Programming the HP 16550A 1

Introduction

This reference, combined with the *HP 16500A/16501A Programming Reference*, provides you with the information needed to program the HP 16550A logic analyzer module. Each module has its own reference to supplement the mainframe manual since not all mainframes will be configured with the same modules.

About This Manual

This manual is organized in seventeen 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 16550A Logic Analyzer command tree
- Alphabetic command-to-subsystem directory

Chapter 2 contains module level commands.

Chapters 3 through 15 contain the subsystem commands for the logic analyzer.

Chapter 16 contains information on the SYSTem:DATA and SYSTem:SETup commands for this module.

Chapter 17 contains program examples that you can use to do common measurements with the HP 16550A over the bus.

Error messages for the HP 16550A are included in generic system error messages and are in the *HP 16500A/16501A Programming Reference*.

Programming the HP 16550A 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 a basic timing measurement. Additional program examples are in chapter 17.

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 through 10 refers to slot A through J 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/16501A Programming Reference* manual.

Programming the Logic Analyzer

A typical logic analyzer program will do the following:

- 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 acquiring data

Programming Reference

HP 16550A 100-MHz State/ 500-MHz Timing Logic Analyzer

for the HP 16500A Logic Analysis System

Manual Part Number 16550-90902

Printed in the U.S.A. March 1992

Product Warranty

This Hewlett-Packard product has a warranty against defects in material and workmanship for a period of one year from date of shipment. During warranty period, Hewlett-Packard Company will, at its option, either repair or replace products that prove to be defective.

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

For products returned to Hewlett-Packard for warranty service, the Buyer shall prepay shipping charges to Hewlett-Packard and Hewlett-Packard shall pay shipping charges to return the product to the Buyer. However, the Buyer shall pay all shipping charges, duties, and taxes for products returned to Hewlett-Packard from another country.

Hewlett-Packard warrants that its software and firmware designated by Hewlett-Packard for use with an instrument will execute its programming instructions when properly installed on that instrument. Hewlett-Packard does not warrant that the operation of the instrument software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by the Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environmental specifications for the product, or improper site preparation or maintenance.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HEWLETT-PACKARD SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OR MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies THE REMEDIES PROVIDED HEREIN ARE THE BUYER'S SOLE AND EXCLUSIVE REMEDIES. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, SPECIAL INCIDENTAL, OR CONSEQUENTIAL DAMAGES, WHETHER BASED ON CONTRACT, TORT, OR ANY OTHER LEGAL THEORY.

Assistance Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Certification Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

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

Printing History

New editions are complete revisions of the manual. Update packages, which are issued between editions, contain additional and replacement pages to be merged into the manual by the customer. The dates on the title page change only when a new edition or a new update is published. No information is incorporated into a reprinting unless it appears as a prior update; the edition does not change when an update is incorporated.

A software code may be printed before the date; this indicates the version level of the software product at the time of the manual or update was issued. Many product updates and fixes do not require manual changes and, conversely, manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual updates.

Edition 1 March 1992 16550-90902

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	March 1992

Contents

Chapter 6: STRigger (STRace) Subsystem

Chapter 7: SLISt Subsystem

Index

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

Example Program:

```
10   OUTPUT XXX;":SELECT 3"
20   OUTPUT XXX;":MACH1:NAME 'TIMING'"
30   OUTPUT XXX;":MACH1:TYPE TIMING"
40   OUTPUT XXX;":MACH1:ASSIGN 1"
50   OUTPUT XXX;":MACH1:TFORMAT:LABEL 'COUNT',POS,0,0,255"
60   OUTPUT XXX;":MACH1:TTRIGGER:TERM A, 'COUNT', '#HFF'"
70   OUTPUT XXX;":MACH1:TWAVEFORM:RANGE 1E-6"
80   OUTPUT XXX;":MENU 3,5"
90   OUTPUT XXX;":MACH1:TWAVEFORM:INSERT 'COUNT'"
100  OUTPUT XXX;":RMODE SINGLE"
110  OUTPUT XXX;":START"
120  END
```

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

Program Comments

Line 10 selects the logic analyzer in slot C.
Line 20 names machine (analyzer) 1 "TIMING".
Line 30 specifies machine 1 is a timing analyzer.
Line 40 assigns pods 1 and 2 to machine 1.
Line 50 sets up the Timing Format menu by assigning the label COUNT, and assigning a polarity and channels to the label.
Line 60 selects the trigger pattern for the timing analyzer.
Line 70 sets the range to 100 ns (10 times s/div).
Line 80 changes the onscreen display to the Timing Waveforms menu.
Line 90 inserts the label "COUNT" in the Timing Waveform menu.
Line 100 specifies the Single run mode.
Line 110 starts data acquisition.

For more information on the specific logic analyzer commands, refer to chapters 2 through 16.

Mainframe Commands

These commands are part of the HP 16500A/16501A mainframe system and are mentioned here only for reference. For more information on these commands, refer to the *HP 16500A/16501A Programming Reference*.

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 16550A logic analyzer is 32. 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,−1,32,2,2,0,0,5`

A returned string of 12,11,−1,−1,32,2,2,0,0,5 means that an oscilloscope time base card (ID number 11) is loaded in slot B and the oscilloscope acquisition card (ID number 12) is loaded in slot A. The next two slots (C and D) are empty (−1). Slot E contains a logic analyzer module (ID number 32).

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, is 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/16501A 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. It defaults to 0 if it is not specified). The query returns the currently selected and displayed menu.

For the HP 16550A Logic Analyzer:

- X,0 – State/Timing Configuration
- X,1 – Format 1
- X,2 – Format 2
- X,3 – Trigger 1
- X,4 – Trigger 2
- X,5 – Waveform 1
- X,6 – Waveform 2
- X,7 – Listing 1
- X,8 – Listing 2
- X,9 – Mixed Display
- X,10 – Compare 1
- X,11 – Compare 2
- X,12 – Chart 1
- X,13 – Chart 2

Note The menus of an "OFF" machine are not available when only one analyzer is turned on. The Mixed Display is available only when one or both analyzers are state analyzers.

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/16501A Programming Reference*.

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 16550A is divided into module-level commands and subsystem commands. Module-level commands are listed in Chapter 2, "Module Level Commands" and each of the subsystem commands are covered in their individual chapters starting with Chapter 3, "MACHine Subsystem."

Each of these chapters contains a description of the subsystem, syntax diagrams, and the commands in alphabetical order. The commands are shown in long form and short form using upper and lowercase letters. For example, LABel indicates that the long form of the command is LABEL and the short form 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 16550A logic analyzer module. The (x) following 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 10, representing slots A through J, respectively.

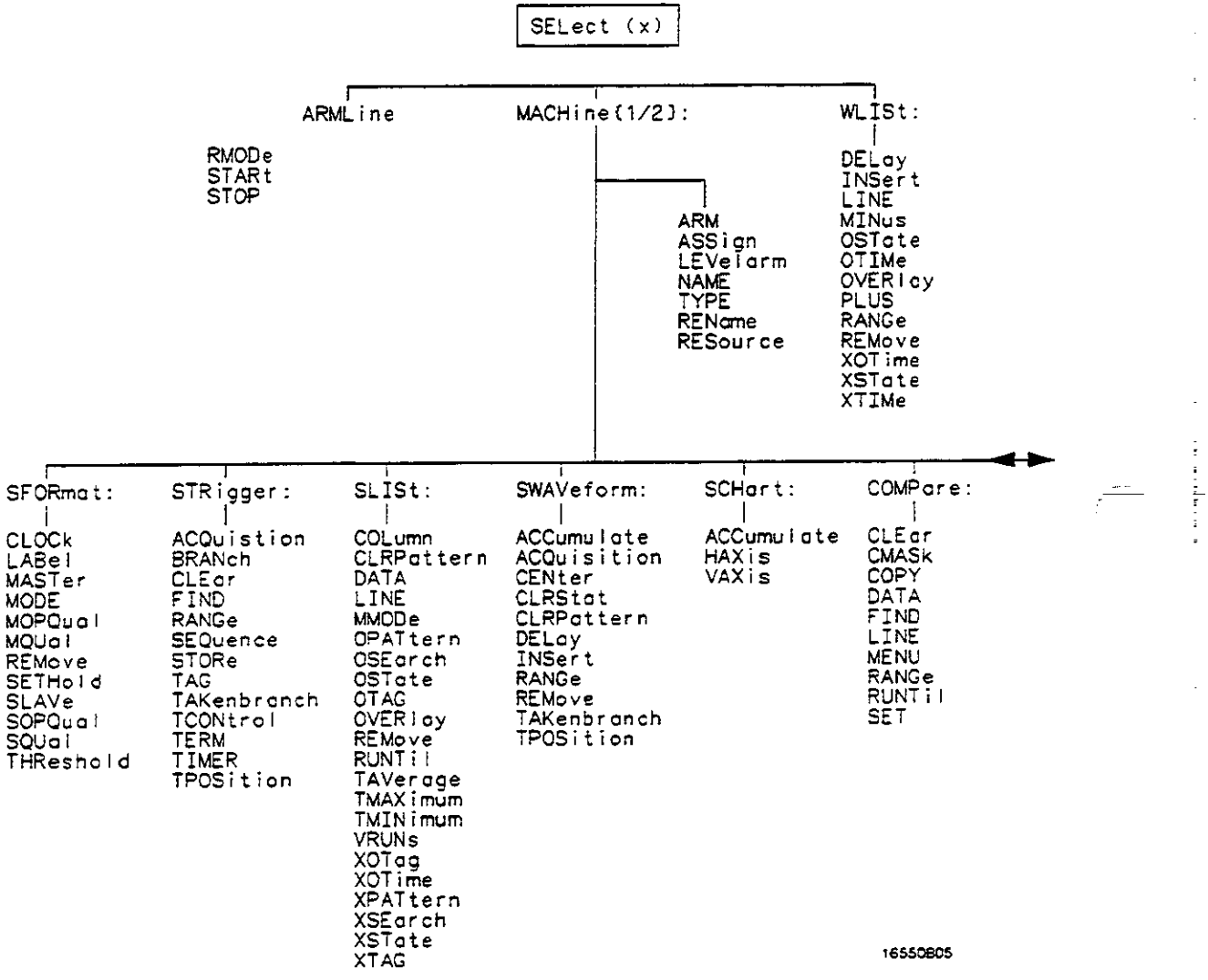


Figure 1-1. HP 16550A Command Tree

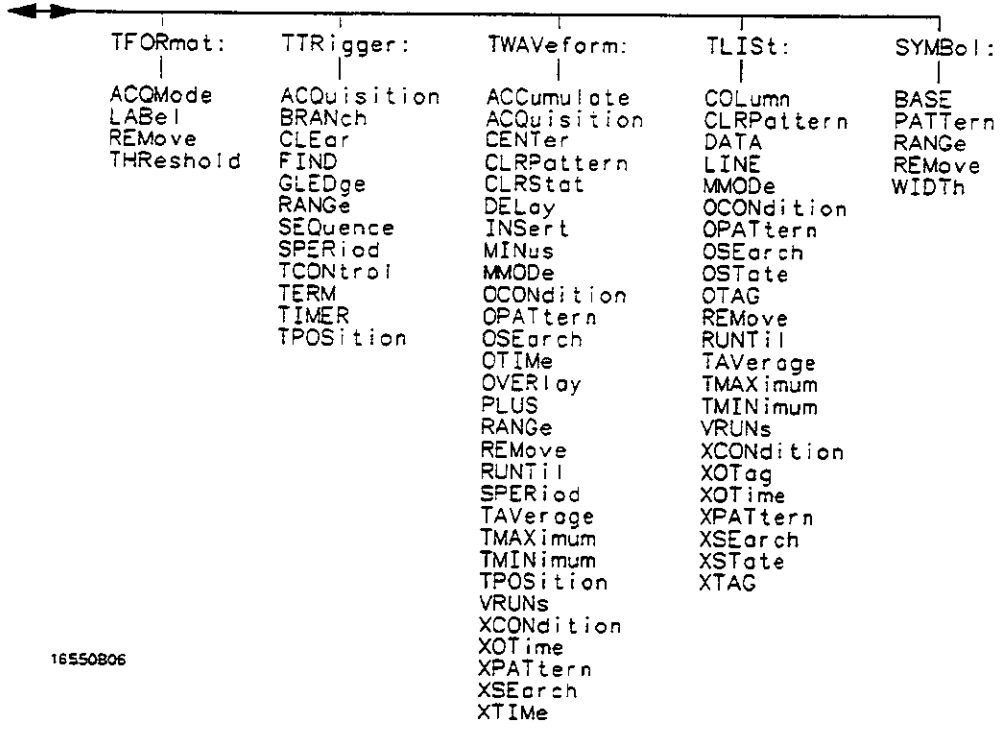


Figure 1-1. HP 16550A Command Tree (continued)

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

Command	Where Used
ACCumulate	SCHart, SWAVeform, TWAVeform
ACQMode	TFORmat
ACQuisition	STRigger, SWAVeform, TTRigger, TWAVeform
ARM	MACHine
ARMline	Module Level Commands
ASSign	MACHine
BASE	SYMBol
BRANch	STRigger, TTRigger
CENter	SWAVeform, TWAVeform
CLEar	COMPare, STRigger, TTRigger
CLOCk	SFORmat
CLRPattern	SLISt, SWAVeform, TLISt, TWAVeform
CLRStat	SWAVeform, TWAVeform
CMASk	COMPare
COLumn	SLISt, TLISt
COPY	COMPare
DATA	COMPare, SLISt, TLISt
DELay	SWAVeform, TWAVeform, WLISt
FIND	COMPare, STRigger, TTRigger
GLEDge	TTRigger
HAXis	SCHart
INSert	SWAVeform, TWAVeform, WLISt
LABel	SFORmat, TFORmat
LEVelarm	MACHine
LINE	COMPare, SLISt, TLISt, WLISt
MASTer	SFORmat
MENU	COMPare
MINus	TWAVeform, WLISt
MMODe	SLISt, TLISt, TWAVeform
MODE	SFORmat
MOPQual	SFORmat
MQUal	SFORmat
NAME	MACHine
OCONdition	TLISt, TWAVeform
OPATtern	SLISt, TLISt, TWAVeform
OSEarch	SLISt, TLISt, TWAVeform
OSTate	SLISt, TLISt, WLISt
OTAG	SLISt, TLISt
OTIMe	TWAVeform, WLISt
OVERlay	SLISt, TWAVeform, WLISt
PATTern	SYMBol
PLUS	TWAVeform, WLISt
RANGe	COMPare, STRigger, SWAVeform, SYMBol, TFORmat, TWAVeform, WLISt,
REMove	SFORmat, SLIST, SWAVeform, SYMBol, TFORmat, TLISt, TWAVeform, WLISt
REName	MACHine
RESource	MACHine
RUNTil	COMPare, SLIST, TLISt, TWAVeform
SEQuence	STRigger, TTRigger
SET	COMPare
SETHold	SFORmat
SLAVe	SFORmat
SOPQual	SFORmat
SPERiod	TFORmat, TWAVeform
SQUal	SFORmat
STORe	STRigger
TAG	STRigger
TAKenbranch	STRigger, SWAVeform
TAVerage	SLIST, TLISt, TWAVeform
TCONtrol	STRigger, TTRigger
TERM	STRigger, TTRigger
THReshold	SFORmat, TFORmat
TIMER	STRigger, TTRigger
TMAXimum	SLIST, TLISt, TWAVeform
TMINimum	SLIST, TLISt, TWAVeform
TPOSition	STRigger, SWAVeform, TTRigger, TWAVeform
TYPE	MACHine
VAXis	SCHart
VRUNs	SLIST, TLISt, TWAVeform
WIDTh	SYMBol
XCONdition	TLISt, TWAVeform
XOTag	SLIST, TLISt
XOTime	SLIST, TLISt, TWAVeform, WLISt
XPATtern	SLIST, TLISt, TWAVeform
XSEarch	SLIST, TLISt, TWAVeform
XSTate	SLIST, TLISt, WLISt
XTAG	SLIST, TLISt
XTIMe	TWAVeform, WLISt

Module Status Reporting

Each module reports its status to the Module Event Status Register (MESR<N>), which in turn reports to the Combined Event Status Register (CESR) in the HP 16500A/16501A mainframe (see *HP 16500A/16501A Programming Reference* 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 16550A'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.

