, WLISt	XTIMe	TWAVeform, WLISt
OTAG	SLISt		
OTIMe	TWAVeform, WLISt		
OVERlay	TWAVeform		

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 (see *HP 16500A Programming Reference* manual chapter 6). The Module Event Status Register is enabled by the Module Event Status Enable Register (MESE<N>).

The MESE<N> and MESR<N> instructions are not used in conjunction with the SELect command, so they are not listed in the HP 16510B's command tree.

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

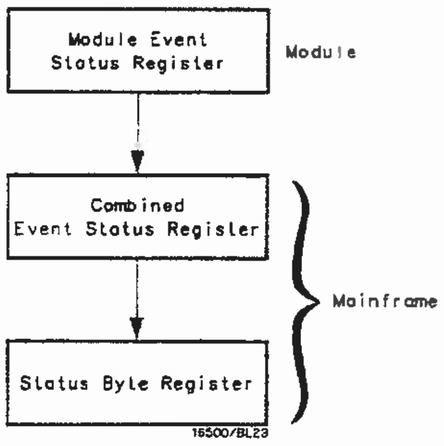


Figure 1-2. Module Status Reporting

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, 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.

The MESE query returns the current setting.

Refer to table 1-2 for information about the Module Event Status register bits, bit weights, and what each bit masks for the module. Complete information for status reporting is in chapter 6 of 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>

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 1-2. 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	Not used
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 1-3 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
```

Table 1-3. 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	Not used
2	4	Not used
1	2	1 = Run until satisified 0 = Run until not satisified
0	1	1 = Measurement complete 0 = Measurement not complete

Module Level Commands 2

Introduction

The logic analyzer Module level commands access the global functions of the HP 16510B logic analyzer module. These commands are:

- ARMLine
- MACHine
- PPOWer
- WLISt

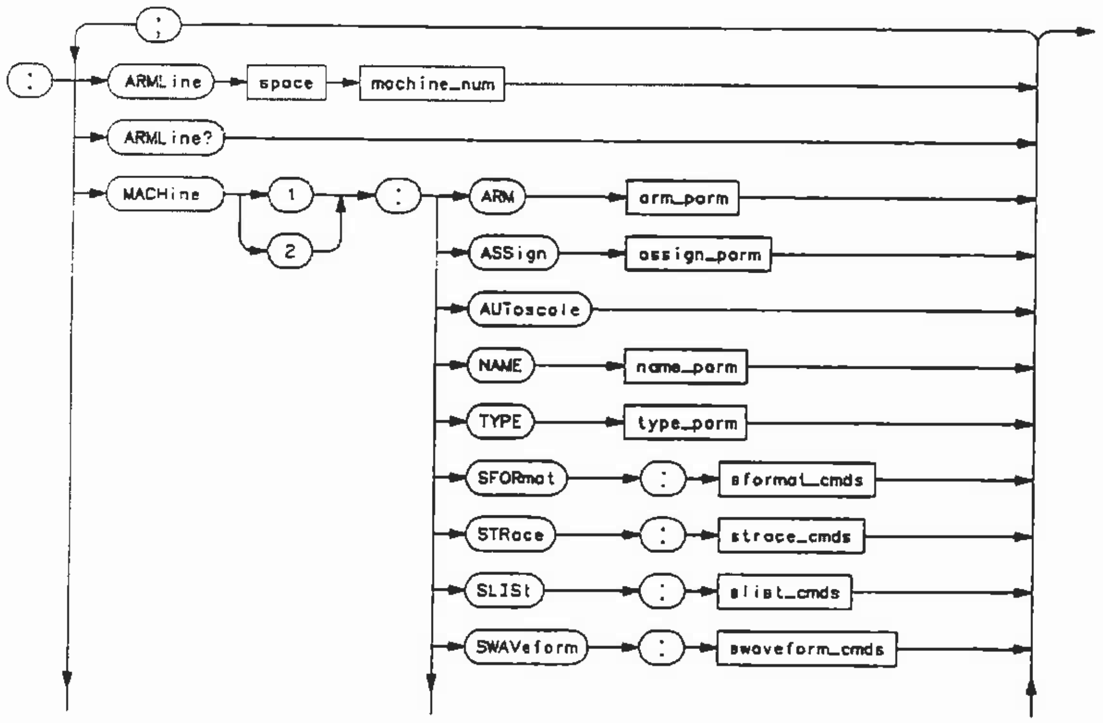
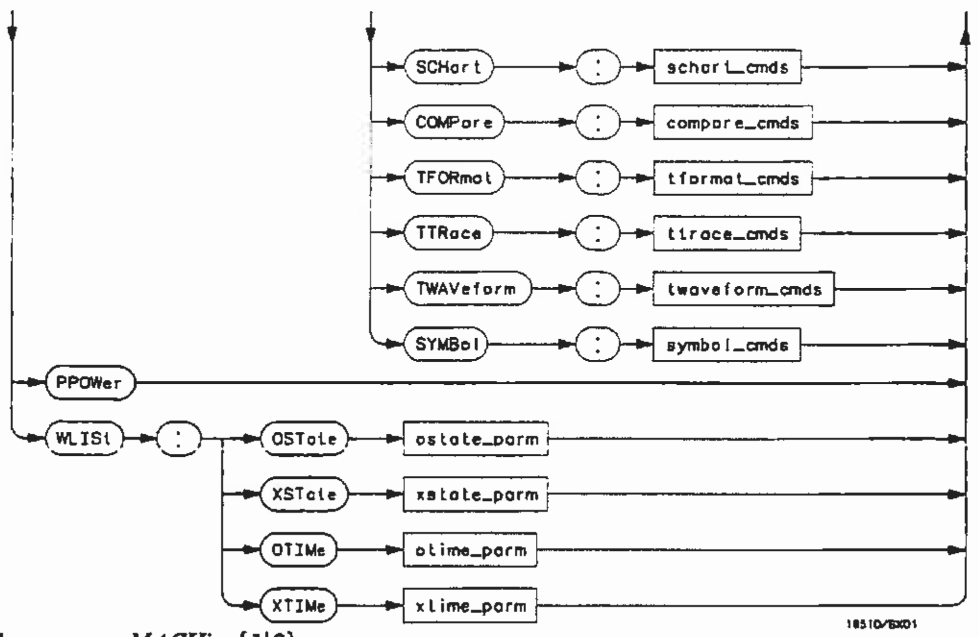


Figure 2-1. Module Level Syntax Diagram



machine_num = *MACHine{1|2}*
arm_parm = *arm parameters (see chapter 3)*
assign_parm = *assignment parameters (see chapter 3)*
name_parm = *name parameters (see chapter 3)*
type_parm = *type parameters (see chapter 3)*
sformat_cmds = *state format subsystem commands (see chapter 5)*
strace_cmds = *state trace subsystem commands (see chapter 6)*
slist_cmds = *state list subsystem commands (see chapter 7)*
swaveform_cmds = *state waveform subsystem commands (see chapter 8)*
schart_cmds = *state chart subsystem commands (see chapter 9)*
compare_cmds = *compare subsystem commands (see chapter 10)*
tformat_cmds = *timing format subsystem commands (see chapter 11)*
ttrace_cmds = *timing trace subsystem commands (see chapter 12)*
twaveform_cmds = *timing waveform subsystem commands (see chapter 13)*
symbol_cmds = *symbol subsystem commands (see chapter 14)*
ostate_parm = *Ostate parameters (see chapter 4)*
xstate_parm = *Xstate parameters (see chapter 4)*
otime_parm = *Otime parameters (see chapter 4)*
xtime_parm = *Xtime parameters (see chapter 4)*

Figure 2-1. Module Level Syntax Diagram (continued)

ARMLine

command/query

The ARMLine command selects which machine generates the arm out signal on the IMB (intermodule bus).

Note This command is only valid when two analyzers are on.

The ARMLine query returns the current IMB arming source.

Command Syntax: :ARMLine {MACHine<N>}

where:

<N> ::= {1|2}

Example:

```
OUTPUT XXX; ":ARMLINE MACHINE1"
```

Query Syntax: :ARMLine?

Returned Format: [:ARMLine]{MACHine<N>}<NL>

Example:

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

MACHine

selector

The MACHine command selects which of the two machines (analyzers) the subsequent commands or queries will refer to. MACHine is also a subsystem containing commands that control the logic analyzer "system" level functions. Examples include pod assignments, analyzer names, and autoscale (see chapter 3 for details).

Command Syntax: :MACHine <N>

where:

<N> ::= {1|2}

Example:

```
OUTPUT XXX;":MACHINE1:NAME 'DRAMTEST'"
```

PPOWer

query

The PPOWer (preprocessor power) query returns the current status of the HP 16510B's high-current limit circuit. If it is functioning properly, 0 is returned. If the current draw is too high, 1 is returned until the problem is corrected and the circuit automatically resets. Sending the query to an HP 16510A module results in -1 being returned.

Query Syntax: :PPOWer?

Returned Format: [:PPOWer] {-1 | 0 | 1}

Example:

```
10 DIM Response$
20 OUTPUT XXX;":PPOWER?"
30 ENTER XXX; Response$
40 PRINT Response$
50 END
```

WLISt

selector

The WLISt selector accesses the commands used to position markers and query marker positions in Timing/State Mixed mode. The details of the WLISt subsystem are in chapter 4 of this manual.

Command Syntax: :WLISt

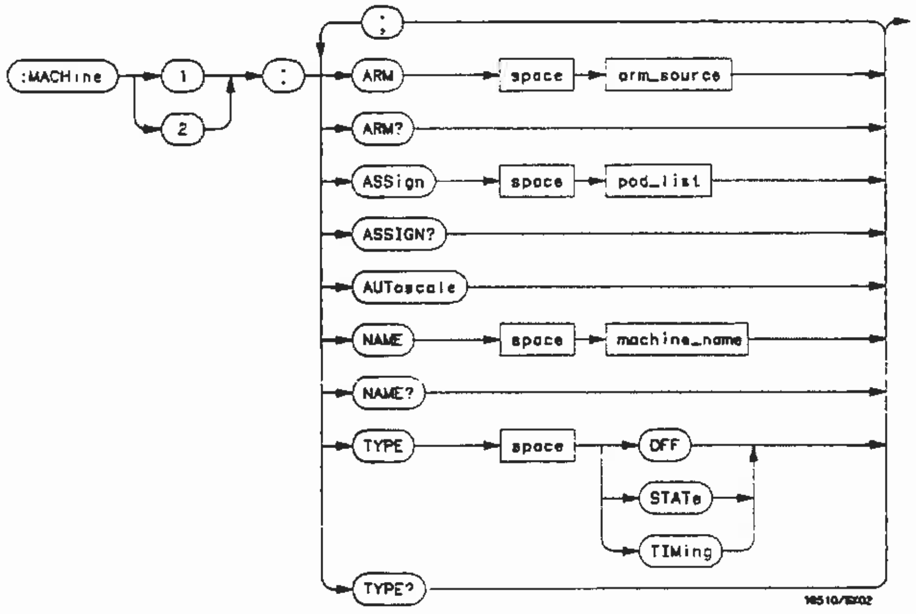
Example:
```
OUTPUT XXX:":WLIST:OTIME 40.0E-6"
```

MACHine Subsystem 3

Introduction

The MACHine subsystem contains the commands available for the State/Timing Configuration menu. These commands are:

- ARM
- ASSign
- AUToscale (Timing Analyzer only)
- NAME
- TYPE



arm_source = {*RUN* | *MACHine* {*1* | *2*}}
pod_list = {*NONE* | <*pod_num*>[, <*pod_num*>]...}
pod_num = {*1* | *2* | *3* | *4* | *5*}
machine_name = *string of up to 10 alphanumeric characters*

Figure 3-1. Machine Subsystem Syntax Diagram

MACHine

selector

The MACHine <N> selector specifies which of the two analyzers (machines) available in the HP 16510B the commands or queries following will refer to. Since the MACHine<N> command is a root level command, it will normally appear as the first element of a compound header.

Command Syntax: :MACHine<N>

where:

<N> ::= {1|2} (the machine number)

Example:

```
OUTPUT XXX; ":MACHINE1:NAME 'DRAMTEST'"
```

ARM

command/query

The ARM command specifies the arming source of the specified analyzer (machine).

Note This command is only valid when two analyzers are on.

The ARM query returns the source that the current analyzer (machine) will be armed by.

Command Syntax: `:MACHine{1|2}:ARM <arm_source>`

where:

`<arm_source>` ::= {RUN|MACHine{1|2}}

Example:

```
OUTPUT XXX;":MACHINE1:ARM MACHINE2"
```

Query Syntax: `:MACHine{1|2}:ARM?`

Returned Format: `[:MACHine{1|2}:ARM] <arm_source><NL>`

Example:

```
10 DIM String$ [100]
20 OUTPUT XXX; ":MACHINE1:ARM?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

ASSign

command/query

The ASSign command assigns pods to a particular analyzer (machine).

The ASSign query returns which pods are assigned to the current analyzer (machine).

Command Syntax: :MACHine{1|2}:ASSign <pod_list>

where:

<pod_list>	::= {NONE | <pod #>[, <pod #>]...}
<pod#>	::= {1|2|3|4|5}

Example:

```
OUTPUT XXX;":MACHINE1:ASSIGN 5, 2, 1"
```

Query Syntax: :MACHine{1|2}:ASSign?

Returned Format: [:MACHine{1|2}:ASSign] <pod_list> <NL>

Example:

```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:ASSIGN?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

AUToscale

command

The AUToscale command causes the current analyzer (machine) to autoscale if the current machine is a timing analyzer. If the current machine is not a timing analyzer, the AUToscale command is ignored.

AUToscale is an Overlapped Command. Overlapped Commands allow execution of subsequent commands while the logic analyzer operations initiated by the Overlapped Command are still in progress. Command overlapping can be avoided by using the *OPC and *WAI commands in conjunction with AUToscale (refer to chapter 5 of the *HP 16500A Programming Reference* manual.)

Note When the AUToscale command is issued, timing analyzer configurations are erased and the other analyzer is turned off.

Command Syntax: `:MACHine{1|2}:AUToscale`

Example:

```
OUTPUT XXX;":MACHINE1:AUTOSCALE"
```

NAME

command/query

The NAME command allows you to assign a name of up to 10 characters to a particular analyzer (machine) for easier identification.

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

Command Syntax: :MACHine{1|2}:NAME <machine_name>

where:

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

Example:

```
OUTPUT XXX;":MACHINE1:NAME 'DRAMTEST'"
```

Query Syntax: :MACHine1|2}:NAME?

Returned Format: [MACHine{1|2}:NAME] <machine name><NL>

Example:

```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:NAME?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TYPE

command/query

The TYPE command specifies what type a specified analyzer (machine) will be. The analyzer types are state or timing. The TYPE command also allows you to turn off a particular machine.

Note Only one timing analyzer can be specified at a time.