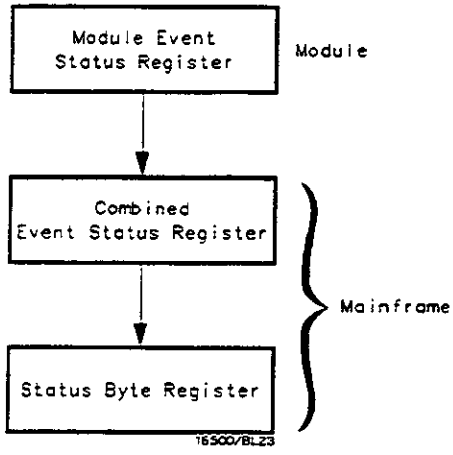


Figure 1-2. Module Status Reporting

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 through 10 refers to the module in slot A through J). 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/16501A Programming Reference* manual.

Command Syntax: `:MESE<N><enable_mask>`

where:

<N> ::= {1|2|3|4|5|6|7|8|9|10} number of slot in which the module resides
<enable_mask> ::= integer from 0 to 255

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

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	Pattern searches failed
2	4	Trigger found
1	2	RNT-Run until satisified
0	1	MC-Measurement complete

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

MESR<N> query

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

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.

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

Query Syntax: `:MESR<N>?`

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

where:

<N> ::= {1|2|3|4|5|6|7|8|9|10} 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	1 = One or more pattern searches failed 0 = Pattern searches did not fail
2	4	1 = Trigger found 0 = Trigger not found
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 16550A logic analyzer module. These commands are:

- ARMLine
- MACHine
- WLISt

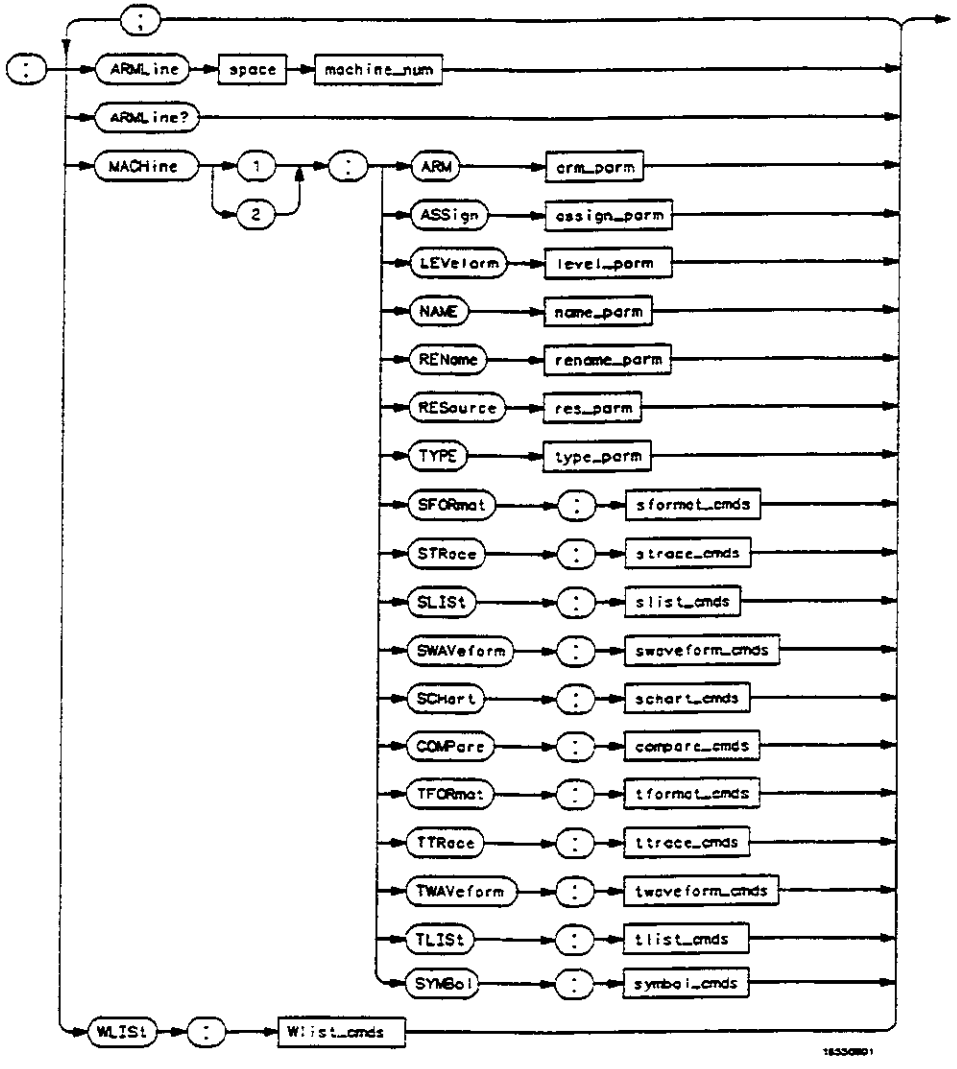


Figure 2-1. Module Level Syntax Diagram

machine_num = *MACHine{1|2}*
arm_parm = *arm parameters (see chapter 3)*
assign_parm = *assignment parameters (see chapter 3)*
level_parm = *level parameters (see chapter 3)*
name_parm = *name parameters (see chapter 3)*
rename_parm = *rename parameters (see chapter 3)*
res_parm = *resource 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)*
tlist_cmds = *timing listing subsystem commands (see chapter 14)*
symbol_cmds = *symbol subsystem commands (see chapter 15)*

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. However, the query is always valid.

Command Syntax: `:ARMLine {MACHine<N>}`

where:

`<N>` ::= {1|2}

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

Query Syntax: `:ARMLine?`

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

Example: `OUTPUT XXX;":ARMLine?"`

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 analyzer type. See chapter 3 for details about the MACHine Subsystem.

Command Syntax: `:MACHine<N>`

where:

<N> ::= {1|2}

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

WLISt selector

The WLISt selector accesses the commands used to place markers and query marker positions in Timing/State Mixed mode. The WLISt subsystem also contains commands that allows you to insert waveforms from other time-correlated machines and modules. The details of the WLISt subsystem are in chapter 4.

Command Syntax: `:WLISt`

Example:

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

MACHine Subsystem 3

Introduction

The MACHine subsystem contains the commands that control the machine level of operation of the logic analyzer. The functions of three of these commands reside in the State/Timing Configuration menu. These commands are:

- ARM
- ASSign
- LEVelarm
- NAME
- TYPE

Even though the functions of the following commands reside in the Format menu they are at the machine level of the command tree and are therefore located in the MACHine subsystem. These commands are:

- RENname
- RESource

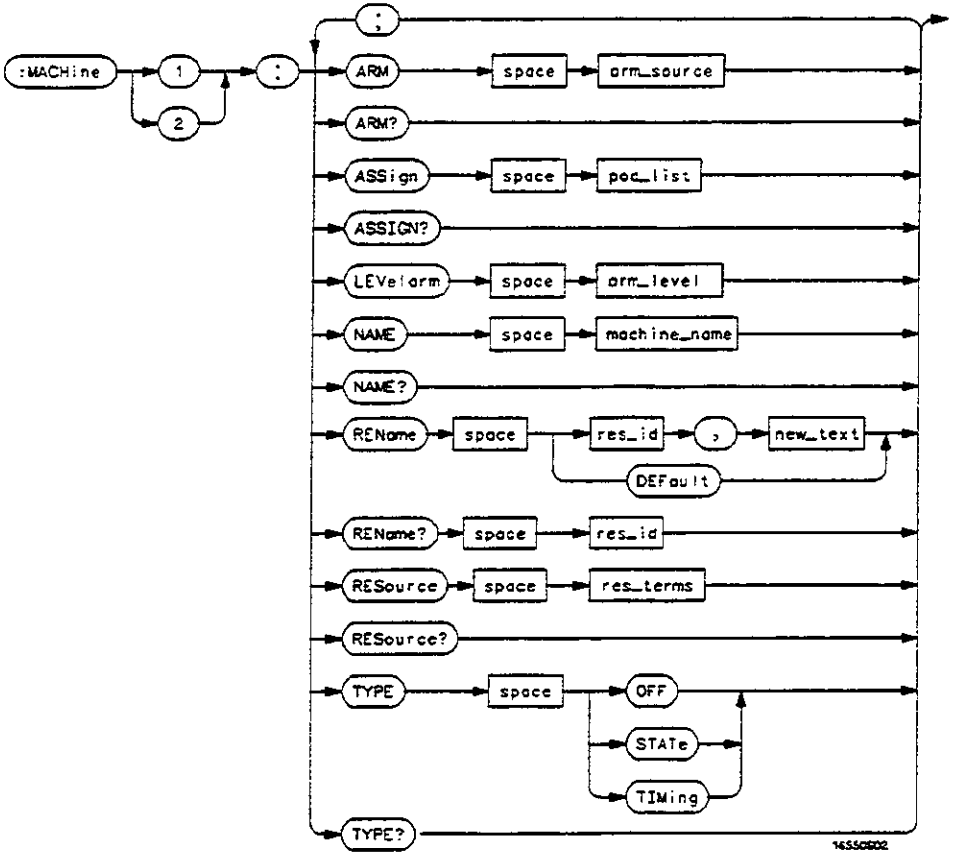


Figure 3-1. Machine Subsystem Syntax Diagram

arm_source = {*RUN* | *INTermodule* | *MACHine* {*1*|*2*}}
pod_list = {*NONE* | <*pod_num*>[, <*pod_num*>]...}
pod_num = {*1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12*}
arm_level = *integer from 1 to 11 representing sequence level*
machine_name = *string of up to 10 alphanumeric characters*
res_id = <*state_terms*> *for state analyzer*
or
{<*state_terms*> |*GLEDge*{*1*|*2*}} *for timing analyzer*
new_text = *string of up to 8 alphanumeric characters*
state_terms = {*A*|*B*|*C*|*D*|*E*|*F*|*G*|*H*|*I*|*J*|*RANGE*{*1*|*2*}|*TIMER*{*1*|*2*}}
res_terms = {<*res_id*>[,<*res_id*>]...}

Figure 3-1. Machine Subsystem Syntax Diagram (continued)

MACHine selector

The MACHine <N> selector specifies which of the two analyzers (machines) available in the HP 16550A the commands or queries following will refer to. Because 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 'TIMING'"`

ARM

command/query

The ARM command specifies the arming source of the specified analyzer (machine). The RUN option disables the arm source. For example, if you do not want to use either the intermodule bus or the other machine to arm the current machine, you specify the RUN option.

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

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

where:

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

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

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

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

Example:
```
OUTPUT XXX;":MACHINE:ARM?"
```

ASSign command/query

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

Note The ASSign command will assign two pods for each pod number you specify because pods must be assigned to analyzers in pairs.

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|6|7|8|9|10|11|12}

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

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

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

Example:
```
OUTPUT XXX;":MACHINE1:ASSIGN?"
```

LEVelarm

command/query

The LEVelarm command allows you to specify the sequence level for a specified machine that will be armed by the Intermodule Bus or the other machine. This command is only valid if the specified machine is on and the arming source is not set to RUN with the ARM command.

The LEVelarm query returns the current sequence level receiving the arming for a specified machine.

Command Syntax: `:MACHine{1|2}:LEVelarm <arm_level>`

where:

<arm_level> ::= integer from 1 to 11 representing sequence level

Example:
```
OUTPUT XXX;":MACHINE1:LEVELARM 2"
```

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

Returned Format: `[:MACHine{1|2}:LEVelarm] <arm_level><NL>`

Example:
```
OUTPUT XXX;":MACHINE1:LEVELARM?"
```

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: `:MACHine{1|2}:NAME?`

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

Example: `OUTPUT XXX;":MACHINE1:NAME?"`

RENname command/query

The RENname command allows you to assign a specific name of up to eight characters to terms A through J, Range 1 and 2, and Timer 1 and 2 in the state analyzer. In the timing analyzer, GLEDge (glitch/edge) 1 and 2 can be renamed in addition to the terms available in the state analyzer. The DEFault option sets all resource term names to the default names assigned when turning on the instrument.

The RENname query returns the current names for specified terms assigned to the specified analyzer.

Command Syntax: `:MACHine{1|2}:REName {<res_id>, <new_text> | DEFault}`

where:

<res_id> ::= <state_terms> for state analyzer
or
::= {<state_terms>|GLEDge{1|2}} for timing analyzer

<new_text> ::= string of up to 8 alphanumeric characters

Example:
```
OUTPUT XXX;":MACHINE1:RENAME A,'DATA'"
```

Query Syntax: `:MACHine{1|2}:RENAME? <res_id>`

Returned Format: `[:MACHine{1|2}:RENAME] <res_id>,<new_text><NL>`

Example:
```
OUTPUT XXX;":MACHINE1:RENAME? D"
```

RESource command/query

The RESource command allows you to assign resource terms A through J, Range 1 and 2, and Timer 1 and 2 to a particular analyzer (machine 1 or 2).

Note In the timing analyzer only, two additional resource terms are available. These terms are GLEDge (Glitch/Edge) 1 and 2. These terms will always be assigned to the the machine that is configured as the timing analyzer.

The RESource query returns the current resource terms assigned to the specified analyzer.

Command Syntax: `:MACHine{1|2}:RESource <res_terms>`

where:

`<res_terms>` `::= {A|B|C|D|E|F|G|H|I|J|TIMer1|TIMer2|RANGe1|RANGe2}`

Example:
```
OUTPUT XXX;":MACHINE1:RESOURCE A,C,RANGE1"
```

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

Returned Format: `[:MACHine{1|2}:RESOURCE] <res_terms>[,<res_terms>,...]<NL>`

Example:
```
OUTPUT XXX;":MACHINE1:RESOURCE?"
```

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:
```
OUTPUT XXX;":MACHINE1:TYPE?"
```

WLISt Subsystem 4

Introduction

The 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. The XSTate and OSTate queries return what states the X and O markers are on. Because 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 state analyzer with time tagging on (use MACHine<N>:STRigger:TAG TIME).

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

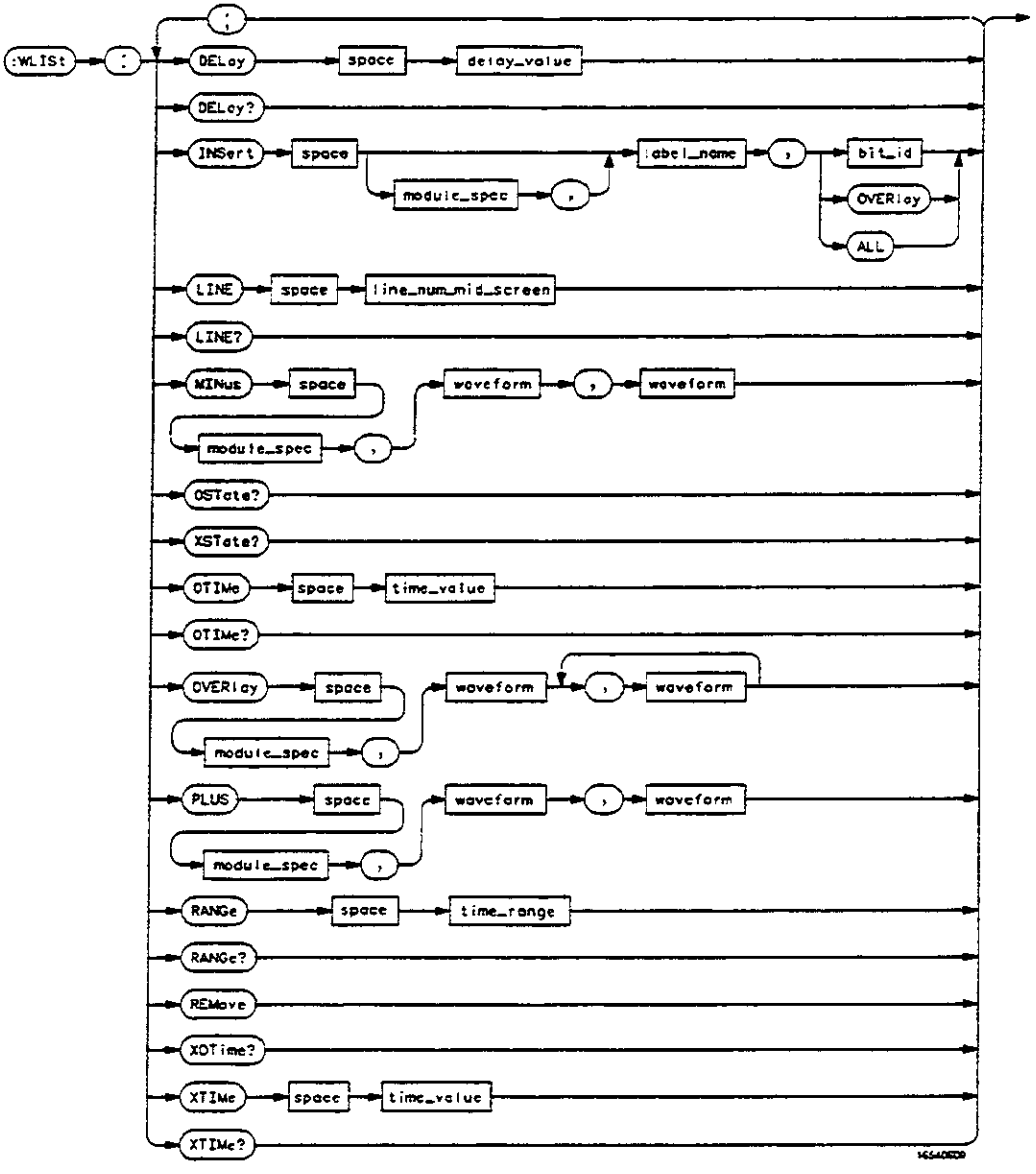


Figure 4-1. WLISt Subsystem Syntax Diagram

delay_value = *real number between* −2500 s and +2500 s
module_spec = {*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|*10*} *(slot where time card is installed*
bit_id = *integer from 0 to 31*
label_name = *string of up to 6 alphanumeric characters*
line_num_mid_screen = *integer from* −8191 to +8191
waveform = *string containing* <*acquisition_spec*> {*1*|*2*}
time_value = *real number*
time_range = *real number between 10 ns and 10 ks*

Figure 4-1. WLISt Subsystem Syntax Diagram (continued)

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. Because 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 or more state analyzers with time tagging on are specified.