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

Command Syntax:	:MACHine{1|2}:TYPE <analyzer type>
where:	
<analyzer type>	::= {OFF|STATe|TIMing}
Example:	`OUTPUT XXX;":MACHINE1:TYPE STATE"`
Query Syntax:	:MACHine{1|2}:TYPE?
Returned Format:	[:MACHine{1|2}:TYPE] <analyzer type><NL>

Example:

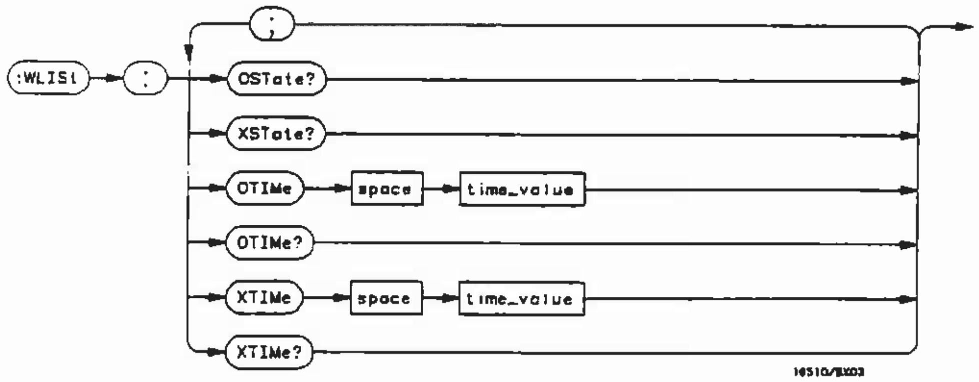
```
10 DIM String$ [100]
20 OUTPUT XXX;":MACHINE1:TYPE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

WLISt Subsystem 4

Introduction

Two commands in the WLISt (Waveforms/LISting) subsystem control the X and O marker placement on the waveforms portion of the Timing/State mixed mode display. These commands are XTIMe and OTIMe. 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 what state (state acquisition memory location) the marked pattern is stored in.

Note In order to have mixed mode, one machine must be a timing analyzer, the other machine must be a state analyzer with time tagging on (use MACHine<N>:STRace:TAG TIME).



time_value = *real number*

Figure 4-1. WLISt Subsystem Syntax Diagram

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 Mode menu. Since the WLISt command is a root level command, it will always appear as the first element of a compound header.

Note The WLISt subsystem is only available when one state analyzer (with time tagging on) and one timing analyzer are specified.

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 mode display. 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 mode 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
```

SFORmat Subsystem 5

Introduction

The SFORmat subsystem contains the commands available for the State Format menu in the HP 16510B logic analyzer module. These commands are:

- CLOCk
- CPERiod
- LABel
- MASTer
- REMove
- SLAVe
- THReshold

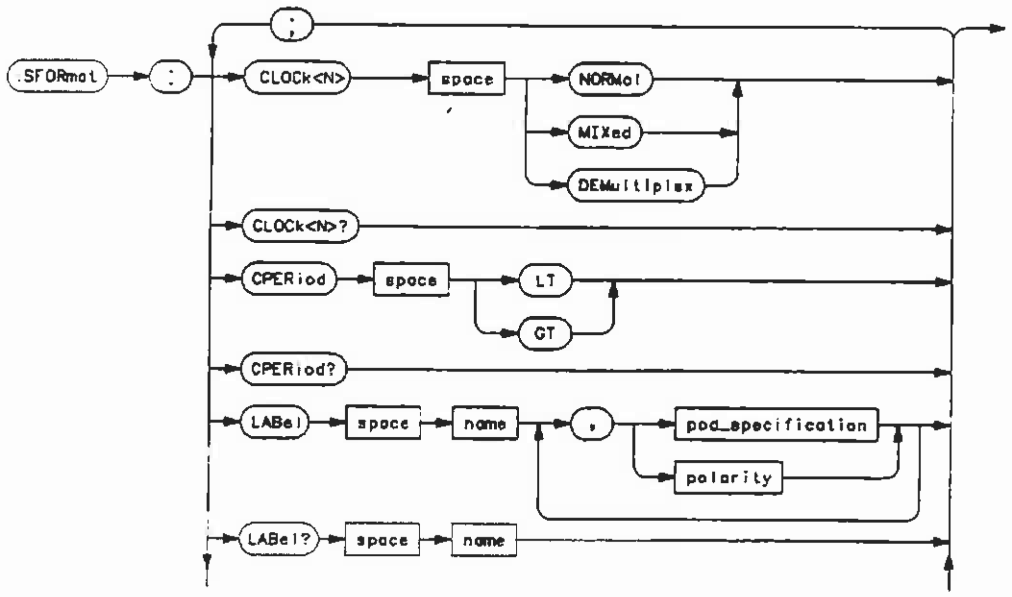
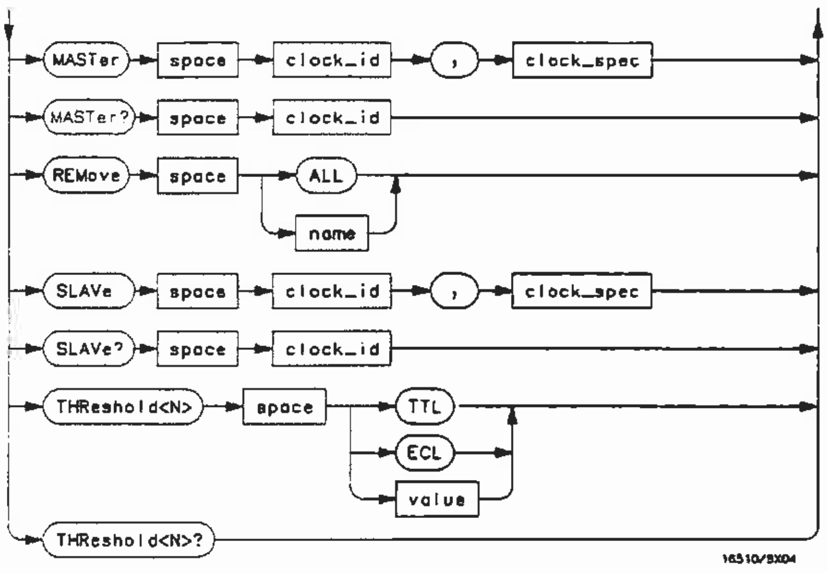


Figure 5-1. SFORmat Subsystem Syntax Diagram



<N> = {*1* | *2* | *3* | *4* | *5*}
GT = *Greater Than 60 ns*
LT = *Less Than 60 ns*
name = *string of up to 6 alphanumeric characters*
polarity = {*POSitive* | *NEGative*}
pod_specification = *format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)*
clock_id = {*J* | *K* | *L* | *M* | *N*}
clock_spec = {*OFF* | *RISing* | *FALLing* | *BOTH* | *LOW* | *HIGH*}
value = *voltage (real number) -9.9 to +9.9*

Figure 5-1. SFORmat Subsystem Syntax Diagram (continued)

SFORmat

selector

The SFORmat (State Format) selector is used as a part of a compound header to access the settings in the State Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SFORmat

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER J, RISING"
```

CLOCk

command/query

The CLOCk command selects the clocking mode for a given pod when the pod is assigned to the state analyzer. When the NORMal option is specified, the pod will sample all 16 channels on the master clock. When the MIXed option is specified, the upper 8 bits will be sampled by the master clock and the lower 8 bits will be sampled by the slave clock. When the DEMultiplex option is specified, the lower 8 bits will be sampled on the slave clock and then sampled again on the master clock. The master clock always follows the slave clock when both are used.

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

Command Syntax: :MACHine{1|2}:SFORmat:CLOCk<N> <clock_mode>

where:

<N>	::= {1|2|3|4|5}
<clock_mode>	::= {NORMal | MIXed | DEMultiplex}

Example:

```
OUTPUT XXX;":MACHINE1:SFORMAT:CLOCK2 NORMAL"
```

Query Syntax: :MACHine{1|2}:SFORmat:CLOCk<N>?

Returned Format: [:MACHine{1|2}:SFORmat:CLOCK<N>] <clock_mode><NL>

Example:

```
10 DIM String$ [100]
20 OUTPUT XXX; ":MACHINE1:SFORMAT:CLOCK2?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

CPERiod

command/query

The CPERiod command allows you to set the state analyzer for input clock periods of greater than or less than 60 ns. Either LT or GT can be specified. LT signifies a state input clock period of less than 60 ns, and GT signifies a period of greater than 60 ns.

Because count tagging requires a minimum clock period of 60 ns, the CPERiod and TAG commands are interrelated (the TAG command is in the STRace subsystem). When the clock period is set to Less Than, count tagging is turned off. When count tagging is set to either state or time, the clock period is automatically set to Greater Than.

The CPERiod query returns the current setting of clock period.

Command Syntax: :MACHine{1|2}:SFORmat:CPERiod {LT|GT}

where:

GT	::= greater than 60 ns
LT	::= less than 60 ns

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:CPERIOD GT"
```

Query Syntax: :MACHine{1|2}:SFORmat:CPERiod?

Returned Format: [:MACHine{1|2}:SFORmat:CPERiod] {GT|LT}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:CPERIOD?
30 ENTER XXX; String$
40 PRINT String$
50 END
```

LABel

command/query

The LABel command allows you to specify polarity and 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.

The order of the pod-specification parameters is significant. The first one listed will match the highest-numbered pod assigned to the machine you're using. Each pod specification after that is assigned to the next-highest-numbered pod. This way they match the left-to-right descending order of the pods you see on the Format display. Not including enough pod specifications results in the lowest-numbered pod(s) being assigned a value of zero (all channels excluded). If you include more pod specifications than there are pods for that machine, the extra ones will be ignored. However, an error is reported anytime more than five pod specifications are listed.

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

Since 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: :MACHine{1|2}:SFORmat:LABel <name>[, {<polarity> | <assignment>}]...

where:

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

Examples:

```
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'STAT', POSITIVE, 65535,127,40312"
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'SIG 1', 64, 12, 0, 20, NEGATIVE"
OUTPUT XXX;":MACHINE1:SFORMAT:LABEL 'ADDR', NEG, #B0011110010101010"
```

Query Syntax: :MACHine{1|2}:SFORmat:LABel? <name>

Returned Format: [:MACHine{1|2}:SFORmat:LABel] <name>,<polarity>[, <assignment>]...<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:LABEL? 'DATA'"
30 ENTER XXX String$
40 PRINT String$
50 END
```

MASTer

command/query

The MASTer clock command allows you to specify a master clock for a given machine. The master clock is used in all clocking modes (Normal, Mixed, and Demultiplexed). Each command deals with only one clock (J,K,L,M,N); therefore, a complete clock specification requires five commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed. Level specifications (LOW or HIGH) are ANDed.

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

Note At least one clock edge must be specified.

Command Syntax: :MACHine{1|2}:SFORmat:MASTer <clock_id>,<clock_spec>

where:

<clock_id>	::= {J|K|L|M|N}
<clock_spec>	::= {OFF|RISing|FALLing|BOTH|LOW|HIGH}

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER J, RISING"
```

Query Syntax: :MACHine{1|2}:SFORmat:MASTer? <clock_id>

Returned Format: [:MACHine{1|2}:SFORmat:MASTer] <clock_id>,<clock_spec><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:MASTER?<clock_id>"
30 ENTER XXX String$
40 PRINT String$
50 END
```

REMove

command

The REMove command allows you to delete all labels or any one label for a given machine.

Command Syntax: :MACHine{1|2}:SFORmat:REMove {<name>|ALL}

where:

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

Examples:

```
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE2:SFORMAT:REMOVE ALL"
```

SLAVe

command/query

The SLAVe clock command allows you to specify a slave clock for a given machine. The slave clock is only used in the Mixed and Demultiplexed clocking modes. Each command deals with only one clock (J,K,L,M,N); therefore, a complete clock specification requires five commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed. Level specifications (LOW or HIGH) are ANDed.

Note When slave clock is being used at least one edge must be specified.

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

Command Syntax: `:MACHine{1|2}:SFORmat:SLAVe <clock_id>,<clock_spec>`

where:

<clock_id>	::= {J|K|L|M|N}
<clock_spec>	::= {OFF|RISing|FALLing|BOTH|LOW|HIGH}

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE J, RISING"
```

Query Syntax: `:MACHine{1|2}:SFORmat:SLAVe?<clock_id>`

Returned Format: `[:MACHine{1|2}:SFORmat:SLAVe] <clock_id>,<clock_spec><NL>`

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE? <clock_id>"
30 ENTER XXX String$
40 PRINT String$
50 END
```

THReshold

command/query

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL, or a specific voltage from -9.9V to +9.9V in 0.1 volt increments.

Note The pod thresholds of pods 1, 2 and 3 can be set independently. The pod thresholds of pods 4 and 5 are slaved together; therefore when you set the threshold on either pod 4 or 5, both thresholds will be changed to the specified value.

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

Command Syntax: :MACHine{1|2}:SFORmat:THReshold<N> {TTL|ECL|<value>}

where:

<N>	::= pod number {1|2|3|4|5}
<value>	::= voltage (real number) -9.9 to +9.9
TTL	::= default value of +1.6V
ECL	::= default value of -1.3V

Example:

```
OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD1 4.0"
```

Query Syntax: :MACHine{1|2}:SFORmat:THReshold<N>?

Returned Format: [:MACHine{1|2}:SFORmat:THReshold<N>] <value><NL>

Example:

```
10 DIM Value$ [100]
20 OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD4?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

STRace Subsystem 6

Introduction

The STRace subsystem contains the commands available for the State Trace menu in the HP 16510B logic analyzer module. The STRace subsystem commands are:

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

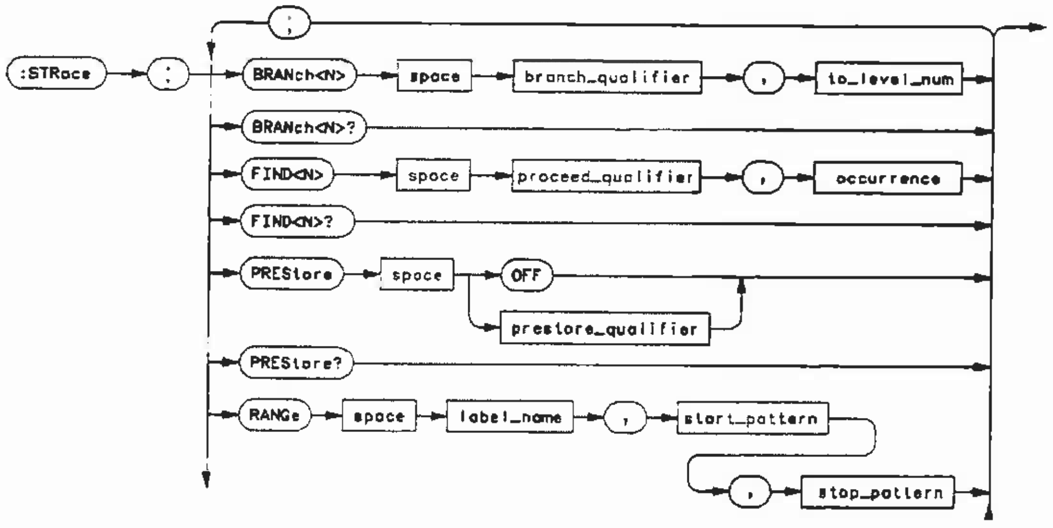


Figure 6-1. STRace Subsystem Syntax Diagram

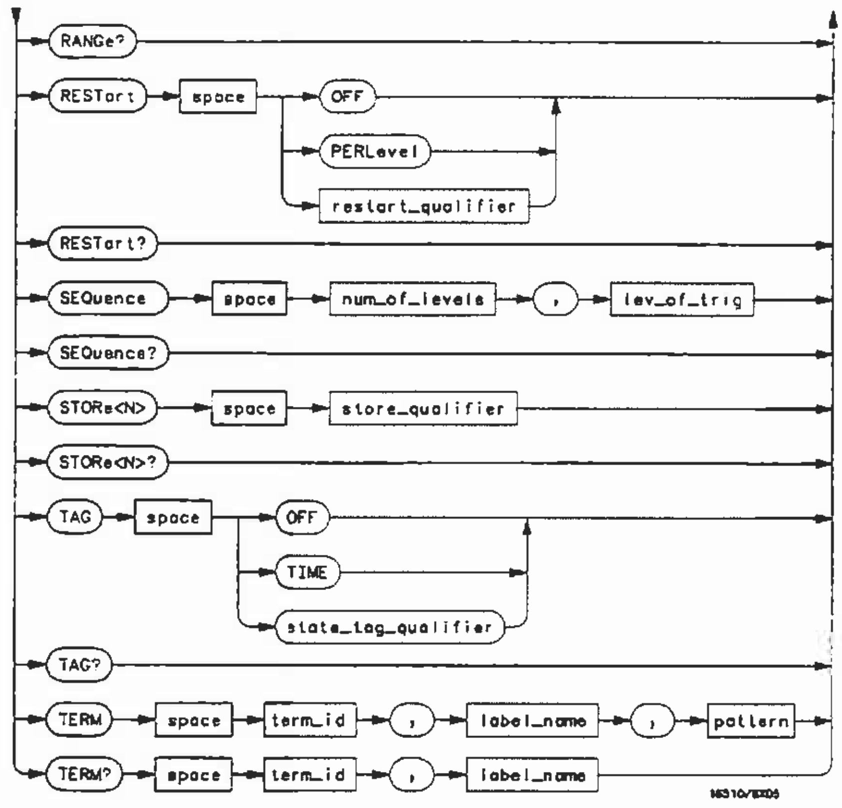


Figure 6-1. STRace Subsystem Syntax Diagram (continued)

branch_qualifier = *<qualifier>*
to_lev_num = *integer from 1 to trigger level when <N> is less than or equal to the trigger level, or from (trigger level + 1) to <num_of_levels> when <N> is greater than the trigger level*
proceed_qualifier = *<qualifier>*
occurrence = *number from 1 to 65535*
prestore_qual = *<qualifier>*
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 8 when ARM is RUN or from 2 to 7 otherwise*
lev_of_trig = *integer from 1 to (number of existing sequence levels - 1)*
store_qualifier = *<qualifier>*
state_tag_qualifier = *<qualifier>*
term_id = {A|B|C|D|E|F|G|H}
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> | (expression1[{AND|OR} <expression2>]) |
(expression2[{AND|OR} <expression1>]) }
any_term = {<or_term1> | <and_term1> | <or_term2> | and_term2}
expression1 = {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
expression2 = {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
or_term1 = {A|B|C|D|INRange|OUTRange}
and_term1 = {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
or_term2 = {E|F|G|H}
and_term2 = {NOTE|NOTF|NOTG|NOTH}

Figure 6-1. STRace Subsystem Syntax Diagram (continued)

STRace selector

The STRace (State Trace) selector is used as a part of a compound header to access the settings found in the State Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:STRace

Example:

```
OUTPUT XXX;":MACHINE1:STRACE:TAG TIME"
```

BRANch

command/query

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

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

The terms used by the branch qualifier (A through H) 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 could manually enter through the touchscreen. 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;":MACHINE1:STRACE:BRANCH1 (C OR D AND F OR G), 1"
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 ((C OR D) AND (F OR G)), 1"
```

Figure 6-2 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 either on or before the trigger level, or they must both be after the trigger level . The trigger level is determined through the SEQuence command.

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

BRANch

Command Syntax: :MACHine{1|2}:STRace:BRANch<N> <branch_qualifier>,<to_level_number>

where:

<N>	::= an integer from 1 to <number_of_levels>
<to_level_number>	::= integer from 1 to trigger level, when <N> is less than or equal to the trigger level or from (trigger level + 1) to <number_of_levels>, when <N> is greater than the trigger level
<number_of_levels>	::= integer from 2 to the number of existing sequence levels (maximum 8)
<branch_qualifier>	::= { ANYState | NOSTate | <any_term> | (<expression1>[{AND|OR} <expression2>]) | (<expression2>[{AND|OR} <expression1>]) }
<any_term>	::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
<expression1>	::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
<expression2>	::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
<or_term1>	::= {A|B|C|D|INRange|OUTRange}
<and_term1>	::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
<or_term2>	::= {E|F|G|H}
<and_term2>	::= {NOTE|NOTF|NOTG|NOTH}

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 ANYSTATE, 3"
OUTPUT XXX;":MACHINE2:STRACE:BRANCH2 A, 7"
OUTPUT XXX;":MACHINE1:STRACE:BRANCH3 ((A OR B) OR NOTG), 1"
```

Query Syntax :MACHine{1|2}:STRace:BRANch<N>?

Returned Format: [:MACHine{1|2}:STRace:BRANch<N>] <branch_qualifier>,<to_level_num><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:BRANCH3?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

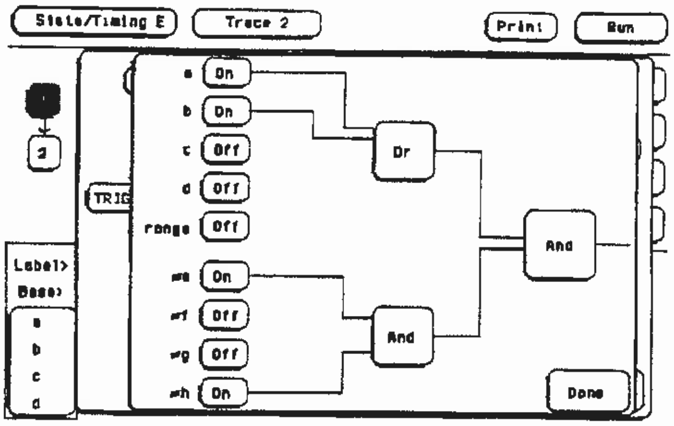


Figure 6-2. Complex qualifier

Figure 6-2 (above) is a front panel representation of the complex qualifier (a Or b) **And** (≠ e **And** ≠ h). The following example would be used to specify this complex qualifier.

```
OUTPUT XXX;":MACHINE1:STRACE:BRANCH1 ((A OR B) AND (NOTE AND NOTH)), 2"
```

Note

Terms **A** through **D** and **RANGE** must be grouped together and terms **E** through **H** must be grouped together. In the first level, terms from one group may not be mixed with terms from the other. For example, the expression ((A OR INRANGE) AND (C OR H)) is not allowed because the term C cannot be specified in the **E** through **H** group.

Keep in mind that, at the first level, the operator you use determines which terms are available. When AND is chosen, only the **NOT** terms may be used. Either AND or OR may be used at the second level to join the two groups together. It is acceptable for a group to consist of a single term. Thus, an expression like (B AND G) is legal, since the two operands are both simple terms from separate groups.

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 will proceed 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 H 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 6-2 for a detailed example.

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

Command Syntax:

```
:MACHine{1|2}:STRace:FIND<N>   <proceed_qualifier>,<occurrence>
```

where:

```
                <N>  ::= integer from 1 to the number of existing sequence levels (maximum 8)
       <occurrence>  ::= integer from 1 to 65535
<proceed_qualifier>  ::= { ANYState | NOSTate | <any_term> |
                     (<expression1>[{AND|OR} <expression2>]) |
                     (<expression2>[{AND|OR} <expression1>]) }
         <any_term>  ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
      <expression1>  ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
      <expression2>  ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
         <or_term1>  ::= {A|B|C|D|INRange|OUTRange}
        <and_term1>  ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
         <or_term2>  ::= {E|F|G|H}
        <and_term2>  ::= {NOTE|NOTF|NOTG|NOTH}
```

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:FIND1 ANYSTATE, 1"
OUTPUT XXX;":MACHINE1:STRACE:FIND2 A, 512"
OUTPUT XXX;":MACHINE1:STRACE:FIND3 ((NOTA AND NOTB) OR 6), 1"
```

Query Syntax: :MACHine{1|2}:STRace:FIND4?

Returned Format: [:MACHine{1|2}:STRace:FIND<N>] <proceed_qualifier>,<occurrence><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:FIND<N>?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

PREStore

command/query

The PREStore command turns the prestore feature on and off. It also defines the qualifier required to prestore only selected states. The terms A through H 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.

A detailed example is provided in figure 6-2.

The PREStore query returns the current prestore specification.

Command Syntax:

```
:MACHine{1|2}:STRace:PREStore  {OFF | <prestore_qualifier>}
```

where:

```
<prestore_qualifier>  ::= { ANYState | NOSTate | <any_term> |
                          (<expression1> [{AND|OR} <expression2>]) |
                          (<expression2> [{AND|OR} <expression1>]) }
          <any_term>  ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
       <expression1>  ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
       <expression2>  ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
          <or_term1>  ::= {A|B|C|D|INRange|OUTRange}
         <and_term1>  ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
          <or_term2>  ::= {E|F|G|H}
         <and_term2>  ::= {NOTE|NOTF|NOTG|NOTH}
```

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE OFF"
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE ANYSTATE"
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE (E)"
OUTPUT XXX;":MACHINE1:STRACE:PRESTORE (A OR B OR D OR F OR H)"
```

Query Syntax: :MACHine{1|2}:STRace:PREStore?

Returned Format: [:MACHine{1|2}:STRace:PREStore] {OFF|<prestore_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:PRESTORE?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

RANGe

command/query

The RANGe command allows you to specify a range recognizer term in the specified machine. Since a range can only be defined across one label and, since 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 Since 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 one of the range recognizers' end points. Don't cares are not allowed in the end point pattern specifications. Since only one range recognizer exists, it is always used by the first state machine defined.

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

Note When two state analyzers are on, the RANGe term is not available in the second state analyzer assigned and there are only 4 pattern recognizers per analyzer.

RANGe

Command Syntax: :MACHine{1|2}:STRace: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;":MACHINE1:STRACE:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":MACHINE1:STRACE:RANGE 'ABC', '#B00001111', '#HCF' "
```

Query Syntax: :MACHine{1|2}:STRace:RANGe?

Returned Format: [:MACHine{1|2}:STRAce:RANGe] <label_name>,<start_pattern>,<stop_pattern><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:RANGE?"
30 ENTER XXX;String$
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 H 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.

A detailed example is provided in figure 6-2.

The RESTart query returns the current restart specification.

Command Syntax: :MACHine{1|2}:STRace:RESTart {OFF | PERLevel | <restart_qualifier>}

where:

<restart_qualifier>	::= { ANYState | NOSTate | <any_term> | (<expression1> [{AND|OR} <expression2>]) | (<expression2> [{AND|OR} <expression1>]) }
<any_term>	::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
<expression1>	::= {<or_term1> [OR <or_term1>]... | <and_term1> [AND <and_term1>]...}
<expression2>	::= {<or_term2> [OR <or_term2>]... | <and_term2> [AND <and_term2>]...}
<or_term1>	::= {A|B|C|D|INRange|OUTRange}
<and_term1>	::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
<or_term2>	::= {E|F|G|H}
<and_term2>	::= {NOTE|NOTF|NOTG|NOTH}

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:RESTART OFF"
OUTPUT XXX;":MACHINE1:STRACE:RESTART PERLEVEL"
OUTPUT XXX;":MACHINE1:STRACE:RESTART (NOTA AND NOTB AND INRANGE)"
OUTPUT XXX;":MACHINE1:STRACE:RESTART (B OR (NOTE AND NOTF))"
```

Query Syntax: :MACHine{1|2}:STRace:RESTart?

Returned Format: [:MACHine{1|2}:STRace:RESTart] {OFF | PERLevel | <restart_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:RESTART?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SEQuence

command/query

The SEQuence command redefines the state analyzer trace sequence. First, it deletes the current trace sequence. Then it inserts the number of levels specified, with default settings, and assigns the trigger to be at a specified sequence level. The number of levels can be between 2 and 8 when the analyzer is armed by the RUN key. When armed by the IMB or the other machine, a level is used by the arm in; therefore, only seven levels are available in the sequence.

The SEQuence query returns the current sequence specification.

Command Syntax: :MACHine{1|2}:STRace:SEQuence <number_of_levels>,<level_of_trigger>

where:

<number_of_levels>	::= integer from 2 to 8 when ARM is RUN or from 2 to 7 otherwise
<level_of_trigger>	::= integer from 1 to (number of existing sequence levels - 1)

Example:

```
OUTPUT XXX;":MACHINE1:STRACE:SEQUENCE 4,3"
```

Query Syntax: :MACHine{1|2}:STRace:SEQuence?

Returned Format: [:MACHine{1|2}:STRace:SEQuence] <number_of_levels>,<level_of_trigger> <NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:SEQUENCE?"
30 ENTER XXX;String$
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 will actually be 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 H 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.

A detailed example is provided in figure 6-2.

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

Command Syntax:

```
:MACHine{1|2}:STRace:STORe <N>   <store_qualifier>
```

where:

```
            <N>   ::= an integer from 1 to the number of existing sequence levels (maximum 8)
<store_qualifier> ::= { ANYState | NOSTate | <any_term> |
                      (<expression1>[{AND|OR} <expression2>]) |
                      (<expression2>[{AND|OR} <expression1>]) }
     <any_term>   ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
   <expression1>  ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
   <expression2>  ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
     <or_term1>   ::= {A|B|C|D|INRange|OUTRange}
    <and_term1>   ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
     <or_term2>   ::= {E|F|G|H}
    <and_term2>   ::= {NOTE|NOTF|NOTG|NOTH}
```

STORe

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:STORE1 ANYSTATE"
OUTPUT XXX;":MACHINE1:STRACE:STORE2 OUTRANGE"
OUTPUT XXX;":MACHINE1:STRACE:STORE3 (NOTC AND NOTD AND NOTH)"
```

Query Syntax: :MACHine{1|2}:STRace:STORe<N>?

Returned Format: [:MACHine{1|2}:STRace:STORe<N>] <store_qualifier><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:STORE4?"
30 ENTER XXX;String$
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 H are defined by the TERM command. The terms INRange and OUTRange are defined 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. A detailed example is provided in figure 6-2.

Because count tagging requires a minimum clock period of 60 ns, the CPERiod and TAG commands are interrelated (the CPERiod command is in the SFORmat subsystem). When the clock period is set to Less Than, count tagging is turned off. When count tagging is set to either state or time, the clock period is automatically set to Greater Than.

The TAG query returns the current count tag specification.

Command Syntax:

```
:MACHine{1|2}:STRace:TAG  {OFF | TIME | <state_tag_qualifier>}
```

where:

```
<state_tag_qualifier>  ::= { ANYState | NOSTate | <any_term> |
                           (<expression1>[{AND|OR} <expression2>]) |
                           (<expression2>[{AND|OR} <expression1>]) }
           <any_term>  ::= {<or_term1> | <and_term1> | <or_term2> | <and_term2>}
        <expression1>  ::= {<or_term1>[OR <or_term1>]... | <and_term1>[AND <and_term1>]...}
        <expression2>  ::= {<or_term2>[OR <or_term2>]... | <and_term2>[AND <and_term2>]...}
           <or_term1>  ::= {A|B|C|D|INRange|OUTRange}
          <and_term1>  ::= {NOTA|NOTB|NOTC|NOTD|INRange|OUTRange}
           <or_term2>  ::= {E|F|G|H}
          <and_term2>  ::= {NOTE|NOTF|NOTG|NOTH}
```

TAG

Examples:

```
OUTPUT XXX;":MACHINE1:STRACE:TAG OFF"
OUTPUT XXX;":MACHINE1:STRACE:TAG TIME"
OUTPUT XXX;":MACHINE1:STRACE:TAG (INRANGE OR NOTF)"
OUTPUT XXX;":MACHINE1:STRACE:TAG ((INRANGE OR A) AND E)"
```

Query Syntax: :MACHine{1|2} :STRace:TAG?

Returned Format: [:MACHine{1|2}:STRace:TAG] {OFF|TIME|<state_tag_qualifier>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:TAG?"
30 ENTER XXX;String$
40 PRINT String$
50 END
```

TERM

command/query

The TERM command allows you to a specify a pattern recognizer term in the specified machine. 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.

When a single state machine is on, all eight terms (A through H) are available in that machine. When two state machines are on, terms A through D are used by the first state machine defined, and terms E through H are used by the second state machine defined.