Command Syntax: :WLISt

Example:

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

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. If the acquisition mode is automatic, then 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 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}:WLISt:DELay <delay_value>`

where:

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

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

Query Syntax: `:MACHine{1|2}:WLISt:DELay?`

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

Example:
```
OUTPUT XXX;":MACHINE1:WLIST:DELAY?"
```

INSert command

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom up to a maximum of 96 waveforms. Once 96 waveforms are present, each time you insert another waveform, it 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, which is the module specifier. 1 through 10 corresponds to modules A through J. If you do not specify the module, 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 you specify OVERlay, all the bits of the label are displayed as a composite overlaid waveform. If you specify ALL, all the bits are displayed sequentially. If you do not specify the third parameter, ALL is assumed.

Command Syntax: `:MACHine{1|2}:WLISt:INSert [<module_spec>,]<label_name>[,{<bit_id>|OVERlay|ALL}]`

where:

<module_spec> ::= {1|2|3|4|5|6|7|8|9|10}
<label_name> ::= string of up to 6 alphanumeric characters
<bit_id> ::= integer from 0 to 31

Example:

```
OUTPUT XXX;":MACHINE1:WLIST:INSERT 3, 'WAVE',10"
```

Inserting Oscilloscope Waveforms

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

Command Syntax: `:MACHine{1|2}:WLISt:INSert <module_spec>,<label_name>`

where:

<module_spec> ::= {1|2|3|4|5|6|7|8|9|10} slot in which timebase card is installed
<label_name> ::= string of one alpha and one numeric character

Example:

```
OUTPUT XXX;":MACHINE1:WLIST:INSERT 5, 'C1'"
```

LINE

command/query

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

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

Command Syntax: `:MACHine{1|2}:WLISt:LINE <line_num_mid_screen>`

where:

<line_num_mid_screen> ::= integer from −8191 to +8191

Example: `OUTPUT XXX;":MACHINE1:WLIST:LINE 0"`

Query Syntax: `:MACHine{1|2}:WLISt:LINE?`

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

Example: `OUTPUT XXX;":MACHINE1:WLIST:LINE?"`

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, where 1 through 10 refers to slots A through J. The next two parameters specify which waveforms will be subtracted from each other.

Note MINus is only available for oscilloscope waveforms.

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

where:

<module_spec> ::= {1|2|3|4|5|6|7|8|9|10}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)

Example:

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

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:

```
OUTPUT XXX;":WLIST:OSTATE?"
```

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:
```
OUTPUT XXX;":WLIST:OTIME?"
```

OVERlay command

The OVERlay command overlays two or more oscilloscope waveforms and adds the resultant waveform to the current waveform 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}:WLISt:OVERlay <module_number>, <label>[, <label>]...`

where:

<module_spec>	::= {1|2|3|4|5|6|7|8|9|10}
<waveform>	::= string containing <acquisition_spec>{1|2}
<acquisition_spec>	::= {A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)

Example:

```
OUTPUT XXX;":MACHINE1:WLIST: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, where 1 through 10 refers to slots A through J. The next two parameters specify which waveforms will be subtracted from each other.

Note PLUS is only available for oscilloscope waveforms.

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

where:

<module_spec> ::= {1|2|3|4|5|6|7|8|9|10}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)

Example:

```
OUTPUT XXX; ":WLIST: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 10 ns to 10 ks.

The RANGe query returns the current full-screen time.

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

where:

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

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

Query Syntax: `:MACHine{1|2}:WLISt:RANGe?`

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

Example:
```
OUTPUT XXX;":MACHINE1:WLIST:RANGE?"
```

REMove command

The REMove command deletes all waveforms from the display.

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

Example:

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

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}:WLISt:XOTime?`

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

where:

<time_value> ::= real number

Example:

```
OUTPUT XXX;":MACHINE1:WLIST:XOTIME?"
```

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:
```
OUTPUT XXX;":WLIST:XSTATE?"
```

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:
```
OUTPUT XXX;":WLIST:XTIME?"
```

SFORmat Subsystem 5

Introduction

The SFORmat subsystem contains the commands available for the State Format menu in the HP 16550A logic analyzer module. These commands are:

- CLOCk
- LABel
- MASTer
- MODE
- MOPQual
- MQUal
- REMove
- SETHold
- SLAVe
- SOPQual
- SQUal
- THReshold

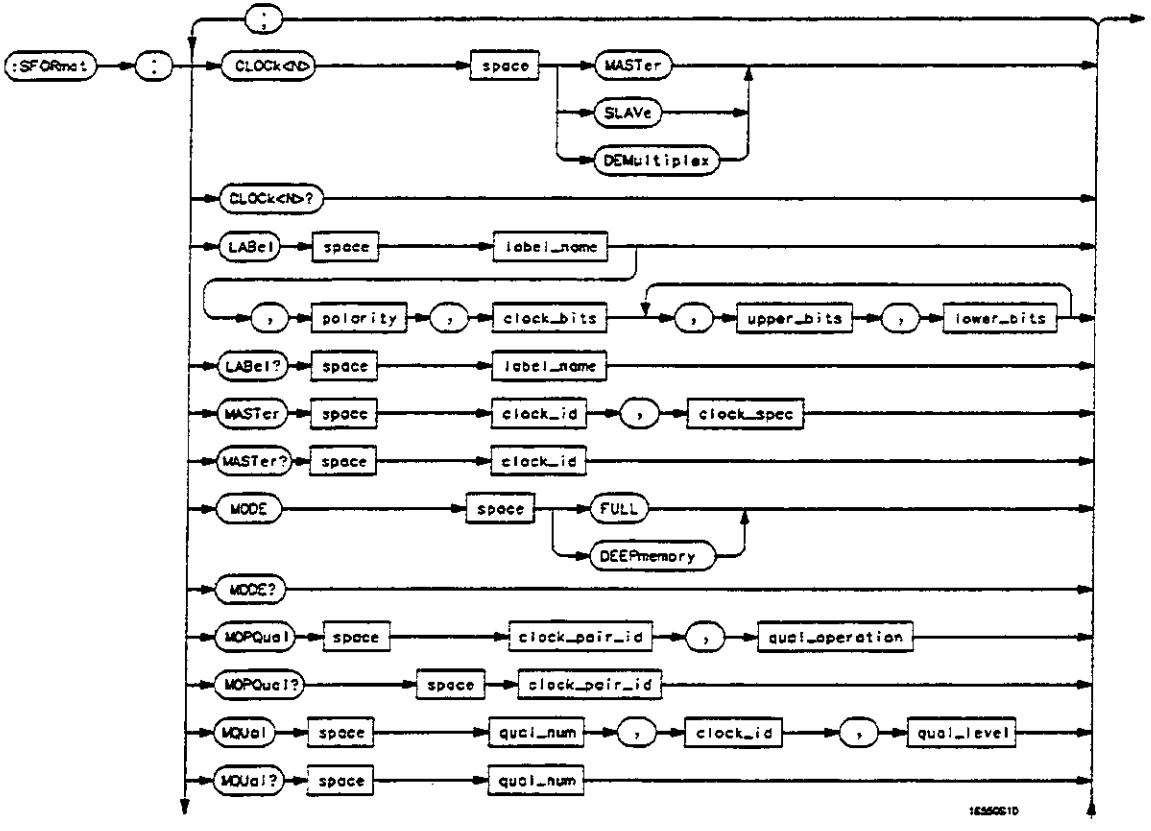
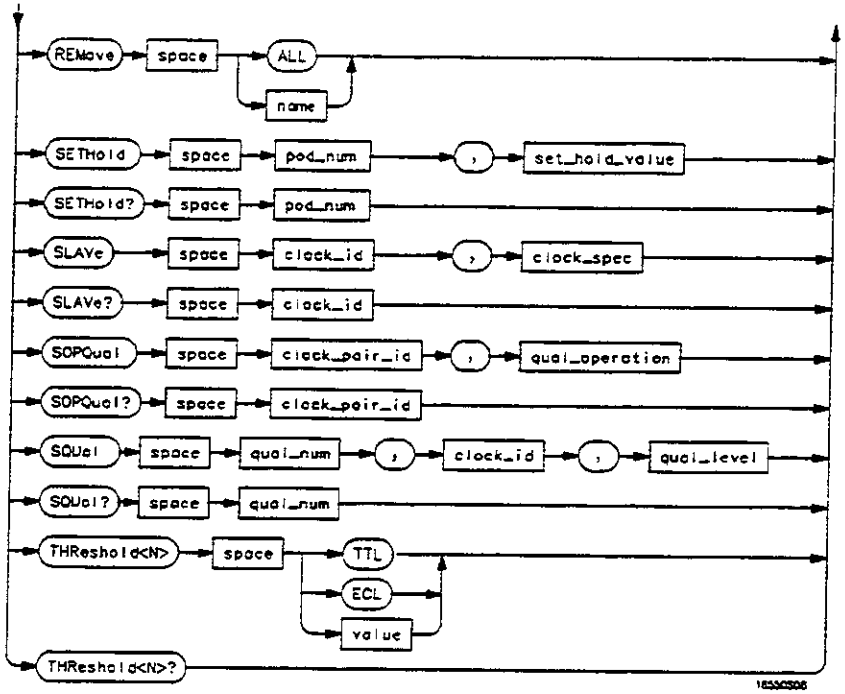


Figure 5-1. SFORmat Subsystem Syntax Diagram



<N> = {{*1* | *2* | *3* | *4* | *5* | *6*}|{*7* | *8* | *9* | *10* | *11* | *12*}}
label_name = *string of up to 6 alphanumeric characters*
polarity = {*POSitive* | *NEGative*}
clock_bits = *format (integer from 0 to 63) for a clock (clocks are assigned in decreasing order)*
upper_bits = *format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)*
lower_bits = *format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)*
clock_id = {*J* | *K* | *L* | *M* | *N* | *P*}
clock_spec = {*OFF* | *RISing* | *FALLing* | *BOTH*}
clock_pair_id = {*1* | *2*}
qual_operation = {*AND*|*OR*}
qual_num = {*1* | *2* | *3* | *4*}
qual_level = {*OFF* | *LOW* | *HIGH*}
pod_num = {*1* | *2*| *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* | *12*}
set_hold_value = {*0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9*}
value = *voltage (real number)* −6.00 to +6.00

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 MASTer option is specified, the pod will sample all 16 channels on the master clock. When the SLAVe option is specified, the pod will sample all 16 channels on the slave clock. When the DEMultiplex option is specified, only one pod of a pod pair can acquire data. The 16 bits of the selected pod will be clocked by the demultiplex master for labels with bits assigned under the Master pod. The same 16 bits will be clocked by the demultiplex slave for labels with bits assigned under the Slave pod. 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|6}|{7|8|9|10|11|12}} 1 through 6 for one card or 1 through 12 for a two-card set

`<clock_mode>` ::= {MASTer | SLAVe | DEMultiplex}

Example:
```
OUTPUT XXX;":MACHINE1:SFORMAT:CLOCK2 MASTER"
```

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

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

Example:
```
OUTPUT XXX; ":MACHINE1:SFORMAT:CLOCK2?"
```

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 when more than 13 pod specifications are listed.

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

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

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

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

Command Syntax: `:MACHine{1|2}:SFORmat:LABel <name>,[<polarity>,<clock_bits>, <upper_bits>, <lower_bits>[,<upper_bits>,<lower_bits>]...]`

where:

<name>	::= string of up to 6 alphanumeric characters
<polarity>	::= {POSitive | NEGative}
<clock_bits>	::= format (integer from 0 to 63) for a clock (clocks are assigned in decreasing order)
<upper_bits>	::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)
<lower_bits>	::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples:

```
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'STAT', POSITIVE, 0,127,40312"
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL 'SIG 1', #B11,#B0000000011111111,
#B0000000000000000 "
```

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

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

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:LABEL? 'DATA'"
```

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 (Master, Slave, and Demultiplexed). Each command deals with only one clock (J,K,L,M,N,P); therefore, a complete clock specification requires six commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed.

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|P}
`<clock_spec>` ::= {OFF|RISing|FALLing|BOTH}

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:
```
OUTPUT XXX;":MACHINE2:SFORMAT:MASTER? <clock_id>"
```

MODE

command/query

The MODE command allows you to select the acquistion mode of the state analyzer. The modes are either full-channel with 4 Kbit of memory depth per channel or half-channel with 8 Kbit of memory depth per channel.

The MODE query returns the current acquistion mode.

Command Syntax: `:MACHine{1|2}:SFORmat:MODE <acq_mode>`

where:

<acq_mode> `::= {FULL|DEEPmemory}`

Example:
```
OUTPUT XXX;":MACHine1:SFORMAT:MODE FULL"
```

Query Syntax: `:MACHine{1|2}:SFORmat:MODE?`

Returned Format: `[:MACHine{1|2}:SFORmat:MODE] <acq_mode><NL>`

Example:
```
OUTPUT XXX;":MACHINE1:SFORMAT:MODE?"
```

MOPQual command/query

The MOPQual (master operation qualifier) command allows you to specify either the AND or the OR operation between master clock qualifier pair 1 and 2, or between master clock qualifier pair 3 and 4. For example, you can specify a master clock operation qualifer 1 AND 2.

The MOPQual query returns the operation qualifier specified for the master clock.

Command Syntax: `:MACHine{1|2}:SFORmat:MOPQual <clock_pair_id>,<qual_operation>`

where:

<clock_pair_id> ::= {1|2}
<qual_operation> ::= {AND|OR}

Example:
```
OUTPUT XXX;":MACHINE1:SFORMAT:MOPQUAL 1,AND"
```

Query Syntax: `:MACHine{1|2}:SFORmat:MOPQUal? <clock_pair_id>`

Returned Format: `[:MACHine{1|2}:SFORmat:MOPQUal <clock_pair_id>] <qual_operation><NL>`

Example:
```
OUTPUT XXX;":MACHine1:SFORMAT:MOPQUAL? 1"
```

MQUal

command/query

The MQUal (master qualifier) command allows you to specify the level qualifier for the master clock.

The MQUal query returns the qualifier specified for the master clock.

Command Syntax: `:MACHine{1|2}:SFORmat:MQUal <qual_num>,<clock_id>,<qual_level>`

where:

<qual_num> ::= {1|2|3|4}
<clock_id> ::= {J|K|L|M|N|P}
<qual_level> ::= {OFF|LOW|HIGH}

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:MQUAL 1,J,LOW"
```

Query Syntax: `:MACHine{1|2}:SFORmat:MQUal? <qual_num>`

Returned Format: `[:MACHine{1|2}:SFORmat:MQUal] <qual_level><NL>`

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:MQUAL? 1"
```

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"
```

SETHold

command/query

The SETHold (setup/hold) command allows you to set the setup and hold specification for the state analyzer.

The SETHold query returns the current setup and hold settings.

Note Even though the command requires integers to specify the setup and hold, the query returns the current settings in a string. For example, if you send the integer 0 for the setup and hold value, the query will return 3.5/0.0 ns as an ASCII string when you have one clock and one edge specified.