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

Command Syntax: :MACHine{1|2}:STRace:TERM <term_id>,<label_name>,<pattern>

where:

<term_id>	::= {A|B|C|D|E|F|G|H}
<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;":MACHINE1:STRACE:TERM A,'DATA','255' "
OUTPUT XXX;":MACHINE1:STRACE:TERM B,'ABC','#BXXXX1101' "
```

TERM

Query Syntax: :MACHine{1|2}:STRace:TERM? <term_id>,<label_name>

Returned Format: [:MACHine{1|2}:STRAce:TERM] <term_id>,<label_name>,<pattern><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:STRACE:TERM? B,'DATA' "
30 ENTER XXX;String$
40 PRINT String$
50 END
```

SLISt Subsystem 7

Introduction

The SLISt subsystem contains the commands available for the State Listing menu in the HP 16510B logic analyzer module. These commands are:

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

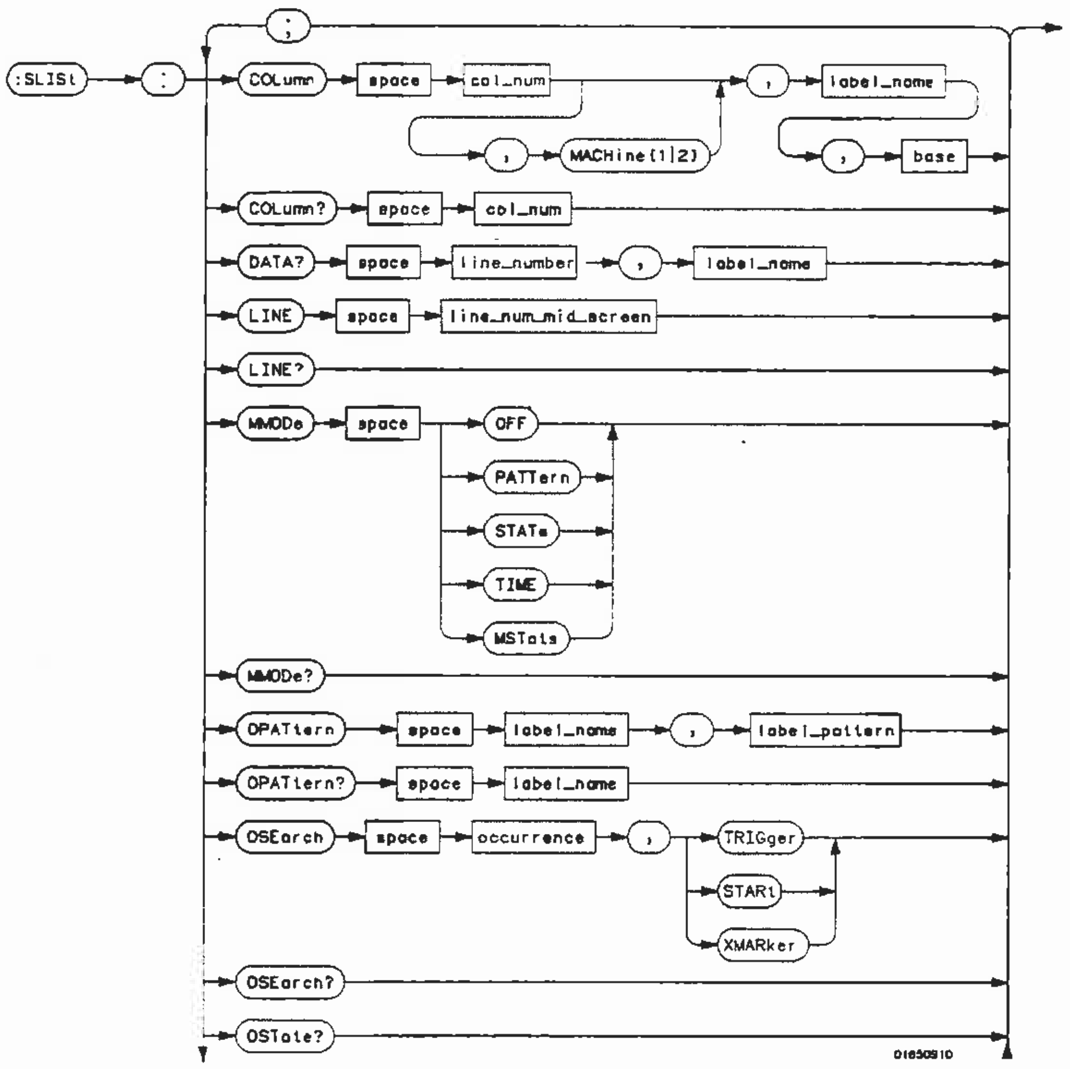


Figure 7-1. SLISt Subsystem Syntax Diagram

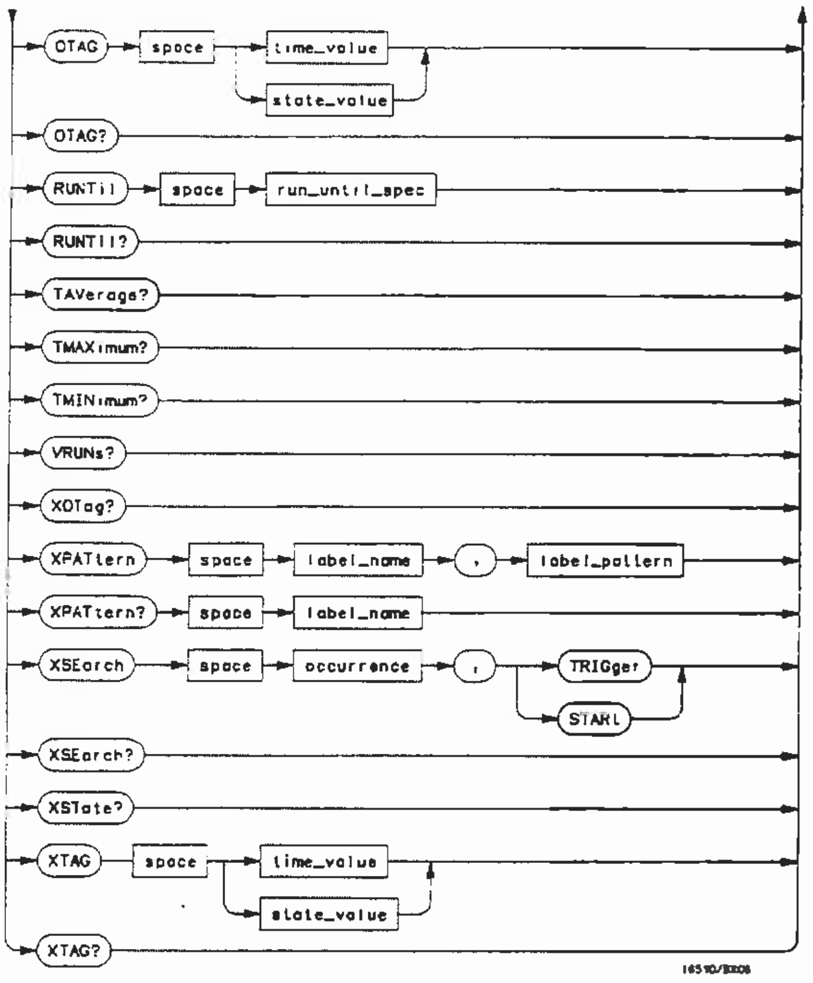


Figure 7-1. SLISt Subsystem Syntax Diagram (continued)

module_num = {*1*|*2*|*3*|*4*|*5*}
mach_num = {*1*|*2*}
col_num = {*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*}
line_number = *integer from -1023 to + 1023*
label_name = *a string of up to 6 alphanumeric characters*
base = {*BINary*|*HEXadecimal*|*OCTal*|*DECimal*|*ASCii*|*SYMBol*|*IASSembler*} *for labels or*
{*ABSolute*|*RELative*} *for tags*
line_num_mid_screen = *integer from -1023 to + 1023*
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 -1023 to + 1023*
time_value = *real number*
state_value = *real number*
run_until_spec = {*OFF*|*LT*, <*value*> |*GT*, <*value*> |*INRange*, <*value*>, <*value*> |
OUTRange, <*value*>, <*value*> }
value = *real number*

Figure 7-1. SLISt Subsystem Syntax Diagram (continued)

SLISt

selector

The SLISt selector is used as part of a compound header to access those settings normally found in the State Listing menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: `:MACHine{1|2}:SLISt`

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:LINE 256"
```

COLumn

command/query

The COLumn command allows you to configure the state analyzer list display by assigning a label name and base to one of the eight vertical columns 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.

The optional parameters allow you to display columns from other time-correlated modules and machines. The optional module number specifies the module number of another time-correlated module. If the module number is not specified, the selected module is assumed. The optional machine number specifies the machine number of another time-correlated machine. If the machine number is not specified, the selected machine is assumed.

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:	:MACHine{1|2}:SLISt:COLumn <col_num>[,<module_num>,MACHine{1|2}],<label_name>,<base>
where:	
<col_num>	::= {1|2|3|4|5|6|7|8}
<module_num>	::= {1|2|3|4|5}
<label_name>	::= a string of up to 6 alphanumeric characters
<base>	::= {BINary|HEXadecimal|OCTal|DECimal|ASCii|SYMBol|IASSembler} for labels or ::= {ABSolute|RELative} for tags

Examples:

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

Query Syntax: :MACHine{1|2}:SLISt:COLumn? <col_num>

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

Example:

```
10 DIM C1$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:COLUMN? 4"
30 ENTER XXX;C1$
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:	:MACHine{1|2}:SLISt:DATA? <line_number>,<label_name>
Returned Format:	[:MACHine{1|2}:SLISt:DATA] <line_number>,<label_name>,<pattern_string><NL>

where:

<line_number>	::= integer from -1023 to +1023
<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$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:DATA? 512, 'RAS'"
30 ENTER XXX;Sd$
40 PRINT Sd$
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: :MACHine{1|2}:SLISt:LINE <line_num_mid_screen>

where:

<line_num_mid_screen> ::= integer from -1023 to +1023

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:LINE 0"
```

Query Syntax: :MACHine{1|2}:SLISt:LINE?

Returned Format: [:MACHine{1|2}:SLISt:LINE] <line_num_mid_screen><NL>

Example:

```
10 DIM Ln$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:LINE?"
30 ENTER XXX;Ln$
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 will be 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.

The MMODe query returns the current marker mode selected.

Command Syntax: :MACHine{1|2}:SLISt:MMODe <marker_mode>

where:

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

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:MMODE TIME"
```

Query Syntax: :MACHine{1|2}:SLISt:MMODe?

Returned Format: [:MACHine{1|2}:SLISt:MMODe] <marker_mode><NL>

Example:

```
10 DIM Mn$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:MMODE?"
30 ENTER XXX;Mn$
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. 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 returns the pattern specification for a given label name.

Command Syntax:

```
:MACHine{1|2}:SLISt: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;":MACHINE1:SLIST:OPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN 'ABC','#BXXXX1101' "
```

OPATtern

Query Syntax: :MACHine{1|2}:SLISt:OPATtern? <label_name>

Returned Format: [:MACHine{1|2}:SLISt:OPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Op$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OPATTERN? 'A'"
30 ENTER XXX;Op$
40 PRINT Op$
50 END
```

OSEarch

command/query

The OSEarch command defines the search criteria for the O marker, which is then used with associated 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: :MACHine{1|2}:SLISt:OSEarch <occurrence>,<origin>

where:

<occurrence>	::= integer from -1023 to +1023
<origin>	::= {TRIGger|STARt|XMARker}

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:OSEARCH +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:SLISt:OSEarch?

Returned Format: [:MACHine{1|2}:SLISt:OSEarch] <occurrence>,<origin><NL>

Example:

```
10 DIM Os$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OSEARCH?"
30 ENTER XXX;Os$
40 PRINT Os$
50 END
```

OSTate

query

The OSTate query returns the line number in the listing where the O marker resides (-1023 to +1023). If data is not valid , the query returns 32767.

Query Syntax: :MACHine{1|2}:SLISt:OSTate?

Returned Format: [:MACHine{1|2}:SLISt:OSTate] <state_num><NL>

where:

<state_num> ::= an integer from -1023 to +1023, or 32767

Example:

```
10 DIM Os$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OSTATE?"
30 ENTER XXX;Os$
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 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: :MACHine{1|2}:SLISt:OTAG {<time_value>|<state_value>}

where:

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

Example:

```
:OUTPUT XXX;":MACHINE1:SLIST:OTAG 40.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:OTAG?

Returned Format: [:MACHine{1|2}:SLISt:OTAG] {<time_value>|<state_value>}<NL>

Example:

```
10 DIM Ot$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:OTAG?"
30 ENTER XXX;Ot$
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 display's 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 STRace 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).

The RUNTil query returns the current stop criteria.

Note The RUNTil instruction (for state analysis) is available in both the SLISt and COMPare subsystems.

RUNTil

Command Syntax: :MACHine{1|2}:SLISt: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;":MACHINE1:SLIST:RUNTIL GT,800.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:RUNTil?

Returned Format: [:MACHine{1|2}:SLISt:RUNTil] <run_until_spec><NL>

Example:

```
10 DIM Ru$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:RUNTIL?"
30 ENTER XXX;Ru$
40 PRINT Ru$
50 END
```

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: `:MACHine{1|2}:SLISt:TAVerage?`

Returned Format: `[:MACHine{1|2}:SLISt:TAVerage] <time_value><NL>`

where:

`<time_value>` ::= real number

Example:

```
10 DIM Tv$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TAVERAGE?"
30 ENTER XXX;Tv$
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: :MACHine{1|2}:SLISt:TMAXimum?

Returned Format: [:MACHine{1|2}:SLISt:TMAXimum] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tx$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TMAXIMUM?"
30 ENTER XXX;Tx$
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: :MACHine{1|2}:SLISt:TMINimum?

Returned Format: [:MACHine{1|2}:SLISt:TMINImum] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tm$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:TMINIMUM?"
30 ENTER XXX;Tm$
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: :MACHine{1|2}:SLISt:VRUNs?

Returned Format: [:MACHine{1|2}:SLISt:VRUNs] <valid_runs>,<total_runs><NL>

where:

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

Example:

```
10 DIM Vr$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:VRUNS?"
30 ENTER XXX;Vr$
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: :MACHine{1|2}:SLISt:XOTag?

Returned Format: [:MACHine{1|2}:SLISt:XOTag] {<XO_time>|<XO_states>}<NL>

where:

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

Example:

```
10 DIM Xot$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XOTAG?"
30 ENTER XXX;Xot$
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 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 returns the pattern specification for a given label name.

Command Syntax: :MACHine{1|2}:SLISt: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;":MACHINE1:SLIST:XPATTERN 'DATA','255' "
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN 'ABC','#BXXXX1101' "
```

XPATtern

Query Syntax: :MACHine{1|2}:SLISt:XPATtern? <label_name>

Returned Format: [:MACHine{1|2}:SLISt:XPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Xp$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XPATTERN? 'A'"
30 ENTER XXX;Xp$
40 PRINT Xp$
50 END
```

XSEarch

command/query

The XSEarch command defines the search criteria for the X Marker, which is then with associated 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.

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

Command Syntax: :MACHine{1|2}:SLISt:XSEarch <occurrence>,<origin>

where:

<occurrence>	::= integer from -1023 to +1023
<origin>	::= {TRIGger|STARt}

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:XSEARCH +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:SLISt:XSEarch?

Returned Format: [:MACHine{1|2}:SLISt:XSEarch] <occurrence>,<origin><NL>

Example:

```
10 DIM Xs$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XSEARCH?"
30 ENTER XXX;Xs$
40 PRINT Xs$
50 END
```

XSTate

query

The XSTate query returns the line number in the listing where the X marker resides (-1023 to +1023). If data is not valid, the query returns 32767.

Query Syntax:	:MACHine{1|2}:SLISt:XSTate?
Returned Format:	[:MACHine{1|2}:SLISt:XSTate] <state_num><NL>
where:	
<state_num>	::= an integer from -1023 to +1023, or 32767

Example:

```
10 DIM Xs$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XSTATE?"
30 ENTER XXX;Xs$
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 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 tagged data, the query returns 9.9E37 for time tagging, 32767 for state tagging.

Command Syntax: :MACHine{1|2}:SLISt:XTAG {<time_value>|<state_value>}

where:

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

Example:

```
:OUTPUT XXX;":MACHINE1:SLIST:XTAG 40.0E-6"
```

Query Syntax: :MACHine{1|2}:SLISt:XTAG?