Command Syntax: `:MACHine{1|2}:SFORmat:SETHold <pod_num>,<set_hold_value>`

where:

<pod_num> ::= {1|2|3|4|5|6} for a single board or {1|2|3|4|5|6|7|8|9|10|11|12} for a pair of boards

<set_hold_value> ::= integer {0|1|2|3|4|5|6|7|8|9} representing the following setup and hold values:

For one clock and one edge	For one clock and both edges	For Multiple Clocks
0 = 3.5/0.0 ns	0 = 4.0/0.0	0 = 4.5/0.0
1 = 3.0/0.5 ns	1 = 3.5/0.5	1 = 4.0/0.5
2 = 2.5/1.0 ns	2 = 3.0/1.0	2 = 3.5/1.0
3 = 2.0/1.5 ns	3 = 2.5/1.5	3 = 3.0/1.5
4 = 1.5/2.0 ns	4 = 2.0/2.0	4 = 2.5/2.0
5 = 1.0/2.5 ns	5 = 1.5/2.5	5 = 2.0/2.5
6 = 0.5/3.0 ns	6 = 1.0/3.0	6 = 1.5/3.0
7 = 0.0/3.5 ns	7 = 0.5/3.5	7 = 1.0/3.5
N/A	8 = 0.0/4.0	8 = 0.5/4.0
N/A	N/A	9 = 0.0/4.5

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:SETHOLD 1,2"
```

SETHold

Query Syntax: `:MACHine{1|2}:SFORMAT:SETHOLD? <pod_0num>`

Returned Format: `[:MACHine{1|2}:SFORmat:SETHold <pod_num>] <set_hold_value><NL>`

Example:

```
OUTPUT XXX;":MACHINE2:SFORMAT:SETHOLD? 3"
```

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 Slave and Demultiplexed clocking modes. Each command deals with only one clock (J,K,L,M,N,P); therefore, a complete clock specification requires six commands, one for each clock. Edge specifications (RISing, FALLing, or BOTH) are ORed.

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|P}
<clock_spec> ::= {OFF|RISing|FALLing|BOTH}

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:
```
OUTPUT XXX;":MACHINE2:SFORMAT:SLAVE? K"
```

SOPQual command/query

The SOPQual (slave operation qualifier) command allows you to specify either the AND or the OR operation between slave clock qualifier pair 1 and 2, or between slave clock qualifier pair 3 and 4. For example you can specify a slave clock operation qualifer 1 AND 2.

The SOPQual query returns the operation qualifier specified for the slave clock.

Command Syntax: `:MACHine{1|2}:SFORmat:SOPQual <clock_pair_id>,<qual_operation>`

where:

<clock_pair_id> ::= {1|2}
<qual_operation> ::= {AND|OR}

Example:
```
OUTPUT XXX;":MACHine2:SFORMAT:SOPQUAL 1,AND"
```

Query Syntax: `:MACHine{1|2}:SFORmat:SOPQual? <clock_pair_id>`

Returned Format: `[:MACHine{1|2}:SFORmat:SOPQual <clock_pair_id>] <qual_operation><NL>`

Example:
```
OUTPUT XXX;":MACHiNE2:SFORMAT:SOPQUAL? 1"
```

SQUal

command/query

The SQUal (slave qualifier) command allows you to specify the level qualifier for the slave clock.

The SQUal query returns the qualifier specified for the slave clock.

Command Syntax: `:MACHine{1|2}:SFORmat:SQUal <qual_num>,<clock_id>,<qual_level>`

where:

<qual_num> ::= {1|2|3|4}
<clock_id> ::= {J|K|L|M|N|P}
<qual_level> ::= {OFF|LOW|HIGH}

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:SQUAL 1,J,LOW"
```

Query Syntax: `:MACHine{1|2}:SFORmat:SQUal?<qual_num>`

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

Example:
```
OUTPUT XXX;":MACHINE2:SFORMAT:SQUAL? 1"
```

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 −6.00 V to +6.00 V in 0.05 volt increments.

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|6|7|8|9|10|11|12}
<value>	::= voltage (real number) −6.00 to +6.00
TTL	::= default value of +1.6 V
ECL	::= default value of −1.3 V

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:
```
OUTPUT XXX;":MACHINE1:SFORMAT:THRESHOLD4?"
```

STRigger (STRace) Subsystem 6

Introduction

The STRigger subsystem contains the commands available for the State Trigger menu in the HP 16550A logic analyzer module. The State Trigger subsystem will also accept the STRace selector as used in previous HP 16500-Series Logic Analyzer modules to eliminate the need to rewrite programs containing STRace as the selector keyword. The STRigger subsystem commands are:

- ACQuisition
- BRANch
- CLEar
- FIND
- RANGe
- SEQuence
- STORe
- TAG
- TAKenbranch
- TCONtrol
- TERM
- TIMER
- TPOSition

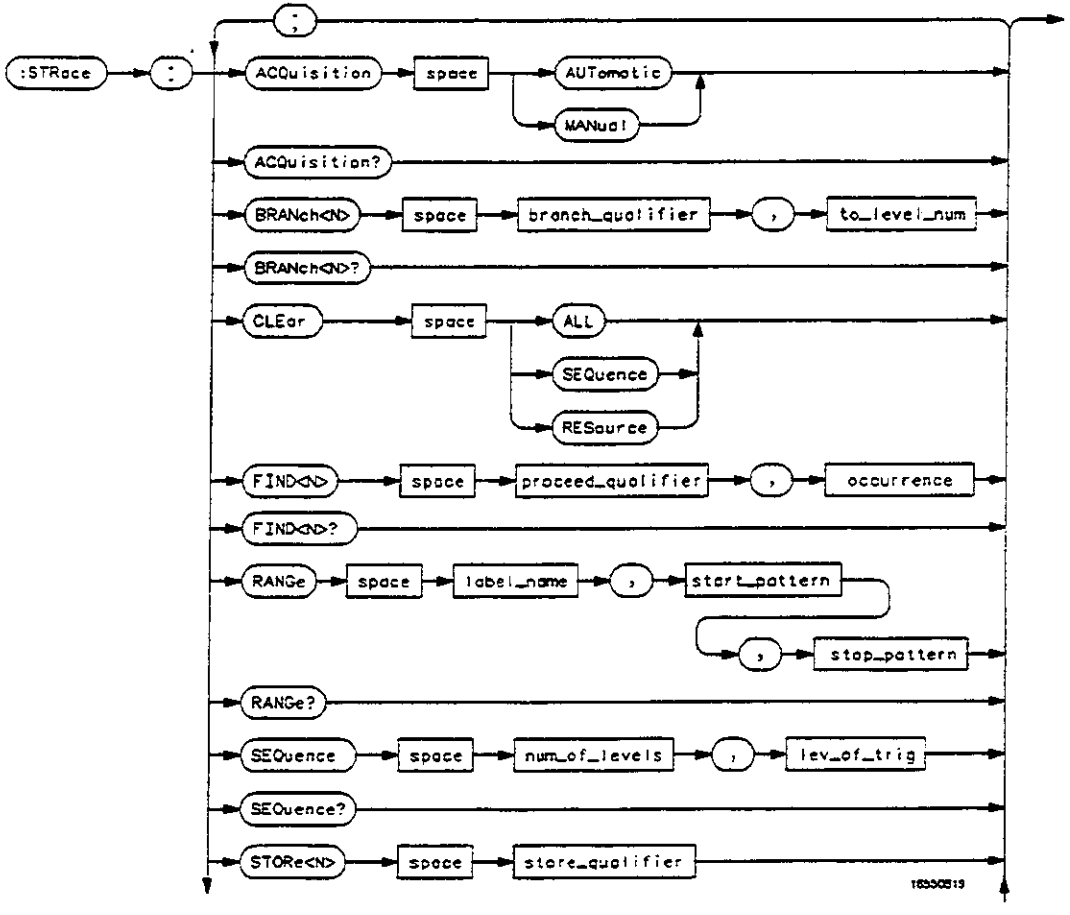


Figure 6-1. STRigger Subsystem Syntax Diagram

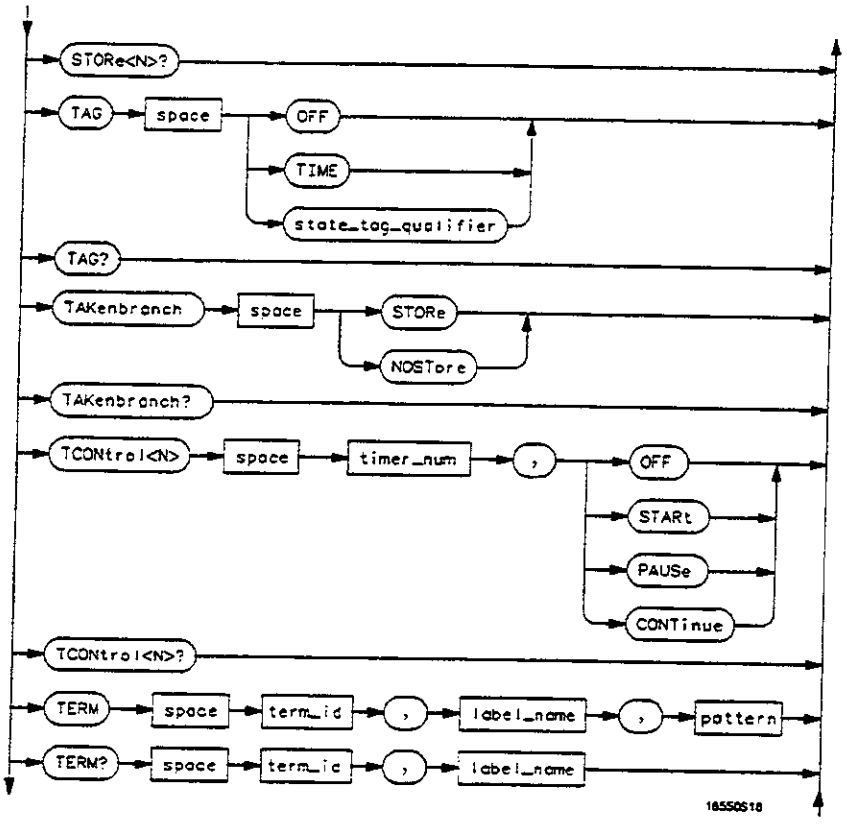
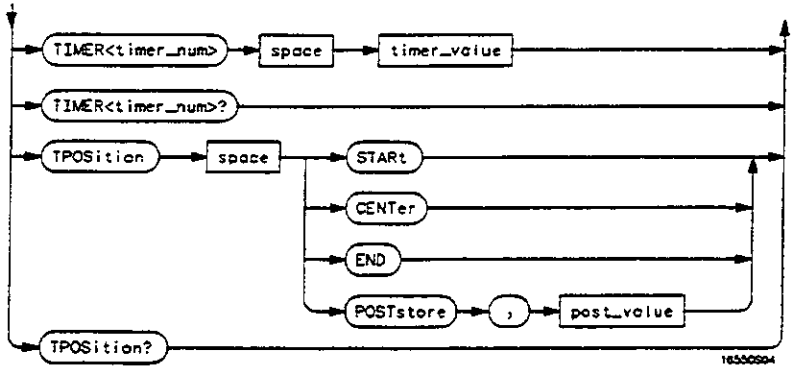


Figure 6-1. STRigger Subsystem Syntax Diagram (continued)



branch_qualifier = *<qualifier>*
to_lev_num = *integer from 1 to last level*
proceed_qualifier = *<qualifier>*
occurrence = *number from 1 to 1048575*
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}... }"
num_of_levels = *integer from 2 to 12*
lev_of_trig = *integer from 1 to (number of existing sequence levels* − 1)
store_qualifier = *<qualifier>*
state_tag_qualifier = *<qualifier>*
timer_num = {1|2}
timer_value = *400 ns to 500 seconds*
term_id = {A|B|C|D|E|F|G|H|I|J}
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 = *see "Qualifier" on page 6-5*
post_value = *integer from 0 to 100 representing percentage*

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

Qualifier

The qualifier for the state trigger subsystem can be terms A through J, Timer 1 and 2, and Range 1 and 2. In addition, qualifiers can be the NOT boolean function of terms, timers, and ranges. The qualifier can also be an expression or combination of expressions as shown below and figure 6-2, "Complex Qualifier," on page 6-11.

The following parameters show how qualifiers are specified in all commands of the STRigger subsystem that use **<qualifier>**.

where:

```
<qualifier>          ::= { "ANYSTATE" | "NOSTATE" | "<expression>" }

<expression>         ::= {<expression1a> | <expression1b> | <expression1a> OR
                         <expression1b> | <expression1a> AND <expression1b>}

<expression1a>       ::= {<expression1a_term> | (<expression1a_term> [ OR
                         <expression1a_term>]* ) | (<expression1a_term> [ AND <expression1a_term>]* )}

<expression1a_term>  ::= { <expression2a> | <expression2b> | <expression2c> | <expression2d> }

<expression1b>       ::= {<expression1b_term> | ( <expression1b_term> [ OR
                         <expression1b_term>]* ) | (<expression1b_term> [ AND <expression1b_term>]* )}

<expression1b_term>  ::= {<expression2e> | <expression2f> | <expression2g> | <expression2h>}

<expression2a>       ::= {<term3a> | <term3b> | (<term3a> <boolean_op> <term3b>)}
<expression2b>       ::= {<term3c> | <range3a> | (<term3c> <boolean_op> <range3a>)}
<expression2c>       ::= {<term3d>}
<expression2d>       ::= {<term3e> | <timer3a> | (<term3e> <boolean_op> <timer3a>)}
<expression2e>       ::= {<term3f> | <term3g> | (<term3f> <boolean_op> <term3g>)}
<expression2f>       ::= {<term3h> | <range3b> | (<term3h> <boolean_op> <range3b>)}
<expression2g>       ::= {<term3i>}
<expression2h>       ::= {<term3j> | <timer3b> | (<term3e> <boolean_op> <timer3b>)}

<boolean_op>         ::= {AND | NAND | OR | NOR | XOR | NXOR}
```

```
<term3a>    ::= { A | NOTA }
<term3b>    ::= { B | NOTB }
<term3c>    ::= { C | NOTC }
<term3d>    ::= { D | NOTD }
<term3e>    ::= { E | NOTE }
<term3f>    ::= { F | NOTF }
<term3g>    ::= { G | NOTG }
<term3h>    ::= { H | NOTH }
<term3i>    ::= { I | NOTI }
<term3j>    ::= { J | NOTJ }

<range3a>   ::= { IN_RANGE1 | OUT_RANGE1 }
<range3b>   ::= { IN_RANGE2 | OUT_RANGE2 }

<timer3a>   ::= { TIMER1< | TIMER1> }
<timer3b>   ::= { TIMER2< | TIMER2> }
```

Qualifier Rules The following rules apply to qualifiers:

- Qualifiers are quoted strings and, therefore, need quotes.
- Expressions are evaluated from left to right.
- Parenthesis are used to change the order evaluation and, therefore, are optional.
- An expression must map into the combination logic presented in the combination pop-up menu within the STRigger menu (see figure 6-2 on page 6-11).

Examples:

```
'A'
'( A OR B )'
'(( A OR B ) AND C )'
'(( A OR B ) AND C AND IN_RANGE2 )'
'(( A OR B ) AND ( C AND IN_RANGE1 ))'
'IN_RANGE1 AND ( A OR B ) AND C'
```

STRigger (STRace)

selector

The STRigger (STRace) (State Trigger) 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}:STRigger`

Example:

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

ACQuisition command/query

The ACQuisition command allows you to specify the acquisition mode for the State analyzer.

The ACQuisition query returns the current acquisition mode specified.

Command Syntax: `:MACHine{1|2}:STRigger:ACQuisition {AUTOmatic|MANual}`

Example: `OUTPUT XXX;":MACHINE1:STRIGGER:ACQUISITION AUTOMATIC"`

Query Syntax: `:MACHine{1|2}:STRigger:ACQuisition?`

Returned Format: `[:MACHine{1|2}:STRigger:ACQuisition] {AUTOmatic|MANual}<NL>`

Example: `OUTPUT XXX;":MACHINE1:STRIGGER:ACQUISITION?"`

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.

The terms used by the branch qualifier (A through J) are defined by the TERM command. The meaning of IN_RANGE and OUT_RANGE 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 State Trigger menu. 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 statements are all correct and have the same meaning. Notice that the conventional rules for precedence are not followed. The expressions are evaluated from left to right.

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'C AND D OR F OR G', 1"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 '((C AND D)  OR (F OR G))', 1"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'F OR (C AND D) OR G',1"
```

Figure 6-2 shows a complex expression as seen in the State Trigger menu.

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

BRANch

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

where:

<N>	::= integer from 1 to <number_of_levels>
<to_level_number>	::= integer from 1 to <number_of_levels>
<number_of_levels>	::= integer from 2 to the number of existing sequence levels (maximum 12)
<branch_qualifier>	::= <qualifier> see "Qualifier" on page 5

Examples:

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

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

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

Example:

```
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH3?"
```

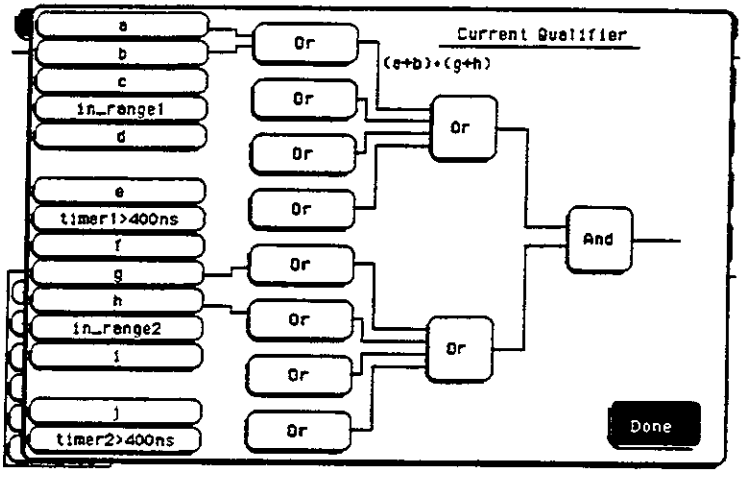


Figure 6-2. Complex qualifier

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

```
OUTPUT XXX;":MACHINE1:STRIGGER:BRANCH1 '((A OR B) AND (G OR H))', 2"
```

Note Terms A through E, RANGE 1, and TIMER 1 must be grouped together and terms F through J, RANGE 2, and TIMER 2 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 IN_RANGE2) AND (C OR H)) is not allowed because the term C cannot be specified in the E through J group.

In the first level, the operators you can use are AND, NAND, OR, NOR, XOR, NXOR. 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.

CLEar command

The CLEar command allows you to clear all settings in the State Trigger menu and replace them with the default, clear only the Sequence levels, or clear only the resource term patterns.

Command Syntax: `:MACHine{1|2}:STRigger:CLEar {All|SEQuence|RESource}`

Example:

```
OUTPUT XXX;":MACHINE1:STRIGGER:CLEAR RESOURCE"
```

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. In the sequence level where the trigger is specified, the FIND command specifies the trigger qualifier (see SEQuence command).

The terms A through J are defined by the TERM command. The meaning of IN_RANGE and OUT_RANGE is determined by the RANGe command. Expressions are limited to what you could manually enter through the State Trigger 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}:STRigger:FIND<N> <proceed_qualifier>,<occurrence>`

where:

<N>	::= integer from 1 to (number of existing sequence levels − 1)
<occurrence>	::= integer from 1 to 1048575
<proceed_qualifier>	::= <qualifier> see "Qualifier" on page 6-5

Examples:

```
OUTPUT XXX;":MACHINE1:STRIGGER:FIND1 'ANYSTATE', 1"
OUTPUT XXX;":MACHINE1:STRIGGER:FIND3 '((NOTA AND NOTB) OR G)', 1"
```

Query Syntax: `:MACHine{1|2}:STRigger:FIND4?`

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

Example:

```
OUTPUT XXX;":MACHINE1:STRIGGER:FIND<N>?"
```

RANGe command/query

The RANGe command allows you to specify a range recognizer term for 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 Because a label can only be defined across a maximum of two pods, a range term is only available across a single label; therefore, the end points of the range cannot be split between labels.

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

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

Command Syntax:

```
:MACHine{1|2}:STRigger: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:STRIGGER:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":MACHINE1:STRIGGER:RANGE 'ABC', '#B00001111', '#HCF' "
```

Query Syntax: `:MACHine{1|2}:STRigger:RANGe?`

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

Example:
```
OUTPUT XXX;":MACHINE1:STRIGGER:RANGE?"
```

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 12 when the analyzer is armed by the RUN key.

The SEQuence query returns the current sequence specification.

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

where:

<number_of_levels> ::= integer from 2 to 12
<level_of_trigger> ::= integer from 1 to (number of existing sequence levels − 1)

Example:
```
OUTPUT XXX;":MACHINE1:STRIGGER:SEQUENCE 4,3"
```

Query Syntax: `:MACHine{1|2}:STRigger:SEQuence?`

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

Example:
```
OUTPUT XXX;":MACHINE1:STRIGGER:SEQUENCE?"
```

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 J are defined by the TERM command. The meaning of IN_RANGE1 and 2 and OUT_RANGE1 and 2 is determined by the RANGe command.

Expressions are limited to what you could manually enter through the State Trigger 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 on page 6-11.

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

Command Syntax: `:MACHine{1|2}:STRigger:STORe<N> <store_qualifier>`

where:

<N> ::= an integer from 1 to the number of existing sequence levels (maximum 12)

<store_qualifier> ::= <qualifier> see "Qualifier" on page 6-5

Examples:

```
OUTPUT XXX;":MACHINE1:STRIGGER:STORE1 'ANYSTATE'"
OUTPUT XXX;":MACHINE1:STRIGGER:STORE2 'OUT_RANGE1'"
OUTPUT XXX;":MACHINE1:STRIGGER:STORE3 '(NOTC AND NOTD AND NOTH)'"
```

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

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

Example:

```
OUTPUT XXX;":MACHINE1:STRIGGER:STORE4?"
```

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 J are defined by the TERM command. The terms IN_RANGE1 and 2 and OUT_RANGE1 and 2 are defined by the RANGe command.

Expressions are limited to what you could manually enter through the State Trigger 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 on page 6-11.

The TAG query returns the current count tag specification.

Command Syntax: `:MACHine{1|2}:STRigger:TAG {OFF|TIME|<state_tag_qualifier>}`

where:

`<state_tag_qualifier>` ::= <qualifier> see "Qualifier" on page 6-5

Examples:
```
OUTPUT XXX;":MACHINE1:STRIGGER:TAG OFF"
OUTPUT XXX;":MACHINE1:STRIGGER:TAG TIME"
OUTPUT XXX;":MACHINE1:STRIGGER:TAG '(IN_RANGE OR NOTF)'"
OUTPUT XXX;":MACHINE1:STRIGGER:TAG '((IN_RANGE OR A) AND E)'"
```

Query Syntax: `:MACHine{1|2} :STRigger:TAG?`

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

Example:
```
OUTPUT XXX;":MACHINE1:STRIGGER:TAG?"
```

TAKenbranch

command/query

The TAKenbranch command allows you to specify whether the state causing the branch is stored or not stored for the specified machine. The state causing the branch is defined by the BRANch command.

The TAKenbranch query returns the current setting.

Command Syntax: `:MACHine{1|2}:STRigger:TAKenbranch {STORe|NOSTore}`

Example: `OUTPUT XXX;":MACHINE2:STRIGGER:TAKENBRANCH STORE"`

Query Syntax: `:MACHine{1|2}:STRigger:TAKenbranch?`

Returned Format: `[:MACHine{1|2}:STRigger:TAKenbranch] {STORe|NOSTore}<NL>`

Example: `OUTPUT XXX;":MACHINE2:STRIGGER:TAKENBRANCH?`

TCONtrol command/query

The TCONtrol (timer control) command allows you to turn off, start, pause, or continue the timer for the specified level. The time value of the timer is defined by the TIMER command. There are two timers and they are available for either machine but not both machines simultaneously.

The TCONtrol query returns the current TCONtrol setting of the specified level.

Command Syntax:
```
:MACHine{1|2}:STRigger:TCONtrol<N> <timer_num>,
{OFF|STARt|PAUSe|CONTinue}
```

where:

<N> ::= integer from 1 to the number of existing sequence levels (maximum 12)

<timer_num> ::= {1|2}

Example:
```
OUTPUT XXX;":MACHINE2:STRIGGER:TCONTROL6 1, PAUSE"
```

Query Syntax:
```
:MACHine{1|2}:STRigger:TCONTROL<N>? <timer_num>
```

Returned Format:
```
[:MACHine{1|2}:STRigger:TCONTROL<N> <timer_num>]
{OFF|STARt|PAUSe|CONTinue}<NL>
```

Example:
```
OUTPUT XXX;":MACHINE2:STRIGGER:TCONTROL?6 1"
```

TERM

command/query

The TERM command allows you to 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. Because 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.

All 10 terms (A through J) are available for either machine but not both simultaneously. If you send the TERM command to a machine with a term that has not been assigned to that machine, an error message "Legal command but settings conflict" is returned.

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

Command Syntax:

```
:MACHine{1|2}:STRigger:TERM <term_id>,<label_name>,<pattern>
```

where:

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

Query Syntax:

```
:MACHine{1|2}:STRigger:TERM? <term_id>,<label_name>
```

Returned Format:

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

Example:

```
OUTPUT XXX;":MACHINE1:STRIGGER:TERM? B,'DATA' "
```

TIMER command/query

The TIMER command sets the time value for the specified timer. The limits of the timer are 400 ns to 500 seconds in 16 ns to 500 μs increments. The increment value varies with the time value of the specified timer. There are two timers and they are available for either machine but not both machines simultaneously.

The TIMER query returns the current time value for the specified timer.

Command Syntax: `:MACHine{1|2}:STRigger:TIMER{1|2} <time_value>`

where:

<time_value> ::= real number from 400 ns to 500 seconds in increments which vary from 16 ns to 500 μs.

Example:
```
OUTPUT XXX;":MACHINE1:STRIGGER:TIMER1 100E-6"
```

Query Syntax: `:MACHine{1|2}:STRigger:TIMER{1|2}?`

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

Example:
```
OUTPUT XXX;":MACHINE1:STRIGGER:TIMER1?"
```

TPOSition

command/query

The TPOSition (trigger position) command allows you to set the trigger at the start, center, end or at any position in the trace (poststore). Poststore is defined as 0 to 100 percent with a poststore of 100 percent being the same as start position and a poststore 0 percent being the same as an end trace.

The TPOSition query returns the current trigger position setting.

Command Syntax:

```
:MACHine{1|2}:STRigger:TPOSition {STARt|CENTer|END|
POSTstore,<poststore>}
```

where:

<poststore> ::= integer from 0 to 100 representing percentage of poststore.

Examples:

```
OUTPUT XXX;":MACHINE1:STRIGGER:TPOSITION END"
OUTPUT XXX;":MACHINE1:STRIGGER:TPOSITION POSTstore,75"
```

Query Syntax:

```
:MACHine{1|2}:STRigger:TPOSition?
```

Returned Format:

```
[:MACHine{1|2}:STRigger:TPOSition] {STARt|CENTer|END|
POSTstore,<poststore>}<NL>
```

Example:

```
OUTPUT XXX;":MACHINE1:STRIGGER:TPOSITION?"
```

SLISt Subsystem 7

Introduction

The SLISt subsystem contains the commands available for the State Listing menu in the HP 16550A logic analyzer module. These commands are:

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

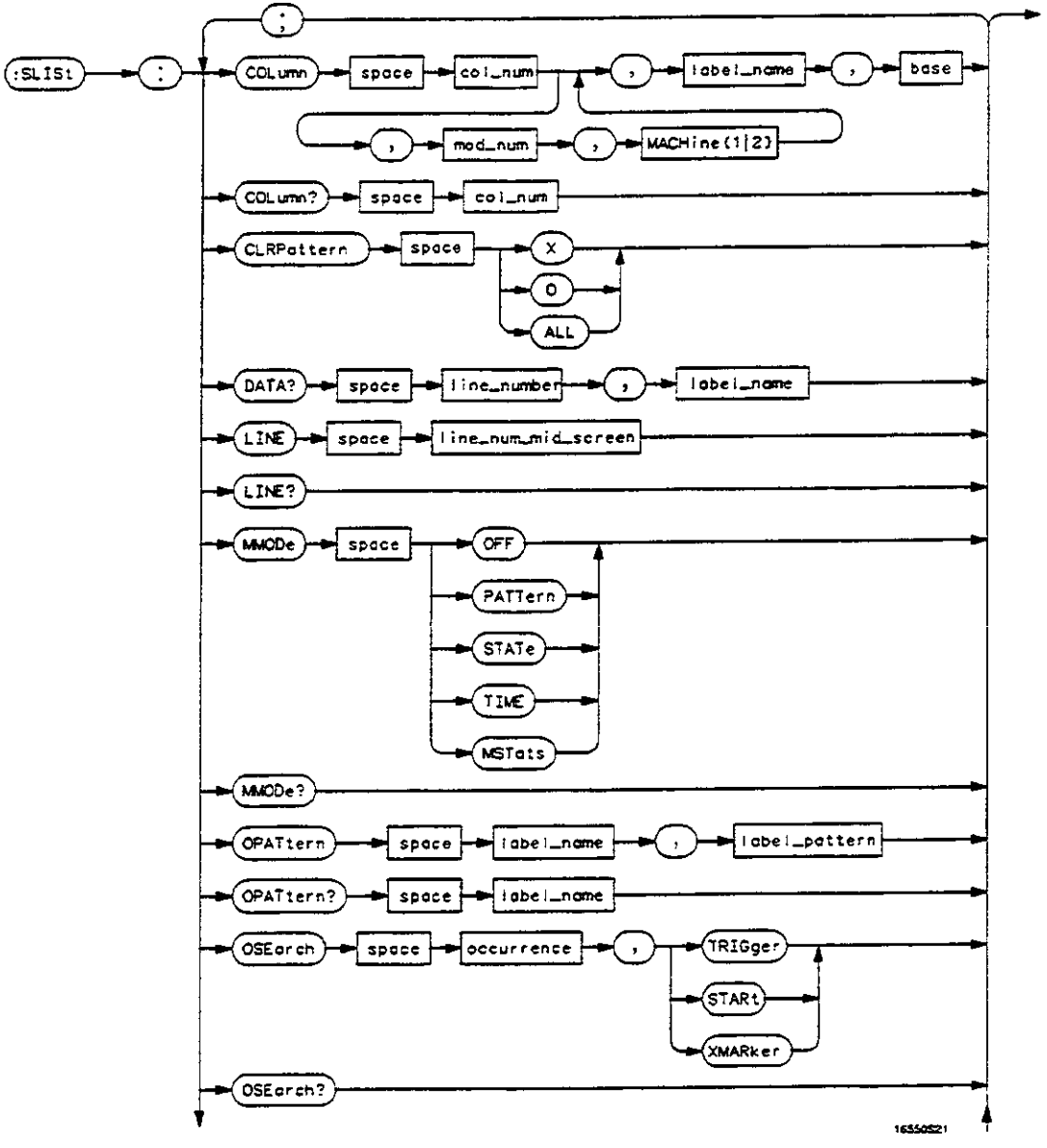


Figure 7-1. SLISt Subsystem Syntax Diagram

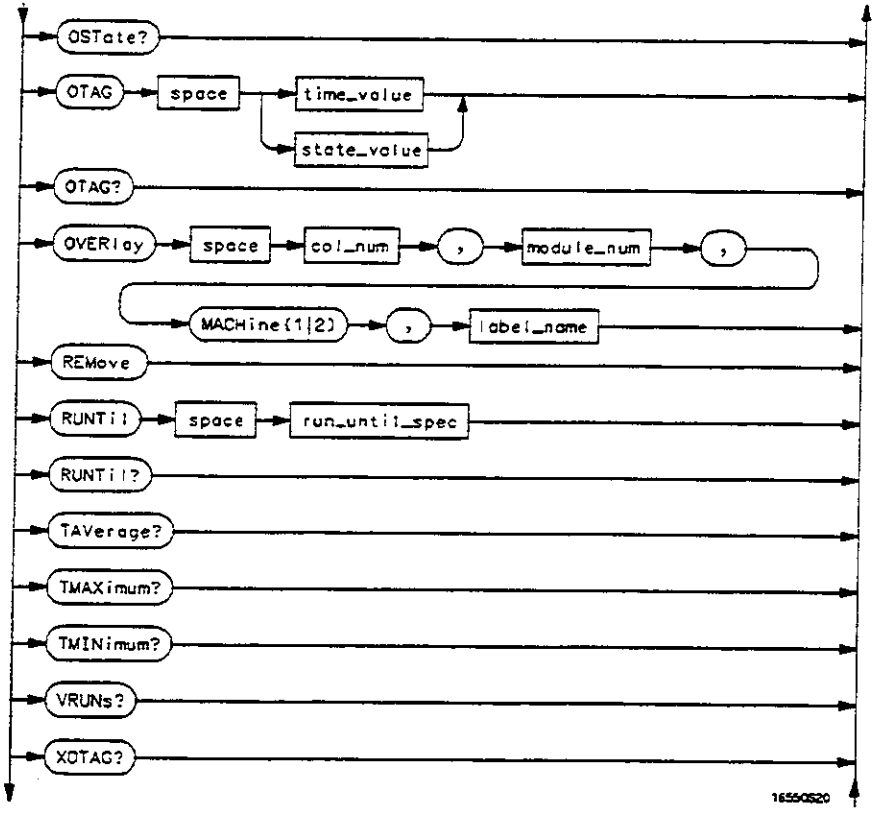
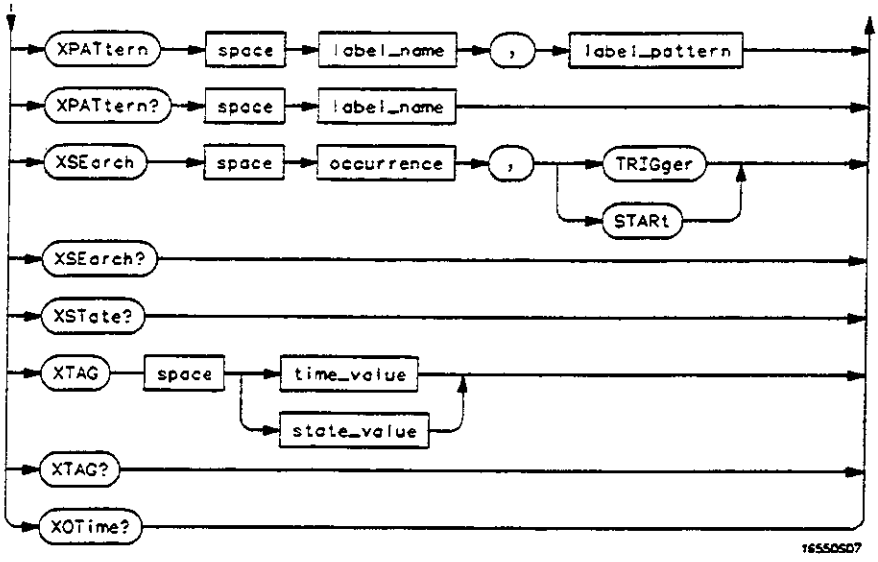


Figure 7-1. SLISt Subsystem Syntax Diagram (continued)



module_num = {*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|*10*}
mach_num = {*1*|*2*}
col_num = *integer from 1 to 61*
line_number = *integer from* −8191 to +8191
label_name = *a string of up to 6 alphanumeric characters*
base = {*BINary*|*HEXadecimal*|*OCTal*|*DECimal*|*TWOS*|*ASCii*|*SYMBol*|*IASSembler*} *for labels or*
{*ABSolute*|*RELative*} *for tags*
line_num_mid_screen = *integer from* −8191 to +8191
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* −8191 to +8192
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 61 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.

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

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.

Command Syntax:

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

where:

<col_num> ::= integer from 1 to 61

<module_num> ::= {1|2|3|4|5|6|7|8|9|10}

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

<base> ::= {BINary|HEXadecimal|OCTal|DECimal|TWOS|ASCii|SYMBol|ASSembler} for labels

or

::= {ABSolute|RELative} for tags

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:COLUMN 4,'A',HEX"
```