Returned Format: [:MACHine{1|2}:SLISt:XTAG] {<time_value>|<state_value>}<NL>

Example:

```
10 DIM Xt$[100]
20 OUTPUT XXX;":MACHINE1:SLIST:XTAG?"
30 ENTER XXX;Xt$
40 PRINT Xt$
50 END
```

SWAVeform Subsystem 8

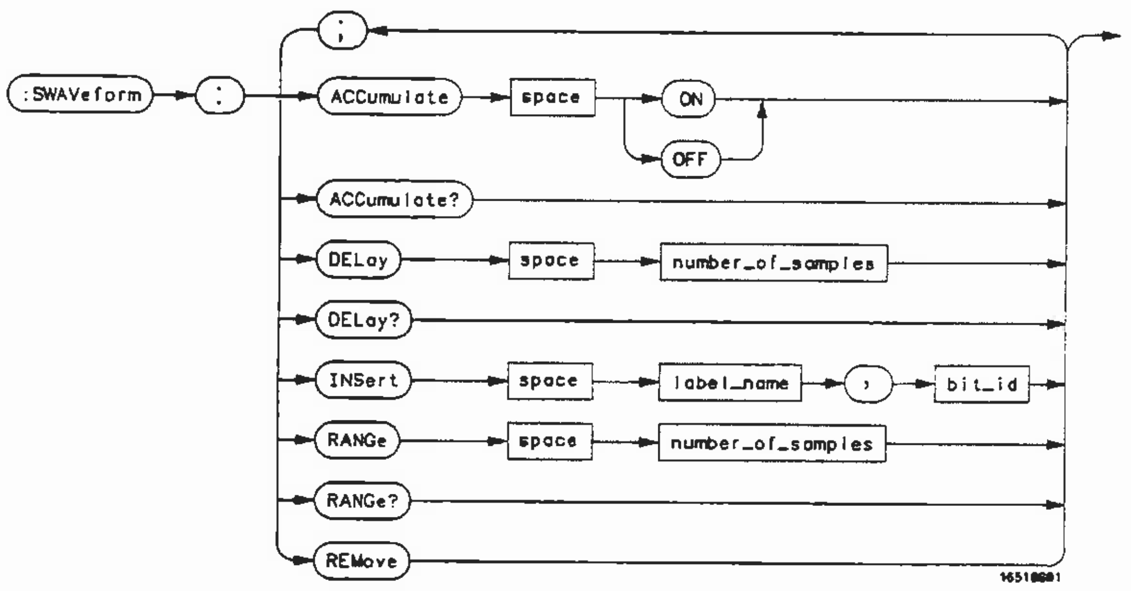
Introduction

The commands in the State Waveform subsystem allow you to configure the display so that you can view state data as waveforms on up to 24 channels 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 whether time tagging is on or off. 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 (SLISt) subsystem. Using the marker commands from the SLISt subsystem will affect the markers on the Waveform display.

The commands in the SWAVeform subsystem are:

- ACCumulate
- DELay
- INSert
- RANGe
- REMove



number_of_samples = *integer from -1023 to + 1024*
label_name = *string of up to 6 alphanumeric characters*
bit_id = {*OVERlay* | <*bit_num*>}
bit_num = *integer representing a label bit from 0 to 31*

Figure 8-1. SWAVeform Subsystem Syntax Diagram

SWAVeform

selector

The SWAVeform (State Waveform) selector is used as part of a compound header to access the settings in the State Waveform menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SWAVeform

Example:

```
OUTPUT XXX;":MACHINE2: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: :MACHine{1|2}:SWAVeform:ACCumulate {{ON | 1} | {OFF | 0}}

Example:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE ON"
```

Query Syntax: MACHine{1|2}:SWAVeform:ACCumulate?

Returned Format: [MACHine{1|2}:SWAVeform:ACCumulate] {0 | 1}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE?"
30 ENTER XXX; String$
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 -1023 to +1024.

The DELay query returns the current sample offset value.

Command Syntax: :MACHine{1|2}:SWAVeform:DELay <number_of_samples>

where:

<number_of_samples>	::= integer from −1023 to +1024

Example:

```
OUTPUT XXX;":MACHINE2:SWAVEFORM:DELAY 127"
```

Query Syntax: MACHine{1|2}:SWAVeform:DELay?

Returned Format: [MACHine{1|2}:SWAVeform:DELay] <number_of_samples><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SWAVEFORM:DELAY?"
30 ENTER XXX;String$
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 24 waveforms are present, inserting additional waveforms 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.

Command Syntax: MACHine{1|2}:SWAVeform:INSert <label_name>,<bit_id>

where:

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

Examples:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:INSERT 'WAVE', 19"
OUTPUT XXX;":MACHINE1:SWAVEFORM:INSERT 'ABC', OVERLAY"
OUTPUT XXX;":MACH1: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 1040 may be entered.

The RANGe query returns the current range value.

Command Syntax:	MACHine{1|2}:SWAVeform:RANGe <number_of_samples>
where:	
<number_of_samples>	::= integer from 10 to 1040
Example:	`OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE 8"`
Query Syntax:	MACHine{1|2}:SWAVeform:RANGe?
Returned Format:	[MACHine{1|2}:SWAVeform:RANGe] <number_of_samples><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

REMove

command

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

Command Syntax: `:MACHine{1|2}:SWAVeform:REMove`

Example:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:REMOVE"
```

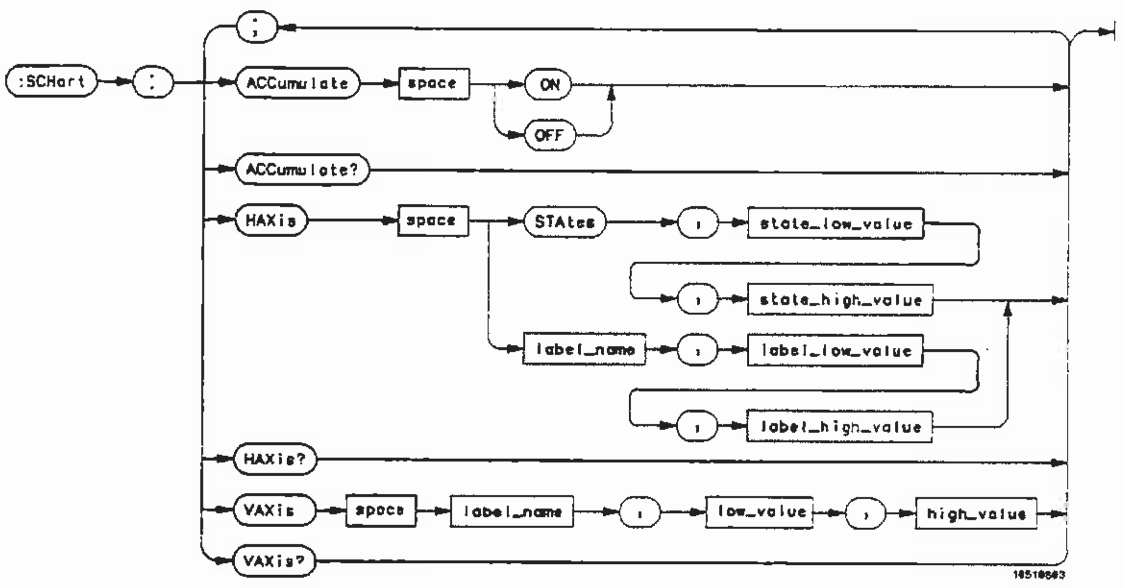
SChart Subsystem 9

Introduction

The State Chart subsystem provides the commands necessary for programming the HP 16510B's Chart display. The commands allow you 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 State 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 State Chart display is simply an alternative way of looking at the data in the State Listing, the X and O markers can be manipulated through the SLISt subsystem. In fact, because the programming commands do not force the menus to switch, you can position the markers in the SLISt subsystem and see the effects in the State Chart display.

The commands in the SCHart subsystem are:

- ACCumulate
- HAXis
- VAXis



state_low_value = *integer from −1023 to +1024*
state_high_value = *integer from <state_low_value> to +1024*
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* low_value *to* 2^{32} *- 1 (#HFFFF)*

Figure 9-1. SCHart Subsystem Syntax Diagram

SCHart

selector

The SCHart selector is used as part of a compound header to access the settings found in the State Chart menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SCHart

Example:

```
OUTPUT XXX;":MACHINE1:SCHART: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: MACHine{1|2}:SCHart:ACCumulate {{ON | 1} | {OFF | 0}}

Example:

```
OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE OFF"
```

Query Syntax: MACHine{1|2}:SCHart:ACCumulate?

Returned Format: [MACHine{1|2}:SCHart:ACCumulate] {0 | 1}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE?"
30 ENTER XXX; String$
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: MACHine{1|2}:SCHart:HAXis {STAtes,<state_low_value>,<state_high_value> | <label_name>,<label_low_value>,<label_high_value>}

where:

<state_low_value>	::= integer from -1023 to 1024
<state_high_value>	::= integer from <state_low_value> to +1024
<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;":MACHINE1:SCHART:HAXIS STATES, -100, 100"
OUTPUT XXX;":MACHINE1:SCHART:HAXIS 'NAUJ', '-511', '511'"
```

Query Syntax: MACHine{1|2}:SCHart:HAXis?

Returned Format: [MACHine{1|2}:SCHart:HAXis] {STAtes,<state_low_value>,<state_high_value> | <label_name>,<label_low_value>,<label_high_value>}

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SCHART:HAXIS?"
30 ENTER XXX; String$
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 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: MACHine{1|2}:SCHart: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;":MACHINE2:SCHART:VAXIS 'SUM1', '0', '99'"
OUTPUT XXX;":MACHINE1:SCHART:VAXIS 'BUS', '#H00FF', '#H0500'"
```

Query Syntax: MACHine{1|2}:SCHart:VAXis?

Returned Format: [MACHine{1|2}:SCHart:VAXis] <label_name>,<low_value>,<high_value><NL>

Example:

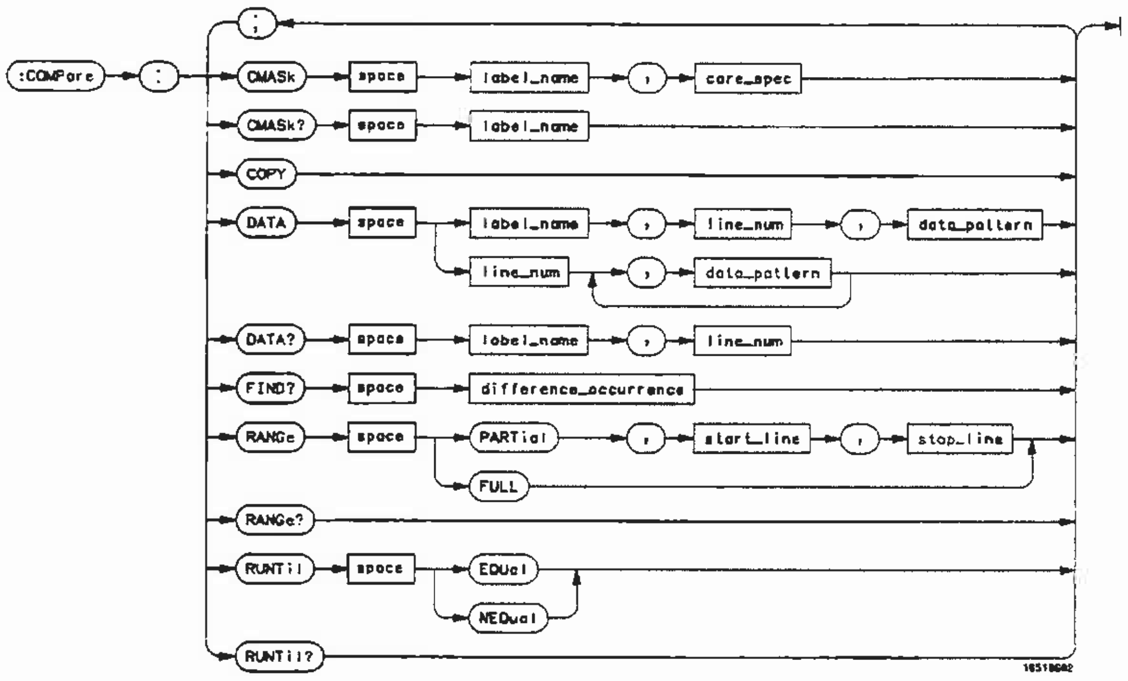
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE1:SCHART:VAXIS?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

COMPare Subsystem 10

Introduction

Commands in the state COMPare subsystem provide the ability to do a bit-by-bit comparison between the acquired state data listing and a compare data image. The commands are:

- COPY
- DATA
- CMASk
- RANGe
- RUNTil
- FIND



label_name = *string of up to 6 characters*
care_spec = *string of characters* "{*|.}..."
* = *care*
. = *don't care*
line_num = *integer from −1023 to +1023*
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 1024*
start_line = *integer from −1023 to +1023*
stop_line = *integer from <start_line> to +1023*

Figure 10-1. COMPare Subsystem Syntax Diagram

COMPare

selector

The COMPare selector is used as part of a compound header to access the settings found in the Compare menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:COMPare

Example:

```
OUTPUT XXX;":MACHINE1: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: MACHine{1|2}: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;":MACHINE2:COMPARE:CMASK 'EKIM', '*.**..**'"
```

Query Syntax: MACHine{1|2}:COMPare:CMASk? <label_name>

Returned Format: [MACHine{1|2}:COMPare:CMASk] <label_name>,<care_spec><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:COMPARE:CMASK? 'POD5'"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

COPY

command

The COPY command copies the current acquired State Listing for the specified machine into the Compare Listing template. It does not affect the compare range or channel mask settings.

Command Syntax: `MACHine{1|2}:COMPare:COPY`

Example:

```
OUTPUT XXX;":MACHINE2: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.

Not specifying the <label_name> parameter allows you to write data patterns to more than one label for the given line number. The first pattern is placed in the left-most label, with the following patterns being placed in a left-to-right fashion (as seen on the Compare display). Specifying more patterns than there are labels simply results in the extra patterns being ignored.

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:

```
MACHine{1|2}:COMPare:DATA  {<label_name>,<line_num>,<data_pattern> |
<line_num>,<data_pattern>[, <data_pattern>]... }
```

where:

```
<label_name>    ::= a string of up 6 alphanumeric characters
<line_num>      ::= integer from -1023 to +1023
<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;":MACHINE2:COMPARE:DATA 'CLOCK', 42, '#B011X101X'"
OUTPUT XXX;":MACHINE2:COMPARE:DATA 'OUT3', 0, '#HFF40'"
OUTPUT XXX;":MACHINE1:COMPARE:DATA 129, '#BXX00', '#B1101', '#B10XX'"
OUTPUT XXX;":MACH2:COMPARE:DATA -511, '4', '64', '16', 256', '8', '16'"
```

Query Syntax: MACHine{1|2}:COMPare:DATA? <label_name>,<line_num>

Returned Format: [MACHine{1|2}: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;":MACHINE2: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;":MACH2: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 (first, second, third, etc) within the current compare range, as dictated by the RANGe command (see next page). 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: MACHine{1|2}:COMPare:FIND? <difference_occurrence>

Returned Format: [MACHine{1|2}:COMPare:FIND] <difference_occurrence>, <line_number><NL>

where:

<difference_occurrence>	::= integer from 1 to 1024
<line_number>	::= integer from −1023 to +1023

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:COMPARE:FIND? 26"
30 ENTER XXX; String$
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: MACHine{1|2}:COMPare:RANGe {FULL | PARTial,<start_line>,<stop_line>}

where:

<start_line>	::= integer from −1023 to +1023
<stop_line>	::= integer from <start_line> to +1023

Examples:

```
OUTPUT XXX;":MACHINE2:COMPARE:RANGE PARTIAL,  -511, 512"
OUTPUT XXX;":MACHINE2:COMPARE:RANGE FULL"
```

Query Syntax: MACHine{1|2}:COMPare:RANGe?

Returned Format: [MACHine{1|2}:COMPare:RANGe] {FULL | PARTial,<start_line>, <stop_line>}<NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE4:COMPARE:RANGE?"
30 ENTER XXX; String$
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 display's 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 STRace 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 (for state analysis) is available in both the SLISt and COMPare subsystems.

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

RUNTil

Command Syntax: MACHine{1|2}: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;":MACHINE2:COMPARE:RUNTIL EQUAL"
```

Query Syntax: MACHine{1|2}:COMPare:RUNTil?

Returned Format: [MACHine{1|2}:COMPare:RUNTil] {OFF| LT,<value>|GT,<value>| INRange,<value>,<value>|OUTRange,<value>,<value>|EQUal|NEQual}<NL>

Example:

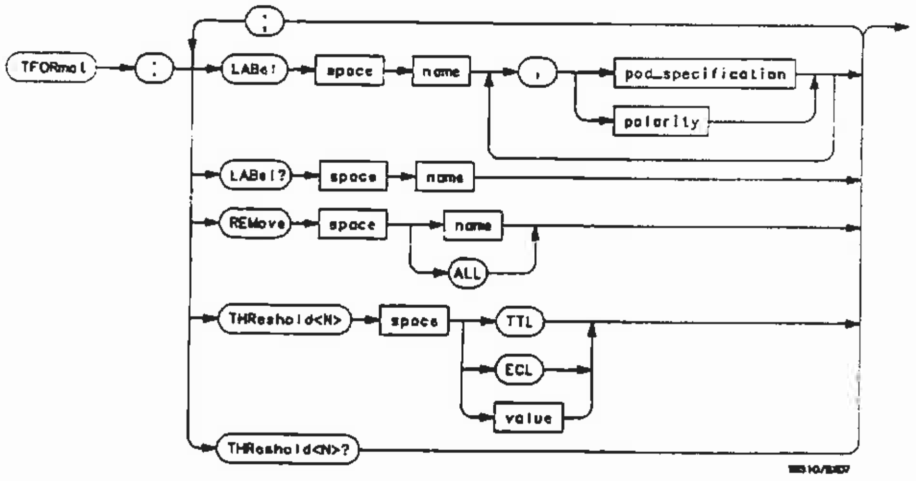
```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:COMPARE:RUNTIL?"
30 ENTER XXX; String$
40 PRINT String$
50 END
```

TFORmat Subsystem 11

Introduction

The TFORmat subsystem contains the commands available for the Timing Format menu in the HP 16510B logic analyzer module. These commands are:

- LABel
- REMove
- THReshold



<N> = *{1 | 2 | 3 | 4 | 5}*
name = *string of up to 6 alphanumeric characters*
polarity = *{POSitive | NEGative}*
pod_specification = *format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)*
value = *voltage (real number) -9.9 to +9.9*

Figure 11-1. TFORmat Subsystem Syntax Diagram

TFORmat

selector

The TFORmat selector is used as part of a compound header to access those settings normally found in the Timing Format menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Command Syntax: :MACHine{1|2}:TFORmat

Example:

```
OUTPUT XXX;":MACHINE1:TFORMAT:LABEL?"
```

LABel

command/query

The LABel command allows you to specify polarity and 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.

The order of the pod-specification parameters is significant. The first one listed will match the highest-numbered pod assigned to the machine you're using. Each pod specification after that is assigned to the next-highest-numbered pod. This way they match the left-to-right descending order of the pods you see on the Format display. Not including enough pod specifications results in the lowest-numbered pod(s) being assigned a value of zero (all channels excluded). If you include more pod specifications than there are pods for that machine, the extra ones will be ignored. However, an error is reported anytime more than five pod specifications are listed.

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

Since 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. Numbers are always returned in decimal format.

LABel

Command Syntax: :MACHine{1|2}:TFORmat:LABel <name>[, {<polarity> | <assignment>}]...

where:

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

Examples:

```
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'DATA', POS, 65535, 127, 40312"
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'STAT', 1, 8096, POSITIVE"
OUTPUT XXX;":MACHINE1:TFORMAT:LABEL 'ADDR', NEGATIVE, #B11110010101010"
```

Query Syntax: :MACHine{1|2}:TFORmat:LABel? <name>

Returned Format: [:MACHine{1|2}:TFORmat:LABel] <name>[,<assignment>]...,<polarity><NL>

Example:

```
10 DIM String$[100]
20 OUTPUT XXX;":MACHINE2:SFORMAT:LABEL? 'DATA'"
30 ENTER XXX String$
40 PRINT String$
50 END
```

REMove

command

The REMove command allows you to delete all labels or any one label specified by name for a given machine.

Command Syntax: :MACHine{1|2}:TFORmat:REMove {<name>|ALL}

where:

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

Examples:

```
OUTPUT XXX;":MACHINE1:TFORMAT:REMOVE 'A'"
OUTPUT XXX;":MACHINE1:TFORMAT:REMOVE ALL"
```

THReshold

command/query

The THReshold command allows you to set the voltage threshold for a given pod to ECL, TTL or a specific voltage from -9.9V to +9.9V in 0.1 volt increments.

Note The pod thresholds of pods 1, 2, and 3 can be set independently. The pod thresholds of pods 4 and 5 are slaved together; therefore when you set the threshold on pod 4 or 5, both thresholds will be changed to the specified value.

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

Command Syntax: :MACHine{1|2}:TFORmat:THReshold<N> {TTL|ECL|<value>}

where:

<N>	::= pod number {1|2|3|4|5}
<value>	::= voltage (real number) -9.9 to +9.9
TTL	::= default value of +1.6V
ECL	::= default value of -1.3V

Example: OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD1 4.0"

Query Syntax: :MACHine{1|2}:TFORmat:THReshold<N>?

Returned Format: [:MACHine{1|2}:TFORmat:THReshold<N>] <value><NL>

Example:

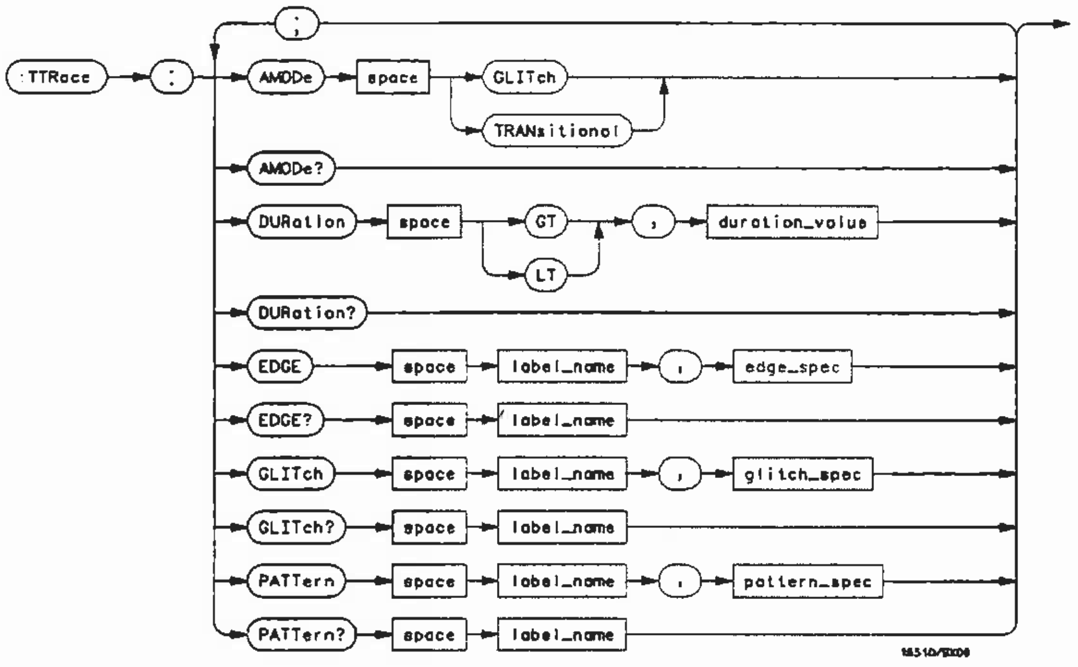
```
10 DIM Value$ [100]
20 OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD2?"
30 ENTER XXX;Value$
40 PRINT Value$
50 END
```

TTRace Subsystem 12

Introduction

The TTRace subsystem contains the commands available for the Timing Trace menu in the HP 16510B logic analyzer module. These commands are:

- AMODe
- DURation
- EDGE
- GLITch
- PATTern



GT = *greater than*
LT = *less than*
duration_value = *real number*
label_name = *string of up to 6 alphanumeric characters*
edge_spec = *string of characters* "{*R*|*F*|*T*|*X*}..."
R = *rising edge*
F = *falling edge*
T = *toggling or either edge*
X = *don't care or ignore this channel*
glitch_spec = *string of characters* "{*|.}..."
* = *search for a glitch on this channel*
. = *ignore this channel*
pattern_spec = "{*#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*} . . . }"

Figure 12-1. TTRace Subsystem Syntax Diagram

TTRace

selector

The TTRace selector is used as part of a compound header to access the settings found in the Timing Trace menu. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the language tree.

Command Syntax: `:MACHine{1|2}:TTRace`

Example:

```
OUTPUT XXX;":MACHINE1:TTRACE:GLITCH 'ABC', '....****'"
```

AMODe

command/query

The AMODe command allows you to select the acquisition mode used for a particular timing trace. The acquisition modes available are TRANsitional and GLITch.

The AMODe query returns the current acquisition mode.

Command Syntax: :MACHine{1|2}:TTRace:AMODe <acquisition_mode>

where:

<acquisition_mode> ::= {GLITch|TRANsitional}

Example:

```
OUTPUT XXX; ":MACHINE1:TTRACE:AMODE GLITCH"
```

Query Syntax: :MACHine1:TTRace:AMODe?

Returned Format: [:MACHine1:TTRace:AMODe] {GLITch|TRANsitional}

Example:

```
10 DIM M$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:AMODE?"
30 ENTER XXX;M$
40 PRINT M$
50 END
```

DURation

command/query

The DURation command allows you to specify the duration qualifier to be used with the pattern recognizer term in generating the timing trigger. The duration value can be specified in 10 ns increments within the following ranges:

- Greater than (GT) qualification - 30 ns to 10 ms
- Less than (LT) qualification - 40 ns to 10 ms.

The DURation query returns the current pattern duration qualifier specification.

Command Syntax: :MACHine{1|2}:TTRace:DURation {GT|LT},<duration_value>

where:

GT	::= greater than
LT	::= less than
<duration_value>	::= real number

Example:

```
OUTPUT XXX; ":MACHINE1:TTRACE:DURATION GT, 40.0E-9"
```

Query Syntax: :MACHine{1|2}:TTRace:DURation?

Returned Format: [:MACHine{1|2}:TTRace:DURation] {GT|LT},<duration_value><NL>

Example:

```
10 DIM D$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:DURATION?"
30 ENTER XXX;D$
40 PRINT D$
50 END
```

EDGE

command/query

The EDGE command allows you to specify the edge recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in the given edge specification; therefore, a complete specification could require several commands. The edge specification uses the characters R, F, T, X to indicate the edges or don't cares as follows:

R = rising edge
F = falling edge
T = toggling or either edge
X = don't care or ignore the channel

The position of these characters in the string corresponds with the position of the channels within the label. All channels without "X" are ORed together to form the edge trigger specification.

The EDGE query returns the edge specification for the specified label.

Command Syntax: `:MACHine{1|2}:TTRace:EDGE <label_name>,<edge_spec>`

where:

<label_name>	::= string or up to 6 alphanumeric characters
<edge_spec>	::= string of characters "{R|F|T|X}..."

Example:

```
OUTPUT XXX; ":MACHINE1:TTRACE:EDGE 'POD1','XXXXXXXR'"
```

EDGE

Query Syntax: :MACHine{1|2}:TTRace:EDGE? <label_name>

Returned Format: [:MACHine{1|2}:TTRace:] <label_name>,<edge_spec><NL>

Example:

```
10 DIM E$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:EDGE? 'POD1'"
30 ENTER XXX;E$
40 PRINT E$
50 END
```

GLITch

command/query

The GLITch command allows you to specify the glitch recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in a given glitch specification, and, therefore a complete specification could require several commands. The glitch specification uses the characters "*" (search this channel) and "." (ignore this channel).

The position of these characters in the string corresponds with the position of the channels within the label. All channels with the "*" are ORed together to form the glitch trigger specification.

The GLITch query returns the glitch specification for the specified label.

Command Syntax: :MACHine{1|2}:TTRace:GLITch <label_name>,<glitch_spec>

where:

<label_name>	::= string of up to 6 alphanumeric characters
<glitch_spec>	::= string of characters "{*|.}..."
"*" (asterisk)	::= search for a glitch on this channel
"." (period)	::= ignore this channel

Example:

```
OUTPUT XXX; ":MACHINE1:TTRACE:GLITCH 'POD1','**.......*'"
```

Query Syntax: :MACHine1:TTRace:GLITch? <label_name>

Returned Format: [:MACHine1:TTRace:GLITch] <label_name>,<glitch_spec><NL>

Example:

```
10 DIM G$[100]
20 OUTPUT XXX; ":MACHINE1:TTRACE:GLITCH? 'POD1'"
30 ENTER XXX;G$
40 PRINT G$
50 END
```

PATTern

command/query

The PATTern command allows you to construct a pattern recognizer term for the timing analyzer trigger on a per label basis. Each command deals with only one label in the given pattern; therefore, a complete timing trace specification could require several commands. Since a label can contain up to 32 bits, the range of the pattern value will be between 0 and (2^{32})-1. The value may be expressed in binary (#B), octal (#Q), hexadecimal (#H) or decimal (default). 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 pattern value can contain don't cares, the pattern specification parameter is handled as a string of characters instead of a number.

The PATTern query returns the pattern specification for the specified label in the base previously defined for the label.

Command Syntax: `:MACHine{1|2}:TTRace:PATTern <label_name>,<pattern_spec>`

where:

```
<label_name>    ::= string of up to 6 alphanumeric characters
<pattern_spec>  ::= "{#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; ":MACHINE1:TTRACE:PATTERN 'DATA', '255'"
```

PATTern

Query Syntax: :MACHine{1|2}:TTRace:PATTern? <label_name>

Returned Format: [:MACHine{1|2}:TTRace:PATTern] <label_name>,<pattern_spec><NL>

Example:

```
10 DIM P$[100]
20 OUTPUT XXX; ":MACHINE2:TTRACE:PATTERN? 'DATA'"
30 ENTER XXX;P$
40 PRINT P$
50 END
```

TWAVeform Subsystem 13

Introduction

The TWAVeform subsystem contains the commands available for the Timing Waveforms menu in the HP 16510B. These commands are:

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

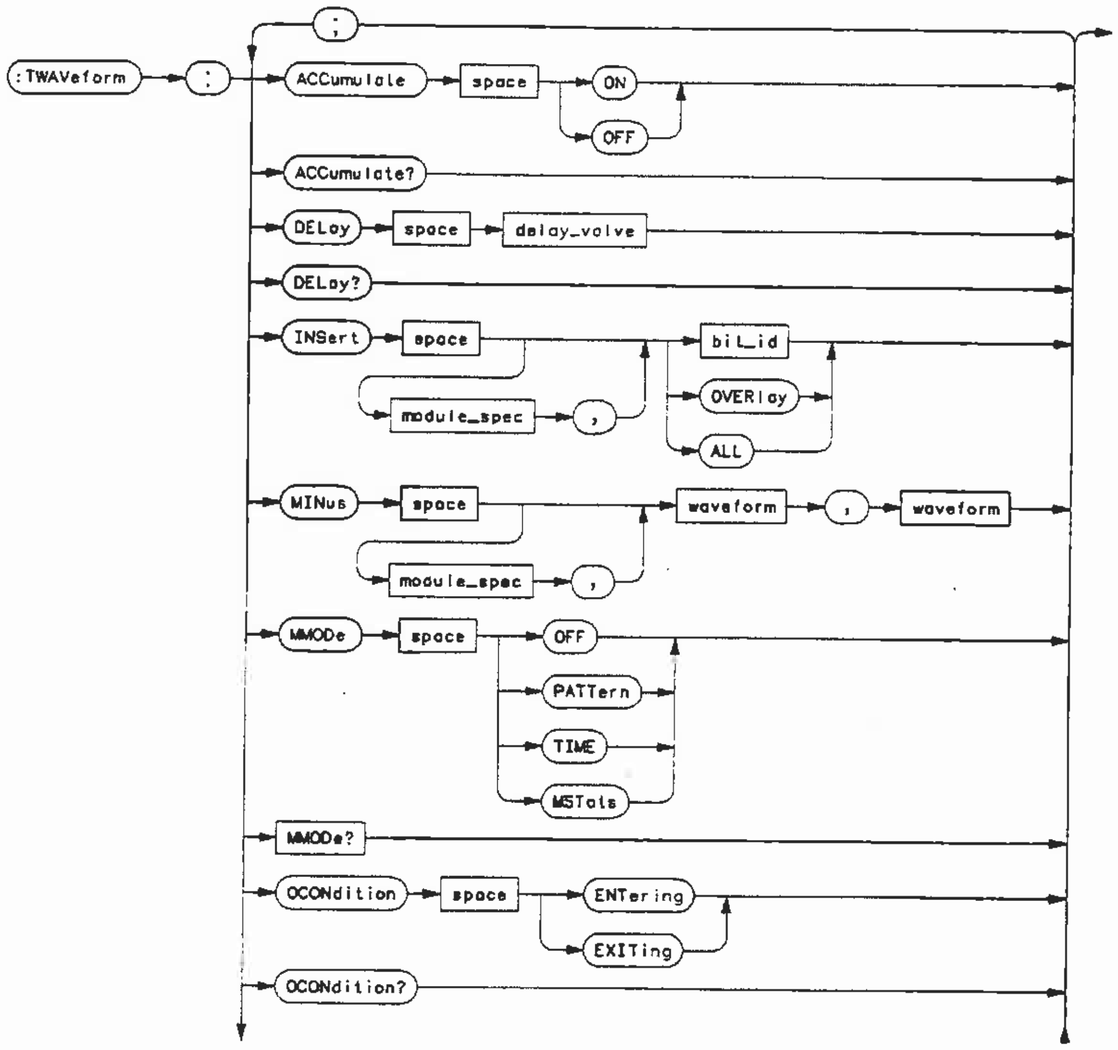


Figure 13-1. TWAVeform Subsystem Syntax Diagram

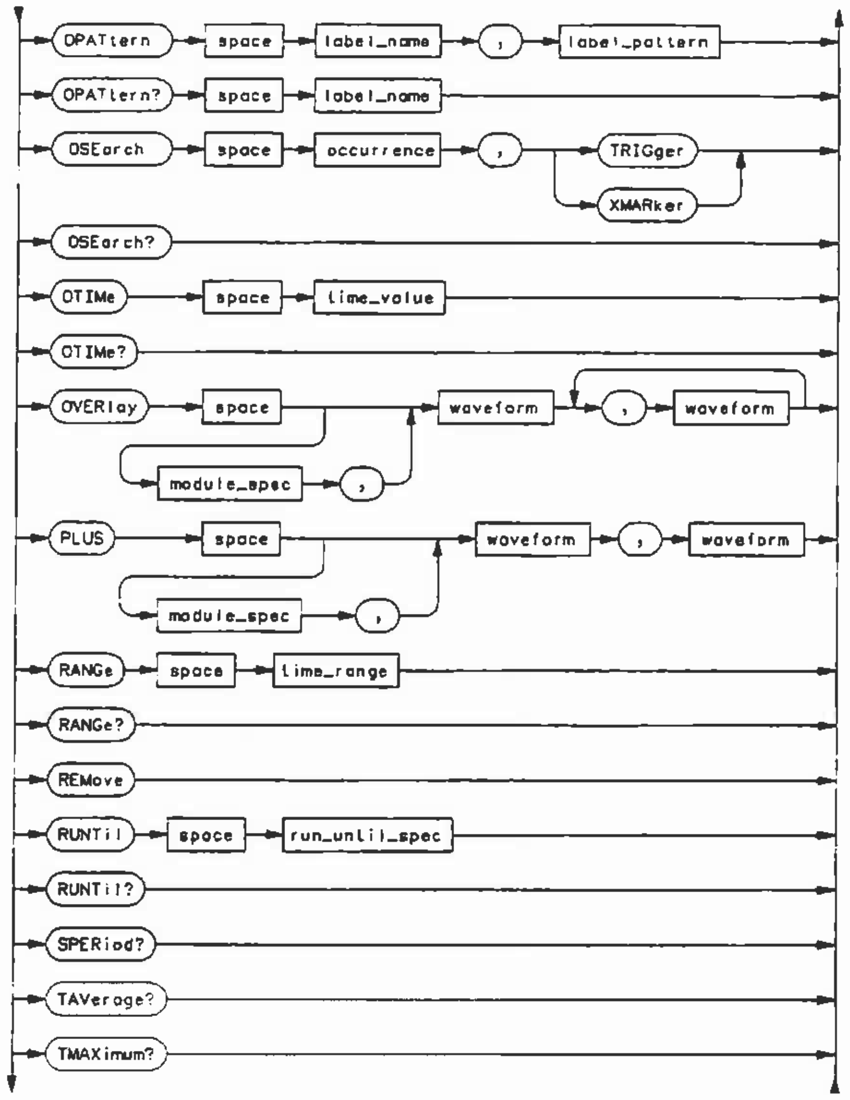


Figure 13-1. TWAVeform Subsystem Syntax Diagram (continued)

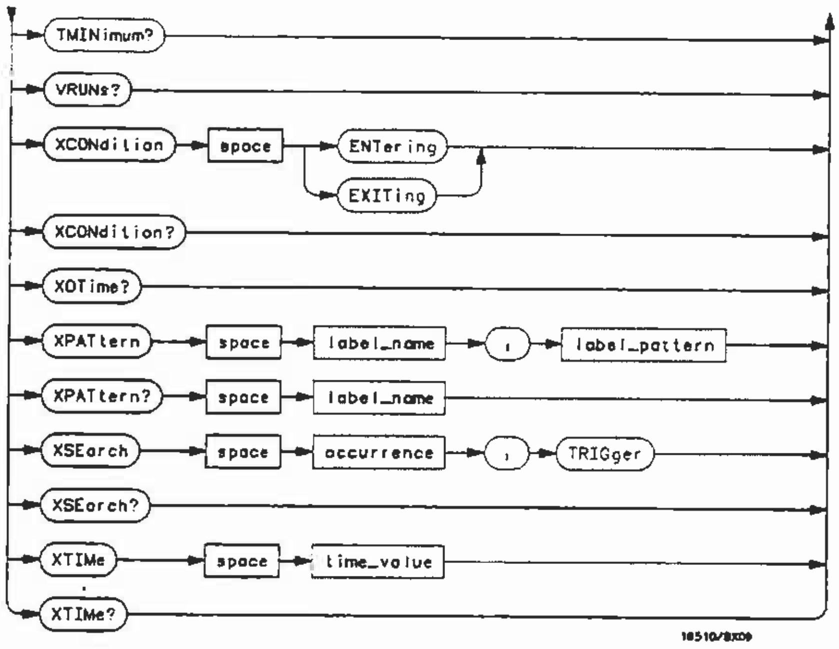


Figure 13-1. TWAVeform Subsystem Syntax Diagram (continued)

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*
label_id = *string of one alpha and one numeric character*
module_num = *slot number in which the timebase card is installed*
time_range = *real number between 100 ns and 10 ks*
run_until_spec =
{*OFF*|*LT*, <*value*> |*GT*, <*value*> |*INRange* <*value*>, <*value*> |*OUTRange* <*value*>, <*value*> }
GT = *greater than*
LT = *less than*
value = *real number*

Figure 13-1. TWAVeform Subsystem Syntax Diagram (continued)

TWAVeform

selector

The TWAVeform selector is used as part of a compound header to access the settings found in the Timing Waveforms menu. It always follows the MACHine selector because it selects a branch below the MACHine level in the command tree.

Command Syntax: `:MACHine{1|2}:TWAVeform`

Example:

```
OUTPUT XXX;":MACHINE1: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: `:MACHine{1|2}:TWAVeform:ACCumulate <setting>`

where:

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

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE ON"
```

Query Syntax: `:MACHine{1|2}:TWAVeform:ACCumulate?`

Returned Format: `[:MACHine{1|2}:TWAVeform:ACCumulate] {0|1}<NL>`

Example:

```
10 DIM P$ [100]
20 OUTPUT XXX;":MACHINE1: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. In glitch acquisition mode, as delay becomes large in an absolute sense, the sample rate is adjusted so that data will be acquired in the time window of interest. In transitional acquisition mode, data may not fall in the time window since the sample period is fixed at 10 ns and the amount of time covered in memory is dependent on how frequent the input signal transitions occur.

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

Command Syntax: :MACHine{1|2}:TWAVeform:DELay <delay_value>

where:

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

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY 100E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:DELay?

Returned Format: [:MACHine{1|2}:TWAVeform:DELay] <time_value><NL>

Example:

```
10 DIM Dl$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY?"
30 ENTER XXX; Dl$
40 PRINT Dl$
50 END
```

INSert

command

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom. When 24 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: :MACHine{1|2}: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;":MACHINE1:TWAVEFORM:INSERT 3, 'WAVE',10"
```

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

Command Syntax: :MACHine{1|2}: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;":MACHINE1: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: :MACHine{1|2}:TWAVeform:MMODe {OFF|PATTern|TIME|MSTats}

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:MMODE TIME"
```

Query Syntax: :MACHine{1|2}:TWAVeform:MMODe?

Returned Format: [:MACHine{1|2}:TWAVeform:MMODe] <marker_mode><NL>

where

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

Example:

```
10 DIM M$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:OCONdition {ENTering|EXITing}

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OCONDITION ENTERING"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OCONdition?

Returned Format: [:MACHine{1|2}:TWAVeform:OCONdition] {ENTering|EXITing}<NL>

Example:

```
10 DIM Oc$ [100]
20 OUTPUT XXX;":MACHINE1: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:

```
:MACHine{1|2}: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; ":MACHINE1:TWAVEFORM:OPATTERN 'A','511'"
```

OPATtern

Query Syntax: :MACHine{1|2}:TWAVeform:OPATtern? <label_name>

Returned Format: [:MACHine{1|2}:TWAVeform:OPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Op$ [100]
20 OUTPUT XXX;":MACHINE1: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 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 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 OSEarch query returns the search criteria for the O marker.

Command Syntax: :MACHine{1|2}:TWAVeform:OSEarch <occurrence>,<origin>

where:

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

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OSEARCH +10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OSEarch?

Returned Format: [:MACHine{1|2}:TWAVeform:OSEarch] <occurrence>,<origin><NL>

Example:

```
10 DIM Os$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:OTIMe <time_value>

where:

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

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:OTIME 30.0E-6"
```

Query Syntax: :MACHine{1|2}:TWAVeform:OTIMe?

Returned Format: [:MACHine{1|2}:TWAVeform:OTIMe] <time_value> <NL>

Example:

```
10 DIM Ot$ [100]
20 OUTPUT XXX;":MACHINE1: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.

Command Syntax: `:MACHine{1|2}: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;":MACHINE1: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 subtracted from 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 10 ks.

The RANGe query returns the current full-screen time.

Command Syntax: :MACHine{1|2}:TWAVeform:RANGe <time_value>

where:

<time_range>	::= real number between 100 ns and 10 ks

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE 100E-9"
```

Query Syntax: :MACHine{1|2}:TWAVeform:RANGe?

Returned Format: [:MACHine{1|2}:TWAVeform:RANGe] <time_value><NL>

Example:

```
10 DIM Rg$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"
30 ENTER XXX; Rg$
40 PRINT Rg$
50 END
```

REMove

command

The REMove command deletes all waveforms from the display.

Command Syntax: :MACHine{1|2}:TWAVeform:REMove

Example:

```
OUTPUT XXX;":MACHINE1: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.

This command affects the timing analyzer only, and has no relation to the RUNTil commands in the SLISt and COMPare subsystems.

The RUNTil query returns the current stop criteria.

Command Syntax: :MACHine{1|2}:TWAVeform:RUNTil <run_until_spec>

where:

<run_until_spec>	::= {OFF | LT,<value> | GT,<value> | INRange<value>,<value> | OUTRange<value>,<value>}
<value>	::= real number

Examples:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL GT, 800.0E-6"
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL INRANGE, 4.5, 5.5"
```

RUNTil

Query Syntax: :MACHine{1|2}:TWAVeform:RUNTil?

Returned Format: [:MACHine{1|2}:TWAVeform:RUNTil] <run_until_spec><NL>

Example:

```
10 DIM Ru$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL?"
30 ENTER XXX; Ru$
40 PRINT Ru$
50 END
```

SPERiod

query

The SPERiod query returns the sample period of the last run.

Query Syntax: :MACHine{1|2}:TWAVeform:SPERiod?

Returned Format: [:MACHine{1|2}:TWAVeform:SPERiod] <time_value><NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Sp$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:TAVerage?

Returned Format: [:MACHine{1|2}:TWAVeform:TAVerage] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Tv$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:TMAXimum?

Returned Format: [:MACHine{1|2}:TWAVeform:TMAXimum] <time_value><NL>

where

<time_value> ::= real number

Example:

```
10 DIM Tx$ [100]
20 OUTPUT XXX;":MACHINE1: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:	:MACHine{1|2}:TWAVeform:TMINimum?
Returned Format:	[:MACHine{1|2}:TWAVeform:TMINimum] <time_value><NL>
where:	
<time_value>	::= real number

Example:

```
10 DIM Tm$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:VRUNs?

Returned Format: [:MACHine{1|2}: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;":MACHINE1: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: :MACHine{1|2}:TWAVeform:XCONdition {ENTering|EXITing}

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XCONDITION ENTERING"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XCONdition?

Returned Format: [:MACHine{1|2}:TWAVeform:XCONdition] {ENTering|EXITing}<NL>

Example:

```
10 DIM Xc$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:XOTime?

Returned Format: [:MACHine{1|2}:TWAVeform:XOTime] <time_value> <NL>

where:

<time_value> ::= real number

Example:

```
10 DIM Xot$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}: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; ":MACHINE1:TWAVEFORM:XPATTERN 'A','511'"
```

XPATtern

Query Syntax: :MACHine{1|2}:TWAVeform:XPATtern? <label_name>

Returned Format: [:MACHine{1|2}:TWAVeform:XPATtern] <label_name>,<label_pattern><NL>

Example:

```
10 DIM Xp$ [100]
20 OUTPUT XXX;":MACHINE1: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. The occurrence parameter determines which occurrence of the XPATtern recognizer specification, relative to the origin, the marker actually searches for. An occurrence of 0 (zero) places a marker on the origin.

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

Command Syntax: :MACHine{1|2}:TWAVeform:XSEarch <occurrence>,<origin>

where:

<origin>	::= TRIGger
<occurrence>	::= integer from -9999 to +9999

Example:

```
OUTPUT XXX; ":MACHINE1:TWAVEFORM:XSEARCH,+10,TRIGGER"
```

Query Syntax: :MACHine{1|2}:TWAVeform:XSEarch? <occurrence>,<origin>

Returned Format: [:MACHine{1|2}:TWAVeform:XSEarch] <occurrence>,<origin><NL>

Example:

```
10 DIM Xs$ [100]
20 OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}:TWAVeform:XTIMe <time_value>

where:

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

Example: OUTPUT XXX; ":MACHINE1:TWAVEFORM:XTIME 40.0E-6"

Query Syntax: :MACHine{1|2}:TWAVeform:XTIMe?

Returned Format: [:MACHine{1|2}:TWAVeform:XTIMe] <time_value> <NL>

Example:

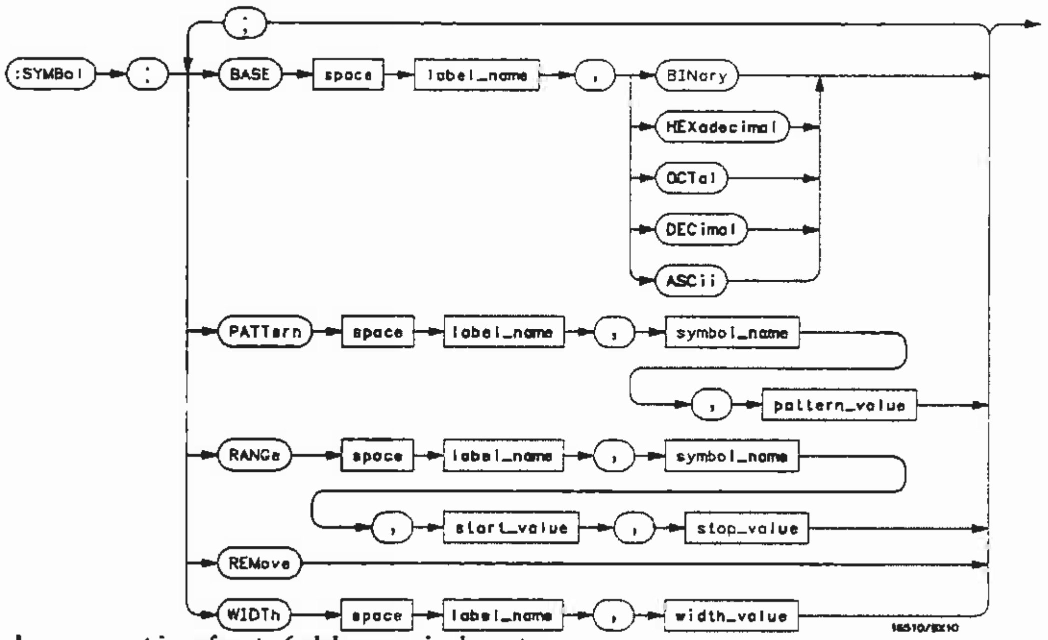
```
10 DIM Xt$ [100]
20 OUTPUT XXX;":MACHINE1:TWAVEFORM:XTIME?"
30 ENTER XXX; Xt$
40 PRINT Xt$
50 END
```

SYMBol Subsystem 14

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 16510B logic analyzer module. The commands in this subsystem are:

- BASE
- PATTern
- RANGe
- REMove
- WIDTh



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*

Figure 14-1. SYMBol Subsystem Syntax Diagram

SYMBol

selector

The SYMBol selector is used as a part of a compound header to access the commands used to create symbols. It always follows the MACHine selector because it selects a branch directly below the MACHine level in the command tree.

Command Syntax: :MACHine{1|2}:SYMBol

Example:

```
OUTPUT XXX;":MACHINE1: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: `:MACHine{1|2}: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;":MACHINE1: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) are allowed in the pattern value, 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.

Command Syntax:

```
:MACHine{1|2}: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;":MACHINE1: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:

```
:MACHine{1|2}: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;":MACHINE1:SYMBOL:RANGE 'STAT', 'IO_ACC','0','#H000F'"
```

REMove

command

The REMove command deletes all symbols from a specified machine.

Command Syntax: :MACHine{1|2}:SYMBol:REMove

Example:

```
OUTPUT XXX;":MACHINE1: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: :MACHine{1|2}: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;":MACHINE1: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 16510B and a controller. Use the DATA instruction to transfer acquired waveform data, and the SETup instruction to transfer instrument configuration data. This is useful for:

- Re-loading to the logic analyzer
- Processing data later
- Processing data in the controller.

This appendix explains how to use these commands.

The format and length of block data depends on the instruction being used and the configuration of the instrument. The SYSTem:DATA section describes each part of the block 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 processing of data in the controller.

Note Do not change the block data in the controller if you intend to send the block data back into the logic analyzer for later processing. Changes made to the block data in the controller could have unpredictable results when sent back to the logic analyer.

SYSTem:DATA

command/query

The SYSTem:DATA command transmits the acquisition memory data from the controller to the HP 16510B logic analyzer.

The SYSTem:DATA query returns the block data.

Note The data sent by the SYSTem:DATA query reflect the configuration of the machines 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 14506 bytes containing information captured by the acquisition chips. The information will be in one of four formats, depending on the type of data captured. Each format is described in the "Acquisition Data Description" section. Since no parameter checking is performed, out-of-range values could cause instrument lockup; therefore, care should be taken when transferring the data string into the HP 16510B.

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 always has a length of 14506 bytes.

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 type of data

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
```

SYSTem:DATA

Section Header Description

The section header uses bytes 1 through 16 (this manual begins counting at 1; there is no byte 0). The 16 bytes of the section header are as follows:

1	10 bytes - section name ("DATA " for the DATA instruction)
11	1 byte - reserved
12	1 byte - module ID (31 for the HP 16510B)
13	4 bytes - length (14506 for the DATA instruction)

Section Data

For the SYSTem:DATA command, the < section data > parameter consists of two parts: the data preamble and the acquisition data. These are described in the following two sections.

Data Preamble Description

The block data is organized as 160 bytes of preamble information, followed by 1024 14-byte groups of information, followed by 10 reserved bytes. The preamble gives information for each analyzer describing the amount and type of data captured, where the trace point occurred in the data, which pods are assigned to which analyzer, and other information.

Each 14-byte group is made up of two bytes (16 bits) of status for Analyzer 1, two bytes of status for Analyzer 2, then five sets of two bytes of information for each of the five 16-bit pods of the HP 16510B.

Note

One analyzer's information is independent of the other analyzer's information. In other words, on any given line, one analyzer may contain data information for a timing machine, while the other analyzer may contain count information for a state machine with time tags enabled. The status bytes for each analyzer describe what the information for that line contains. Therefore, when describing the different formats that data may contain below, keep in mind that this format pertains only to those pods that are assigned to the analyzer of the specified type. The other analyzer's data is TOTALLY independent and conforms to its own format.

The preamble (bytes 17 through 176) consists of the following 160 bytes:

17 2 bytes - Instrument ID (always 16500 for HP 16510B)

19 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 (bytes 21 and 99) may be STATE with tagging, while the current setup of the analyzer is TIMING.

The next 78 bytes are for Analyzer 1 Data Information.

21 1 byte - Machine data mode, one of the following values:
0 = off
1 = state data (with either time or state tags)
2 = state data (without tags)
3 = glitch timing data
4 = transitional timing data

22 1 byte - List of pods in this analyzer, where a 1 indicates that the corresponding pod is assigned to this analyzer.

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
unused	unused	Pod 1	Pod 2	Pod 3	Pod 4	Pod 5	unused

23 1 byte - Master chip in this analyzer - When several chips are grouped together in a single analyzer, one chip is designated as a master chip. This byte identitfies the master chip. A value of 4 represents POD 1, 3 for POD 2, 2 for POD 3, 1 for POD 4, and 0 for POD 5.

24 1 byte - Reserved

25 10 bytes - Number of rows of valid data for this analyzer - Indicates the number of rows of valid data for each of the five pods. Two bytes are used to store each pod value, with the first 2 bytes used to hold POD 5 value, the next 2 for POD 4 value, and so on.

35 1 byte - Trace point seen in this analyzer - Was a trace point seen (value = 1) or forced (value = 0)

36 1 byte - Reserved

37 10 bytes - Trace point location for this analyzer - Indicates the row number in which the trace point was found for each of the five pods. Two bytes are used to store each pod value, with the first 2 bytes used to hold POD 5 value, the next 2 for POD 4 value, and so on.

47 4 bytes - Time from arm to trigger for this analyzer - The number of 40 ns ticks that have taken place from the arm of this machine to the trigger of this machine. A value of -1 (all 32 bits set to 1) indicates counter overflow.

51 1 byte - Armer of this analyzer - Indicates what armed this analyzer (1 = RUN, 2 = BNC, 3 = other analyzer)

52 1 byte - Devices armed by this analyzer - Bitmap of devices armed by this machine

bit 8	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1
unused	unused	unused	unused	unused	BNC out	Mach. 1	Mach. 2

A 1 in a given bit position implies that this analyzer arms that device, while a 0 means the device is not armed by this analyzer.

53 4 bytes - Sample period for this analyzer (timing only) - Sample period at which data was acquired. Value represents the number of nanoseconds between samples.

57 4 bytes - Delay for this analyzer (timing only) - Delay at which data was acquired. Value represents the amount of delay in nanoseconds.

61 1 byte - Time tags on (state with tagging only) - In state tagging mode, was the data captured with time tags (value = 1) or state tags (value = 0).

62 1 byte - Reserved

63 **5 bytes** - Demultiplexing (state only) - For each of the five pods (first byte is POD 5, fifth byte is POD 1) in a state machine, describes multiplexing of each of the five pods. (0 = NO DEMUX, 1 = TRUE DEMUX, 2 = MIXED CLOCKS).

68 **1 byte** - Reserved

69 **20 bytes** - Trace point adjustment for pods - Each pod uses 4 bytes to show the number of nanoseconds that are to be subtracted from the trace point described above to get the actual trace point value. The first 4 bytes are for Pod 5, the next four are for Pod 4, and so on.

89 **10 bytes** - Reserved

The next 78 bytes are for Analyzer 2 Data Information. They are organized in the same manner as Analyzer 1 above, but they occupy bytes 99 through 176

Acquisition Data Description

The acquisition data section consists of 14336 bytes (1024 14-byte groups), appearing in bytes 177 through 14512. The final ten bytes, from 14513 to 14522, are reserved. The data contained in the data section will appear in one of four forms depending on the mode in which it was acquired (as indicated in byte 21 for machine 1 and byte 99 for machine 2). The four modes are:

- State Data (without tags)
- State Data (with either time or state tags)
- Glitch Timing Data
- Transitional Timing Data

The following four sections describe the four data modes that may be encountered. Each section describes the Status bytes (shown under the Machine 1 and Machine 2 headings), and the Information bytes (shown under the Pod 5 through Pod 1 headings).

State Data (without tags)

Status Bytes. In normal state mode, only the least significant bit (bit 1) is used. When bit 1 is set, this means that there has been a sequence level transition.

Information Bytes. In state acquisition with no tags, data is obtained from the target system with each clock and checked with the trace specification. If the state matches this specification, the data is stored, and is placed into the memory.

	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	Data	Data	Data	Data	Data
205	Status	Status	Data	Data	Data	Data	Data
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14499	Status	Status	Data	Data	Data	Data	Data

*The headings are not a part of the returned data.

State Data (with either time or state tags)

Status Bytes. In state tagging mode, the tags indicate whether a given row of the data is a data line, a count (tag) line, or a prestore line.

Bit 2 is the Data vs. Count bit. Bit 3 is the Prestore vs. Tag bit. The two bits together show what the corresponding Information bytes represent.

Bit 3	Bit 2	Information byte represents:
0	0	Acquisition Data
0	1	Count
1	0	Prestore Data
1	1	Invalid

If Bit 2 is clear, the information contains either actual acquisition data as obtained from the target system (if Bit 3 is clear), or prestore data (if Bit 3 is set). If Bit 2 is set and Bit 3 is clear, this row's bytes for the pods assigned to this machine contain tags. If Bit 2 and Bit 3 are set, the corresponding Information bytes are invalid and should be ignored. Bit 1 is used only when Bit 2 is clear. Whenever there has been a sequence level transition Bit 1 will be set, and otherwise will be clear.

Information Bytes. In the State acquisition mode with tags, data is obtained from the target system with each clock and checked with the trace specification. If the state does not match the trace specification, it is checked against the prestore qualifier. If it matches the prestore qualifier, then it is placed in the prestore buffer. If the state does not match either the sequencer qualifier or the prestore qualifier, it is discarded.

The type of information in the bytes labeled Data depends on the Prestore vs. Tags bit. When the Data bytes are used for prestore information, the following Count bytes (in the same column) should be ignored. When the Data bytes are used for tags, the Count bytes are formatted as floating-point numbers in the following fashion:

bits 16 through 12	bits 11 through 1
EEEEE	MMMMMMMMMMM

The five most-significant bits (EEEEE) store the exponent, and the eleven least-significant bits (MMMMMMMMMMM) store the mantissa. The actual value for Count is given by the equation:

$$\text{Count} = (2048 + \text{mantissa}) \times 2^{\text{exponent}} - 2048$$

Since the counts are relative counts from one state to the one previous, the count for the first state in the data structure is invalid.

If time tagging is on, the count value represents the number of 40 nanosecond ticks that have elapsed between the two stored states. In the case of state tagging, the count represents the number of qualified states that were encountered between the stored states.

If a state matches the sequencer qualifiers, the prestore buffer is checked. If there are any states in the prestore buffer at this time, these prestore states are first placed in memory, along with a dummy count row. After this check, the qualified state is placed in memory, followed by the count row which specified how many states (or 40 ns ticks) have elapsed since the last stored state. If this is the first stored state in memory, then the count information that is stored should be discarded.

	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	⊗	⊗	⊗	⊗	⊗
205	Status	Status	Data	Data	Data	Data	Data
219	Status	Status	Count	Count	Count	Count	Count
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14425	Status	Status	Data	Data	Data	Data	Data
14439	Status	Status	Count	Count	Count	Count	Count

*The headings are not a part of the returned data.
⊗ = Invalid data

Glitch Timing Data

Status Bytes. In glitch timing mode, the status bytes indicate whether a given row in the data contains actual acquisition data information or glitch information.

Bit 1 is the Data vs. Glitch bit. If Bit 1 is set, this row of information contains glitch information. If Bit 1 is clear, then this row contains actual acquisition data as obtained from the target system.

Information Bytes. In the Glitch timing mode, the target system is sampled at every sample period. The data is then stored in memory and the glitch detectors are checked. If a glitch has been detected between the previous sample and the current sample, the corresponding glitch bits are set. The glitch information is then stored. If this is the first stored sample in memory, then the glitch information stored should be discarded.

	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	⊗	⊗	⊗	⊗	⊗
205	Status	Status	Data	Data	Data	Data	Data
219	Status	Status	Glitch	Glitch	Glitch	Glitch	Glitch
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14425	Status	Status	Data	Data	Data	Data	Data
14439	Status	Status	Glitch	Glitch	Glitch	Glitch	Glitch

*The headings are not a part of the returned data.
⊗ = Invalid data

Transitional Timing Data

Status Bytes. In transitional timing mode, the status bytes indicate whether a given row in the data contains acquisition information or transition count information.

bits 10-9	bits 8-7	bits 6-5	bits 4-3	bits 2-1
Pod 5	Pod 4	Pod 3	Pod 2	Pod 1

Each pod uses two bits to show what is being represented in the corresponding Information bytes. Bits 10, 8, 6, 4 and 2 are set when the appropiate pod's Information bytes represent acquisition data. When that bit is clear, the next bit shows if the Information bytes represent the first word of a count. Together there are three possible combinations:

10 - This pod's Information bytes contain acquisition data as obtained from the target system.
01 - This pod's Information bytes contain the first word of a count.
00 - This pod's Information bytes contain part of a count other than the first word.

Information Bytes. In the Transitional timing mode the logic analyzer performs the following steps to obtain the information bytes:

1. Four samples of data are taken at 10 nanosecond intervals. The data is stored and the value of the last sample is retained.

2. Four more samples of data are taken. If any of these four samples differ from the last sample of the step 1, then these four samples are stored and the last value is once again retained.

3. If all four samples of step 2 are the same as the last sample taken in step 1, then no data is stored. Instead, a counter is incremented. This process will continue until a group of four samples is found which differs from the retained sample. At this time, the count will be stored in the memory, the counters reset, the current data stored, and the last sample of the four once again retained for comparison.

Note The stored count indicates the number of 40 ns intervals that have elapsed between the old data and the new data.

The rows of the acquisition data may, therefore, be either four rows of data followed by four more rows of data, or four rows of data followed by four rows of count. Rows of count will always be followed by four rows of data except for the last row, which may be either data or count.

Note This process is performed on a pod-by-pod basis. The individual status bits will indicate what each pod is doing.

The following table is just an example. The meaning of the Information bytes (Data or Count) depends upon the corresponding Status bytes.

Example:	Machine 1	Machine 2	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1*
177	Status	Status	Data	Data	Data	Data	Data
191	Status	Status	Data	Data	Data	Data	Data
205	Status	Status	Data	Data	Data	Data	Data
219	Status	Status	Data	Data	Data	Data	Data
233	Status	Status	Data	Count	Count	Data	Data
247	Status	Status	Data	Count	Count	Data	Data
261	Status	Status	Data	Count	Count	Data	Data
275	Status	Status	Data	Count	Count	Data	Data
289	Status	Status	Count	Data	Data	Count	Data
303	Status	Status	Count	Data	Data	Count	Data
317	Status	Status	Count	Data	Data	Count	Data
331	Status	Status	Count	Data	Data	Count	Data
345	Status	Status	Data	Data	Count	Data	Data
359	Status	Status	Data	Data	Count	Data	Data
373	Status	Status	Data	Data	Count	Data	Data
387	Status	Status	Data	Data	Count	Data	Data
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
14457	Status	Status	Data	Data	Data	Data	Data
14471	Status	Status	Data	Data	Data	Data	Data
14485	Status	Status	Data	Data	Data	Data	Data
14499	Status	Status	Data	Data	Data	Data	Data

*The headings are not a part of the returned data.

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     "`
- `"1650 DISP "`
- `"1650 DISP2"`

Additionally, the following sections may also be included, depending on what's available:

- `"SYMBOLS A "`
- `"SYMBOLS B "`
- `"SPA DATA A"`
- `"SPA DATA B"`
- `"INVASM A  "`
- `"INVASM B  "`
- `"COMPARE   "`

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 (31 for the logic analyzer) 4 bytes for the length of the section data in bytes
<section data>	::= format depends on the type of data

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

After P. Index 6

put TAB: All 17 TABS IN BACK OF BOOK