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:

```
OUTPUT XXX;":MACHINE1:SLIST:COLUMN? 4"
```

CLRPattern command

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Command Syntax: `:MACHine{1|2}:SWAVeform:CLRPattern {X|0|ALL}`

Example:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:CLRPATTERN X"
```

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 −8191 to +8191

<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:
```
OUTPUT XXX;":MACHINE1:SLIST:DATA? 512, 'RAS'"
```

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 highlights at the center of the screen.

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

Command Syntax: `:MACHine{1|2}:SLISt:LINE <line_num_mid_screen>`

where:

`<line_num_mid_screen>` ::= integer from −8191 to +8191

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

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

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. Because 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' "
```

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

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

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:OPATTERN? 'A'"
```

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 OSEarch 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 −8191 to +8191

`<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:
```
OUTPUT XXX;":MACHINE1:SLIST:OSEARCH?"
```

OSTate

query

The OSTate query returns the line number in the listing where the O marker resides (−8191 to +8191). 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 −8191 to +8191, or 32767

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:OSTATE?"
```

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, or returns 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:
```
OUTPUT XXX;":MACHINE1:SLIST:OTAG?"
```

OVERlay command

The OVERlay command allows you to add time-correlated labels from other modules or machines to the state listing.

Command Syntax:

```
:MACHine{1|2}:SLISt:OVERlay <col_num>,<module_num>,MACHine{1|2},
<label_name>
```

where:

<col_num>	::= integer from 1 to 61
<Module_num>	::= {1|2|3|4|5|6|7|8|9|10}
<label_name>	::= a string of up to 6 alphanumeric characters

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:OVERlay,25,5,MACHINE2,'DATA'"
```

REMove command

The REMove command removes all labels, except the leftmost label, from the listing menu.

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

Example:

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

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, when 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 8 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. The analyzer 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:
```
OUTPUT XXX;":MACHINE1:SLIST:RUNTIL?"
```

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:

```
OUTPUT XXX;":MACHINE1:SLIST:TAVERAGE?"
```

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:
```
OUTPUT XXX;":MACHINE1:SLIST:TMAXIMUM?"
```

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:

```
OUTPUT XXX;":MACHINE1:SLIST:TMINIMUM?"
```

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:

```
OUTPUT XXX;":MACHINE1:SLIST:VRUNS?"
```

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:

```
OUTPUT XXX;":MACHINE1:SLIST:XOTAG?"
```

XOTime query

The XOTime 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:XOTime?`

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

where:

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

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:XOTIME?"
```

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' "
```

Query Syntax:

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

Returned Format:

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

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:XPATTERN? 'A'"
```

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 −8191 to +8191
<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:
```
OUTPUT XXX;":MACHINE1:SLIST:XSEARCH?"
```

XSTate

query

The XSTate query returns the line number in the listing where the X marker resides (−8191 to +8191). 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 −8191 to +8191, or 32767

Example:

```
OUTPUT XXX;":MACHINE1:SLIST:XSTATE?"
```

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, or retruns 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:
```
OUTPUT XXX;":MACHINE1:SLIST:XTAG?"
```

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 96 channels identified by label name and bit number. The 11 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
- ACQuisition
- CENter
- CLRPattern
- CLRStat
- DELay
- INSert
- RANGe
- REMove
- TAKenbranch
- TPOSition

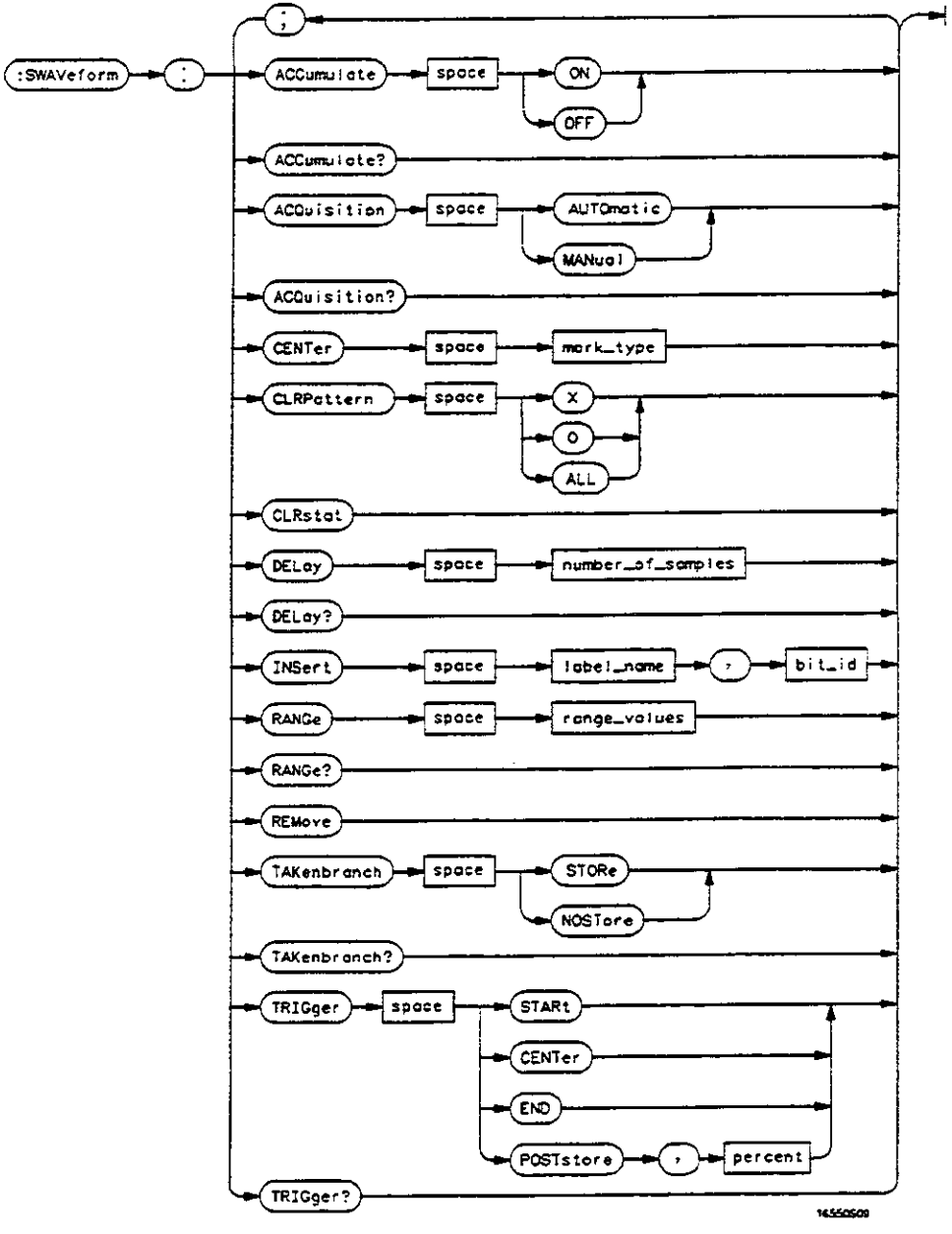


Figure 8-1. SWAVeform Subsystem Syntax Diagram

number_of_samples = *integer from* −8191 to +8191
label_name = *string of up to 6 alphanumeric characters*
bit_id = {*OVERlay*|<*bit_num*>|*ALL*}
bit_num = *integer representing a label bit from 0 to 31*
range_values = *integer from 10 to 5000 (representing (10 × states/Division))*
mark_type = {*X*|*O*|*XO*|*TRIGger*}
percent = *integer from 0 to 100*

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

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 40"
```

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 characters, "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: `OUTPUT XXX;":MACHINE1:SWAVEFORM:ACCUMULATE?"`

ACQuisition command/query

The ACQuisition command allows you to specify the acquisition mode for the state analyzer. The acquisition modes are automatic and manual.

The ACQusition query returns the current acquisition mode.

Command Syntax: `:MACHine{1|2}:SWAVeform:ACQuisition {AUTOmatic|MANual}`

Example: `OUTPUT XXX;":MACHINE2:SWAVEFORM:ACQUISITION AUTOMATIC"`

Query Syntax: `MACHine{1|2}:SWAVeform:ACQuisition?`

Returned Format: `[MACHine{1|2}:SWAVeform:ACQuisition] {AUTOmatic|MANual}<NL>`

Example: `OUTPUT XXX;":MACHINE2:SWAVEFORM:ACQUISITION?"`

CENTer command

The CENTer command allows you to center the waveform display about the specified markers. The markers are placed on the waveform in the SLISt subsystem.

Command Syntax: `:MACHine{1|2}:SWAVeform:CENTer <marker_type>`

where:

<marker_type> ::= {X|O|XO|TRIGger}

Example:

```
OUTPUT XXX;":MACHINE1:SWAVEFORM:CENTER X"
```

CLRPattern command

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Command Syntax: `:MACHine{1|2}:SWAVeform:CLRPattern {X|O|ALL}`

Example:

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

CLRStat command

The CLRStat command allows you to clear the waveform statistics without having to stop and restart the acquisition.

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

Example:

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

DELay command/query

The DELay command allows you to specify the number of samples between the State trigger and the horizontal center of the screen for the waveform display. The allowed number of samples is from −8191 to +8191.

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 −8191 to +8191

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:
```
OUTPUT XXX;":MACHINE1:SWAVEFORM:DELAY?"
```

INSert command

The INSert command allows you to add waveforms to the state waveform display. Waveforms are added from top to bottom on the screen. When 96 waveforms are present, inserting additional waveforms replaces the last waveform. Bit numbers are zero based, so a label with 8 bits is referenced as bits 0 through 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> | ALL}
<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 (states/Div) on the front panel. A number between 10 and 5000 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 5000

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE 80"
```

Query Syntax: `MACHine{1|2}:SWAVeform:RANGe?`

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

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:RANGE?"
```

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"
```

TAKenbranch command/query

The TAKenbranch command allows you to control whether the states that cause branching are stored or not stored. This command is only available when the acquisition mode is set to manual.

The TAKenbranch query returns the current setting.

Command Syntax: `MACHine{1|2}:SWAVeform:TAKenbranch {STORe|NOSTore}`

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TAKENBRANCH STORE"
```

Query Syntax: `MACHine{1|2}:SWAVeform:TAKenbranch?`

Returned Format: `[MACHine{1|2}:SWAVeform:TAKenbranch] {STORe|NOSTore}<NL>`

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TAKENBRANCH?"
```

TPOSition command/query

The TPOSition command allows you to control where the trigger point is placed. The trigger point can be placed at the start, center, end, or at a percentage of post store. The post store option is the same as the User Defined option when setting the trigger point from the front panel.

The TPOSition command is only available when the acquisition mode is set to manual.

The TPOSition query returns the current trigger setting.

Command Syntax: `MACHine{1|2}:SWAVeform:TPOSition {STARt|CENTer|END|POSTstore,<percent>}`

where:

<percent> ::= integer from 1 to 100

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TPOSITION CENTER"
```

Query Syntax: `MACHine{1|2}:SWAVeform:TPOSition?`

Returned Format:
```
[MACHine{1|2}:SWAVeform:TPOSition] {STARt|CENTer|END|POSTstore,
<percent>}<NL>
```

Example:
```
OUTPUT XXX;":MACHINE2:SWAVEFORM:TPOSition?"
```

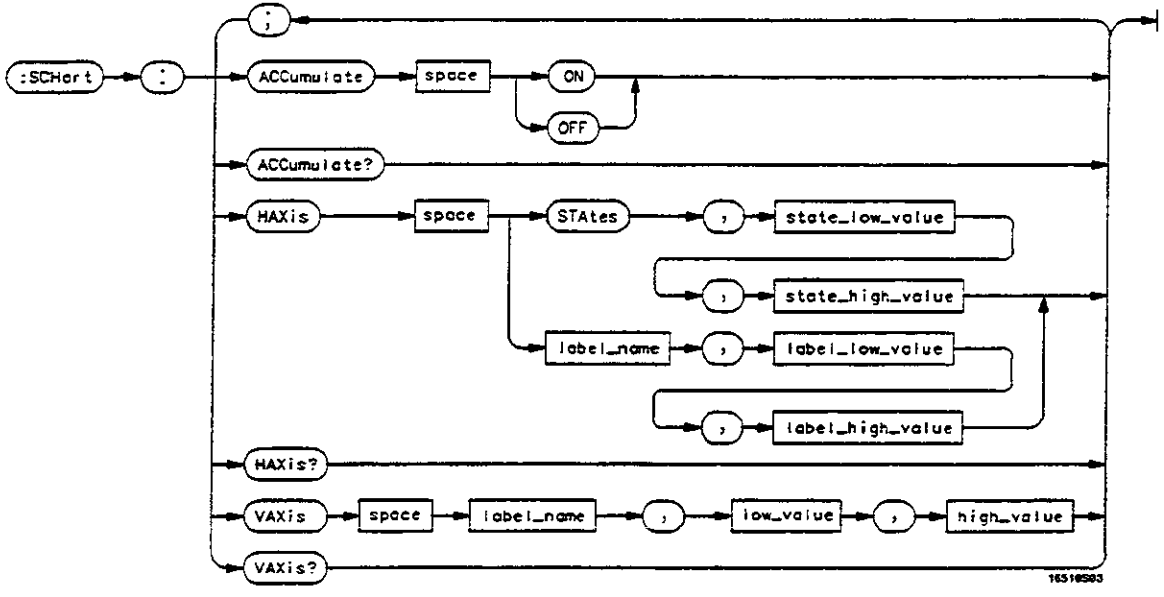
SCHart Subsystem 9

Introduction

The State Chart subsystem provides the commands necessary for programming the HP 16550A'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. Because 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. 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 –8191 to +8191*
state_high_value = *integer from <state_low_value> to +8191*
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: `OUTPUT XXX;":MACHINE1:SCHART:ACCUMULATE?"`

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 −8191 to +8191
<state_high_value>	::= integer from <state_low_value> to +8191
<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 'READ', '−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:

```
OUTPUT XXX;":MACHINE1:SCHART:HAXIS?"
```

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>`

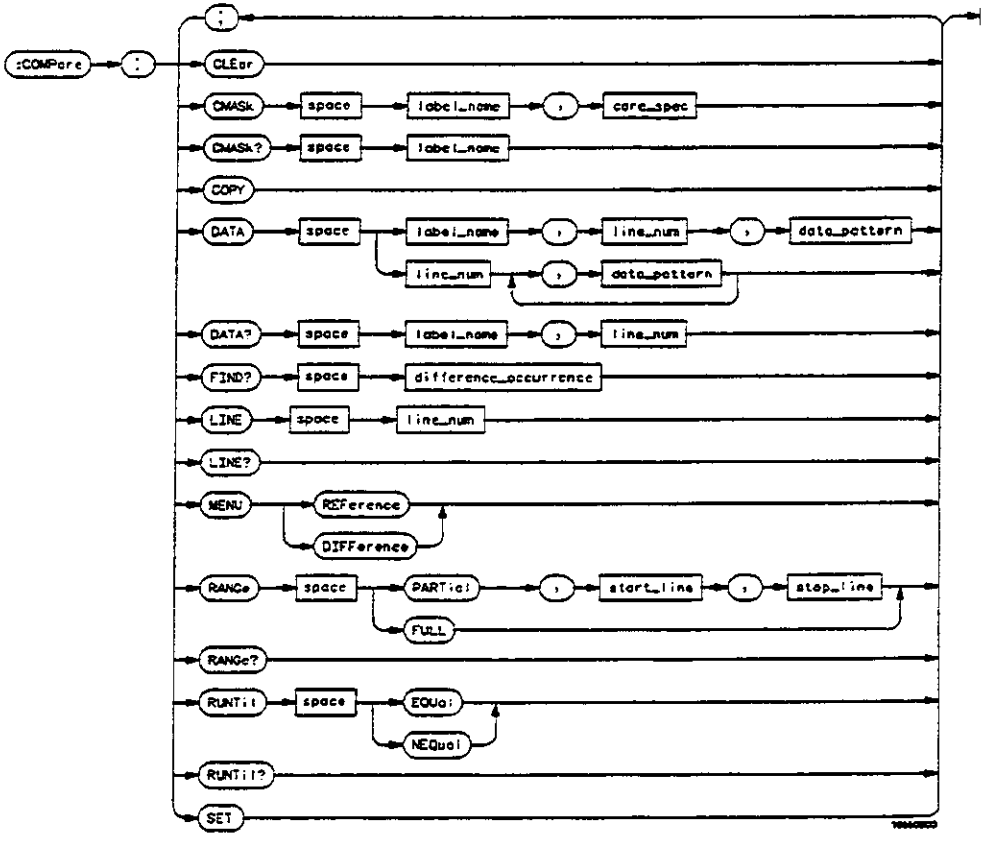
Example:
```
OUTPUT XXX;":MACHINE1:SCHART:VAXIS?"
```

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:

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



label_name = *string of up to 6 characters*
care_spec = *string of characters* "{*|.}..."
* = *care*
. = *don't care*
line_num = *integer from –8191 to +8191*
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 8192*
start_line = *integer from –8191 to +8191*
stop_line = *integer from <start_line> to +8191*

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"
```

CLEar command

The CLEar command clears all "don't cares" in the reference listing and replaces them with zeros except when the CLEar command immediately follows the SET command (see SET command).

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

Example:

```
OUTPUT XXX;":MACHINE2:COMPARE:CLEAR
```

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 'DATA', '*.**..**'"
```

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; although, 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 to 6 alphanumeric characters

<line_num> ::= integer from –8191 to +8191

<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'"
```

DATA

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 8192

<line_number> ::= integer from –8191 to +8191

Example:

```
OUTPUT XXX;":MACHINE2:COMPARE:FIND? 26"
```

LINE

command/query

The LINE command allows you to center the compare listing data about a specified line number.

The LINE query returns the current line number specified.

Command Syntax: `:MACHine{1|2}:COMPare:LINE <line_num>`

where:

<line_num> ::= integer from –8191 to +8191

Example:
```
OUTPUT XXX;":MACHINE2:COMPARE:LINE -511"
```

Query Syntax: `:MACHine{1|2}:COMPare:LINE?`

Returned Format: `[:MACHine{1|2}:COMPare:LINE] <line_num>}<NL>`

Example:
```
OUTPUT XXX;":MACHINE4:COMPARE:LINE?"
```

MENU command

The MENU command allows you to display the reference or the difference listings in the Compare menu.

Command Syntax: `:MACHine{1|2}:COMPare:MENU {REFerence|DIFFerence}`

Example:
```
OUTPUT XXX;":MACHINE2:COMPARE:MENU REFERENCE"
```

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 acquire 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 −8191 to +8191
<stop_line> ::= integer from <start_line> to +8191

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$[100]
20 OUTPUT 707;":SELECT 2"
30 OUTPUT 707;":MACHINE1:COMPARE:RANGE?"
40 ENTER 707;String$
50 PRINT "RANGE IS ";String$
60 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 8 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: OUTPUT XXX;":MACHINE2:COMPARE:RUNTIL?"

SET command

The SET command sets every state in the reference listing to "don't cares." If you send the SET command by mistake you can immediately send the CLEar command to restore the previous data. This is the only time the CLEar command will not replace "don't cares" with zeros.

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

Example:

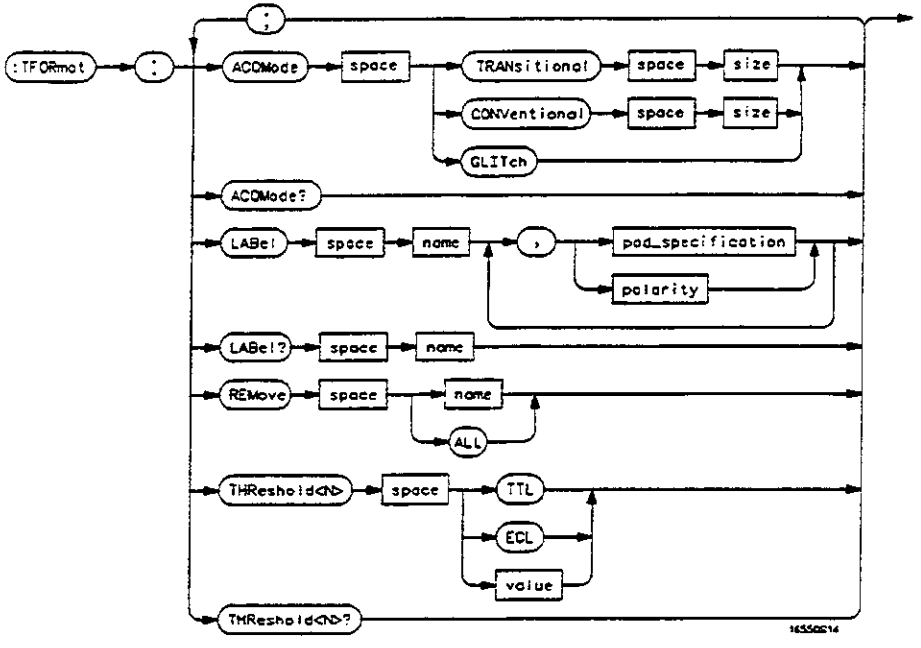
```
OUTPUT XXX;":MACHINE2:COMPARE:SET"
```

TFORmat Subsystem 11

Introduction

The TFORmat subsystem contains the commands available for the Timing Format menu in the HP 16550A logic analyzer module. These commands are:

- ACQMode
- LABel
- REMove
- THReshold



size = {*FULL* | *HALF*}
<N> = {{*1* | *2* | *3* | *4* | *5* | *6*} | {*7* | *8* | *9* | *10* | *11* | *12*}}
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)* −6.00 to +6.00

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:ACQMODE?"
```

ACQMode command/query

The ACQMode (acquisition mode) command allows you to select the acquisition mode for the timing analyzer. The options are:

- conventional mode at full-channel 250 MHz
- conventional mode at half-channel 500 Mhz
- transitional mode at full-channel 125 MHz
- transitional mode at half-channel 250 MHz
- glitch mode.

The ACQMode query returns the current acquisition mode.

Command Syntax: `:MACHine{1|2}:TFORmat:ACQMode {TRANSitional <size>|CONVentional <size>|GLITch}`

where:

`<size>` `::= {FULL|HALF}`

Example:
```
OUTPUT XXX;":MACHINE2:TFORMAT:ACQMODE TRANSITIONAL HALF"
```

Query Syntax: `:MACHine{1|2}:TFORmat:ACQMode?`

Returned Format: `[:MACHine{1|2}:TFORmat:ACQMode] {TRANSitional <size>|CONVentional <size>|GLITch}<NL>`

Example:
```
OUTPUT XXX;":MACHINE2:TFORMAT:ACQMODE?"
```

LABel command/query

The LABel command allows you to specify polarity and to assign channels to new or existing labels. If the specified label name does not match an existing label name, a new label will be created.

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 pods 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 13 pod specifications are listed.

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

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

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

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

LABel

Command Syntax: `:MACHine{1|2}:Tformat:LABel <name>,[<polarity>,<clock_bits>, <upper_bits>, <lower_bits>[,<upper_bits>,<lower_bits>]...]`

where:

<name> ::= string of up to 6 alphanumeric characters
<polarity> ::= {POSitive | NEGative}
<clock_bits> ::= format (integer from 0 to 63) for a clock (clocks are assigned in decreasing order)
<upper_bits> ::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)
<lower_bits> ::= format (integer from 0 to 65535) for a pod (pods are assigned in decreasing order)

Examples:

```
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'STAT', POSITIVE, 0,127,40312"
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL 'SIG 1', #B11,#B0000000011111111,
#B0000000000000000 "
```

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

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

Example:

```
OUTPUT XXX;":MACHINE2:TFORMAT:LABEL? 'DATA'"
```

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 −6.00 V to +6.00 V in 0.05 volt increments.

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|6|7|8|9|10|11|12}
<value> ::= voltage (real number) −6.00 to +6.00
TTL ::= default value of +1.6 V
ECL ::= default value of −1.3 V

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:
```
OUTPUT XXX;":MACHINE1:TFORMAT:THRESHOLD2?"
```

TTRigger (TTRace) Subsystem 12

Introduction

The TTRigger subsystem contains the commands available for the Timing Trigger menu in the HP 16550A logic analyzer module. The Timing Trigger subsystem will also accept the TTRace selector as used in previous HP 16500-series logic analyzer modules to eliminate the need to rewrite programs containing TTRace as the selector keyword. The TTRigger subsystem commands are:

- ACQuisition
- BRANch
- CLEar
- FIND
- GLEDge
- RANGe
- SEQuence
- SPERiod
- TCONtrol
- TERM
- TIMER
- TPOSition

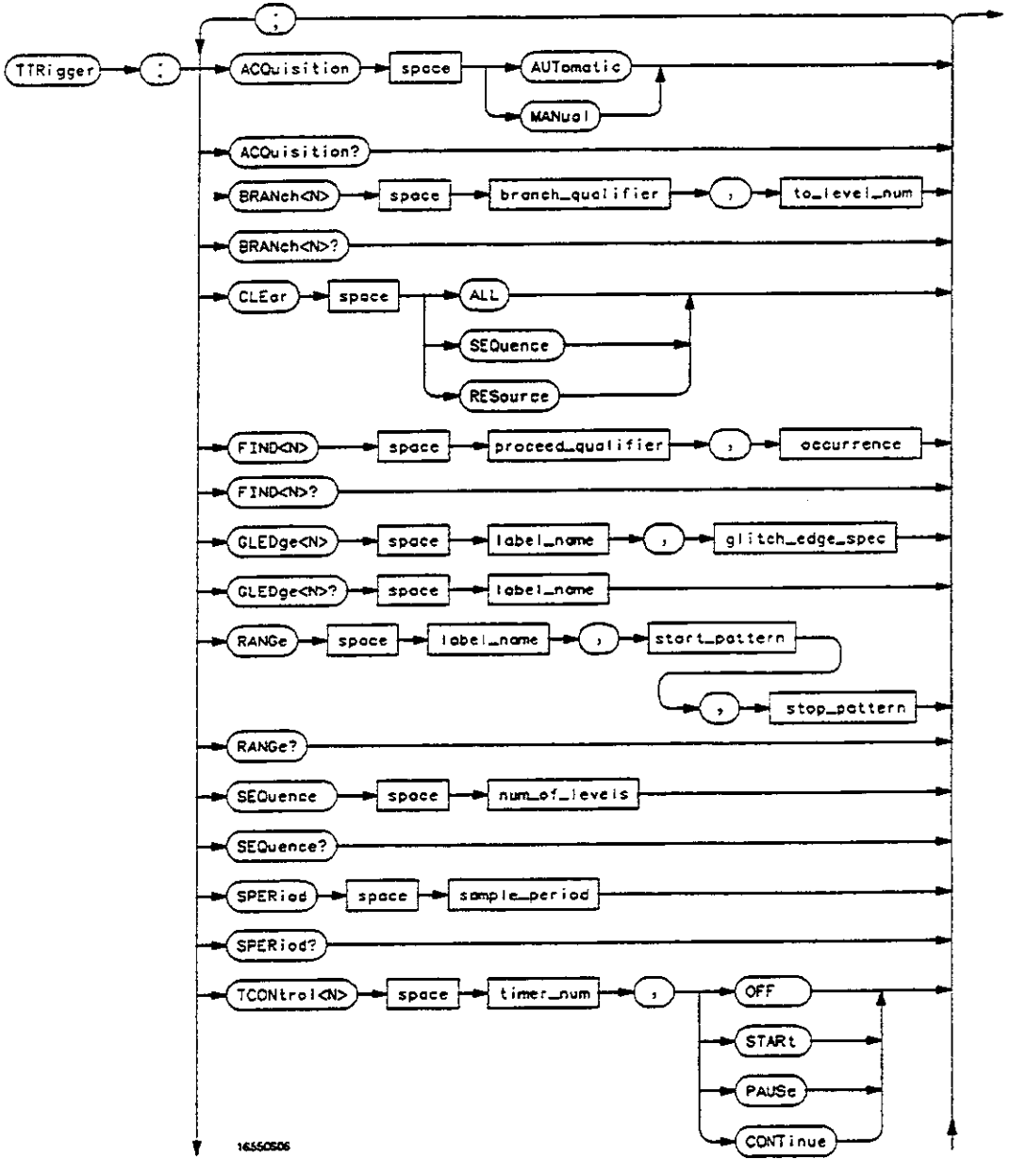


Figure 6-1. TTRigger Subsystem Syntax Diagram

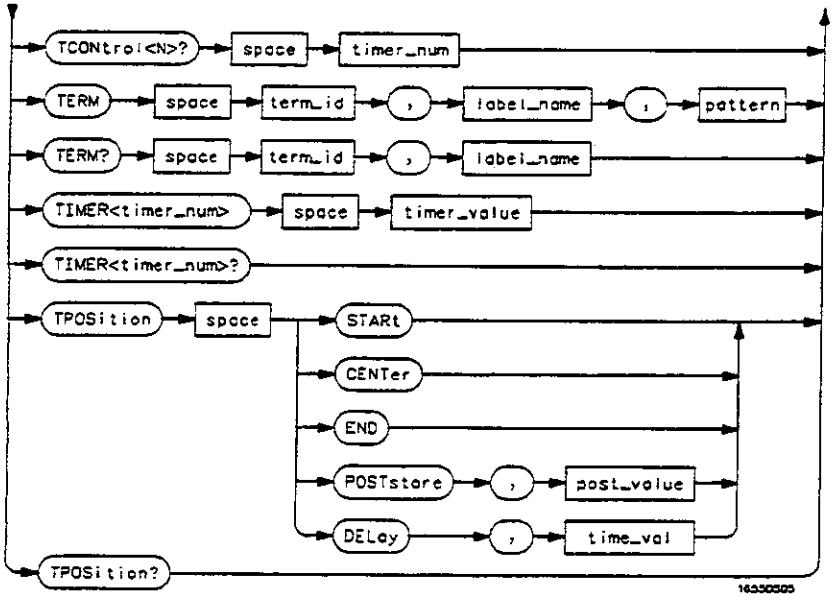


Figure 6-1. TTRigger Subsystem Syntax Diagram (continued)

branch_qualifier = *<qualifier>*
to_lev_num = *integer from 1 to last level*
proceed_qualifier = *<qualifier>*
occurrence = *number from 1 to 1048575*
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}... }"
num_of_levels = *integer from 1 to 10*
lev_of_trig = *integer from 1 to (number of existing sequence levels)*
store_qualifier = *<qualifier>*
state_tag_qualifier = *<qualifier>*
timer_num = {1|2}
timer_value = *400 ns to 500 seconds*
term_id = {A|B|C|D|E|F|G|H|I|J}
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 = *see "Qualifier" on page 5*
post_store = *integer from 0 to 100 representing percentage*
time_val = *integer from 0 to 500 representing seconds*

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

Qualifier

The qualifier for the timing trigger subsystem can be terms A through J, Timer 1 and 2, and Range 1 and 2. In addition, qualifiers can be the NOT boolean function of terms, timers, and ranges. The qualifier can also be an expression or combination of expressions as shown below and figure 12-2, "Complex Qualifier," on page 12-11.

The following parameters show how qualifiers are specified in all commands of the TTRigger subsystem that use **<qualifier>**.

where:

```
<qualifier>          ::= { "ANYSTATE" | "NOSTATE" | "<expression>" }

<expression>         ::= {<expression1a> | <expression1b> | <expression1a> OR
                         <expression1b> | <expression1a> AND <expression1b>}

<expression1a>       ::= {<expression1a_term> | (<expression1a_term> [ OR
                         <expression1a_term>]*) | (<expression1a_term> [ AND <expression1a_term>]*)}

<expression1a_term>  ::= { <expression2a> | <expression2b> | <expression2c> | <expression2d>}

<expression1b>       ::= {<expression1b_term> | ( <expression1b_term> [ OR
                         <expression1b_term>]*) | (<expression1b_term> [ AND <expression1b_term>]*)}

<expression1b_term>  ::= {<expression2e> | <expression2f> | <expression2g> | <expression2h>}

<expression2a>       ::= {<term3a> | <term3b> | (<term3a> <boolean_op> <term3b>)}
<expression2b>       ::= {<term3c> | <range3a> | (<term3c> <boolean_op> <range3a>)}
<expression2c>       ::= {<term3d> | <gledge3a | (<term3d> <boolean_op> <gledge3a>)}
<expression2d>       ::= {<term3e> | <timer3a> | (<term3e> <boolean_op> <timer3a>)}
<expression2e>       ::= {<term3f> | <term3g> | (<term3f> <boolean_op> <term3g>)}
<expression2f>       ::= {<term3h> | <range3b> | (<term3h> <boolean_op> <range3b>)}
<expression2g>       ::= {<term3i> | <gledge3b> | (<term3i> <boolean_op> <gledge3b>)}
<expression2h>       ::= {<term3j> | <timer3b> | (<term3e> <boolean_op> <timer3b>)}

<boolean_op>         ::= {AND | NAND | OR | NOR | XOR | NXOR}
```

```
<term3a>    ::= { A | NOTA }
<term3b>    ::= { B | NOTB }
<term3c>    ::= { C | NOTC }
<term3d>    ::= { D | NOTD }
<term3e>    ::= { E | NOTE }
<term3f>    ::= { F | NOTF }
<term3g>    ::= { G | NOTG }
<term3h>    ::= { H | NOTH }
<term3i>    ::= { I | NOTI }
<term3j>    ::= { J | NOTJ }

<range3a>   ::= { IN_RANGE1 | OUT_RANGE1 }
<range3b>   ::= { IN_RANGE2 | OUT_RANGE2 }

<gledge3a>  ::= {GLEDge1 | NOT GLEDge1}
<gledge3b>  ::= {GLEDge2 | NOT GLEDge2}

<timer3a>   ::= { TIMER1< | TIMER1>}
<timer3b>   ::= { TIMER2< | TIMER2>}
```

* = is optional such that it can be used zero or more times
\+ = must be used at least once and can be repeated

Qualifier Rules The following rules apply to qualifiers:

- Qualifiers are quoted strings and, therefore, need quotes.
- Expressions are evaluated from left to right.
- Parenthesis are used to change the order evaluation and, therefore, are optional.
- An expression must map into the combination logic presented in the combination pop-up menu within the TTRigger menu.

Examples:

```
'A'
'( A OR B )'
'(( A OR B ) AND C )'
'(( A OR B ) AND C AND IN_RANGE2 )'
'(( A OR B ) AND ( C AND IN_RANGE1 ))'
'IN_RANGE1 AND ( A OR B ) AND C'
```

TTRigger (TTRace) selector

The TTRigger (TTRace) (Trace Trigger) selector is used as a 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 command tree.

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

Example:

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

ACQuisition command/query

The ACQuisition command allows you to specify the acquisition mode for the Timing analyzer.

The ACQuisition query returns the current acquisition mode specified.

Command Syntax: `:MACHine{1|2}:TTRigger:ACQuisition {AUTOmatic|MANual}`

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:ACQUISITION AUTOMATIC"
```

Query Syntax: `:MACHine{1|2}:TTRigger:ACQuisition?`

Returned Format: `[:MACHine{1|2}:TTRigger:ACQuisition] {AUTOmatic|MANual}<NL>`

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:ACQUISITION?"
```

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.

The terms used by the branch qualifier (A through J) are defined by the TERM command. The meaning of IN_RANGE and OUT_RANGE 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 Timing Trigger menu. 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 statements are all correct and have the same meaning. Notice that the conventional rules for precedence are not followed. The expressions are evaluated from left to right.

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'C AND D OR F OR G', 1"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 '((C AND D)  OR (F OR G))', 1"
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 'F OR (C AND D) OR G',1"
```

Figure 12-2, on page 12-11 shows a complex expression as seen in the Timing Trigger menu.

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

BRANch

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

where:

<N>	::= integer from 1 to <number_of_levels>
<to_level_number>	::= integer from 1 to <number_of_levels>
<number_of_levels>	::= integer from 1 to the number of existing sequence levels (maximum 10)
<branch_qualifier>	::= <qualifier> see "Qualifier" on page 12-5

Examples:

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

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

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

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH3?"
```

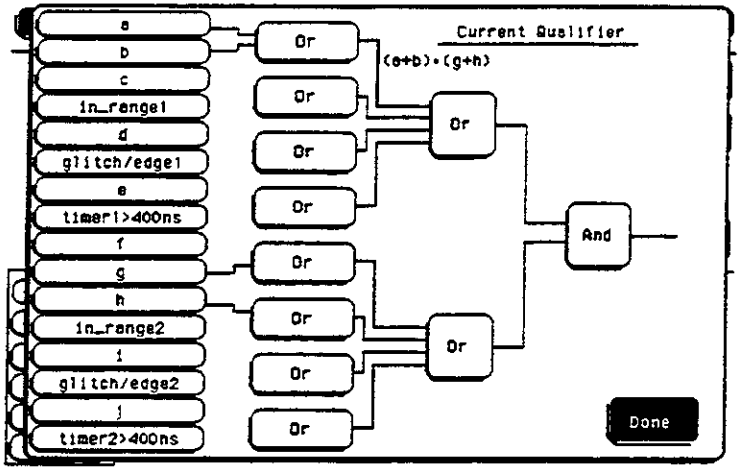


Figure 12-2. Complex Qualifier

Figure 12-2 is a front-panel representation of the complex qualifier (a Or b) And (g Or h). The following example would be used to specify this complex qualifier.

```
OUTPUT XXX;":MACHINE1:TTRIGGER:BRANCH1 '((A OR B) AND (G OR H))', 2"
```

Note

Terms A through E, RANGE 1, GLITCH/EDGE1, and TIMER 1 must be grouped together and terms F through J, RANGE 2, GLITCH/EDGE2, and TIMER 2 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 IN_RANGE2) AND (C OR H)) is not allowed because the term C cannot be specified in the E through J group.

In the first level, the operators you can use are AND, NAND, OR, NOR, XOR, NXOR. 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.

CLEar command

The CLEar command allows you to clear all settings in the Timing Trigger menu and replace them with the default, clear only the sequence levels, or clear only the resource term patterns.

Command Syntax: `:MACHine{1|2}:TTRigger:CLEar {All|SEQuence|RESource}`

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:CLEAR RESOURCE"
```

FIND

command/query

The FIND command defines the time qualifier for a given sequence level. The qualifier tells the timing 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. In the sequence level where the trigger is specified, the FIND command specifies the trigger qualifier (see SEQuence command).

The terms A through J are defined by the TERM command. The meaning of IN_RANGE and OUT_RANGE is determined by the RANGe command. Expressions are limited to what you could manually enter through the Timing Trigger 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 12-2 on page 12-11 for a detailed example.

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

Command Syntax: `:MACHine{1|2}:TTRigger:FIND<N> <time_qualifier>,<condition_mode>`

where:

<N>	::= integer from 1 to the number of existing sequence levels (maximum 10)
<condition_mode>	::= {{GT|LT}, <duration_time> |OCCurrence, <occurrence>}
GT	::= greater than
LT	::= less than
<duration_time>	::= real number from 8 ns to 5.00 seconds depending on sample period
<occurrence>	::= integer from 1 to 1048575
<time_qualifier>	::= <qualifier> see "Qualifier" on page 12-5

Examples:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:FIND1 'ANYSTATE', GT, 10E-6"
OUTPUT XXX;":MACHINE1:TTRIGGER:FIND3 '((NOTA AND NOTB) OR G)',
OCCURRENCE, 10"
```

FIND

Query Syntax: `:MACHine{1|2}:TTRigger:FIND4?`

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

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:FIND<N>?"
```

GLEDge

command/query

The GLEDge (glitch/edge) command allows you to define edge and glitch specifications for a given label. Edge specifications can be R (rising), F (falling), E (either), or "." (don't care). Glitch specifications consist of G (glitch) or "." (don't care). Edges and glitches are sent in the same string with the right most string character specifying what the right most bit will be.

Note The <glitch_edge_spec> string length must match the exact number of bits assigned to the specified label. If the string length does not match the number of bits, the "Parameter string invalid" message is displayed.

The GLEDge query returns the current specification for the given label.

Command Syntax:

```
:MACHine{1|2}:TTRigger:GLEDge<N> <label_name>, <glitch_edge_spec>
```

where:

<N>	::= {1|2}
<label_name>	::= string of up to 6 alphanumeric characters
<glitch_edge_spec>	::= string consisting of {R|F|E|G|.|[to total number of bits]}

Example: For 8 bits assigned and no glitch:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:GLEDGE1 'DATA', '....F..E'"
```

For 16 bits assigned with glitch:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:GLEDGE1 'DATA', '....GGG.....F..R'"
```

Query Syntax:

```
:MACHine{1|2}:TTRigger:GLEDe<N>? <label_name>
```

Returned Format:

```
[:MACHine{1|2}:TTRigger:GLEDe<N>] <label_name>,<glitch_edge_pattern><NL>
```

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:GLEDGE1? 'DATA'"
```

RANGe

command/query

The RANGe command allows you to specify a range recognizer term for 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.

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

Command Syntax:

```
:MACHine{1|2}:TTRigger: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:TTRIGGER:RANGE 'DATA', '127', '255' "
OUTPUT XXX;":MACHINE1:TTRIGGER:RANGE 'ABC', '#B00001111', '#HCF' "
```

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

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

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:RANGE?"
```

SEQuence command/query

The SEQuence command defines the timing analyzer trace sequence. First, it deletes the current trace sequence. Then, it inserts the number of levels specified, with default settings. The number of levels can be between 1 and 10 when the analyzer is armed by the RUN key.

The SEQuence query returns the current sequence specification.

Command Syntax: `:MACHine{1|2}:TTRigger:SEQuence <number_of_levels>`

where:

<number_of_levels> ::= integer from 1 to 10

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:SEQUENCE 4"
```

Query Syntax: `:MACHine{1|2}:TTRigger:SEQuence?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:SEQUENCE?"
```

SPERiod

command/query

The SPERiod command allows you to set the sample period of the timing analyzer in the Conventional and Glitch modes. The sample period range depends on the mode selected and is as follows:

2 ns to 8 ms for Conventional Half Channel 500 MHz
4 ns to 8 ms for Conventional Full Channel 250 MHz
4 ns for Transitional Half Channel
8 ns for Transitional Full Channel
8 ns to 8 ms for Glitch Half Channel 125 MHz

The SPERiod query returns the current sample period.

Command Syntax: `:MACHine{1|2}:TTRigger:SPERiod <sample_period>`

where:

<sample_period> ::= real number from 2 ns to 8 ms depending on mode

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:SPERIOD 50E-9"
```

Query Syntax: `:MACHine{1|2}:TTRigger:SPERiod?`

Returned Format: `[:MACHine{1|2}:TTRigger:SPERiod] <sample_period><NL>`

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:SPERIOD?"
```

TCONtrol command/query

The TCONtrol (timer control) command allows you to turn off, start, pause, or continue the timer for the specified level. The time value of the timer is defined by the TIMER command.

The TCONtrol query returns the current TCONtrol setting of the specified level.

Command Syntax:
```
:MACHine{1|2}:TTRigger:TCONtrol<N> <timer_num>,
{OFF|STARt|PAUSe|CONTinue}
```

where:

<N> ::= integer from 1 to the number of existing sequence levels (maximum 10)
<timer_num> ::= {1|2}

Example:
```
OUTPUT XXX;":MACHINE2:TTRIGGER:TCONTROL6 1, PAUSE"
```

Query Syntax:
```
:MACHine{1|2}:TTRigger:TCONTROL<N>? <timer_num>
```

Returned Format:
```
[:MACHine{1|2}:TTRigger:TCONTROL<N> <timer_num>]
{OFF|STARt|PAUSe|CONTinue}<NL>
```

Example:
```
OUTPUT XXX;":MACHINE2:TTRIGGER:TCONTROL6? 1"
```

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.

All 10 terms (A through J) are available to either machine but not both simultaneously. If you send the TERM command to a machine with a term that has not been assigned to that machine, an error message "Legal command but settings conflict" is returned.

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

Command Syntax:

```
:MACHine{1|2}:TTRigger:TERM <term_id>,<label_name>,<pattern>
```

where:

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

Query Syntax:

```
:MACHine{1|2}:TTRigger:TERM? <term_id>,<label_name>
```

Returned Format:

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

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TERM? B,'DATA' "
```

TIMER command/query

The TIMER command sets the time value for the specified timer. The limits of the timer are 400 ns to 500 seconds in 16 ns to 500 μs increments. The increment value varies with the time value of the specified timer.

The TIMER query returns the current time value for the specified timer.

Command Syntax: `:MACHine{1|2}:TTRigger:TIMER{1|2} <time_value>`

where:

<time_value> ::= real number from 400 ns to 500 seconds in increments which vary from 16 ns to 500 μs.

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:TIMER1 100E-6"
```

Query Syntax: `:MACHine{1|2}:TTRigger:TIMER{1|2}?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TTRIGGER:TIMER1?"
```

TPOSition

command/query

The TPOSition (trigger position) command allows you to set the trigger at the start, center, end or at any position in the trace (poststore). Poststore is defined as 0 to 100 percent with a poststore of 100 percent being the same as start position and a poststore 0 percent being the same as an end trace.

The TPOSition query returns the current trigger position setting.

Command Syntax: `:MACHine{1|2}:TTRigger:TPOSition {STARt|CENTer|END|DELay, <time_val>|POSTstore,<poststore>}`

where:

<time_val> ::= real number from either (2 × sample period) or 16 ns whichever is greater to (1048575 × sample period).

<poststore> ::= integer from 0 to 100 representing percentage of poststore.

Examples:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TPOSITION END"
OUTPUT XXX;":MACHINE1:TTRIGGER:TPOSITION POSTstore,75"
```

Query Syntax: `:MACHine{1|2}:TTRigger:TPOSition?`

Returned Format: `[:MACHine{1|2}:TTRigger:TPOSition] {STARt|CENTer|END|DELay, <time_val>|POSTstore,<poststore>}<NL>`

Example:

```
OUTPUT XXX;":MACHINE1:TTRIGGER:TPOSITION?"
```

TWAVeform Subsystem 13

Introduction

The TWAVeform subsystem contains the commands available for the Timing Waveforms menu in the HP 16550A. These commands are:

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

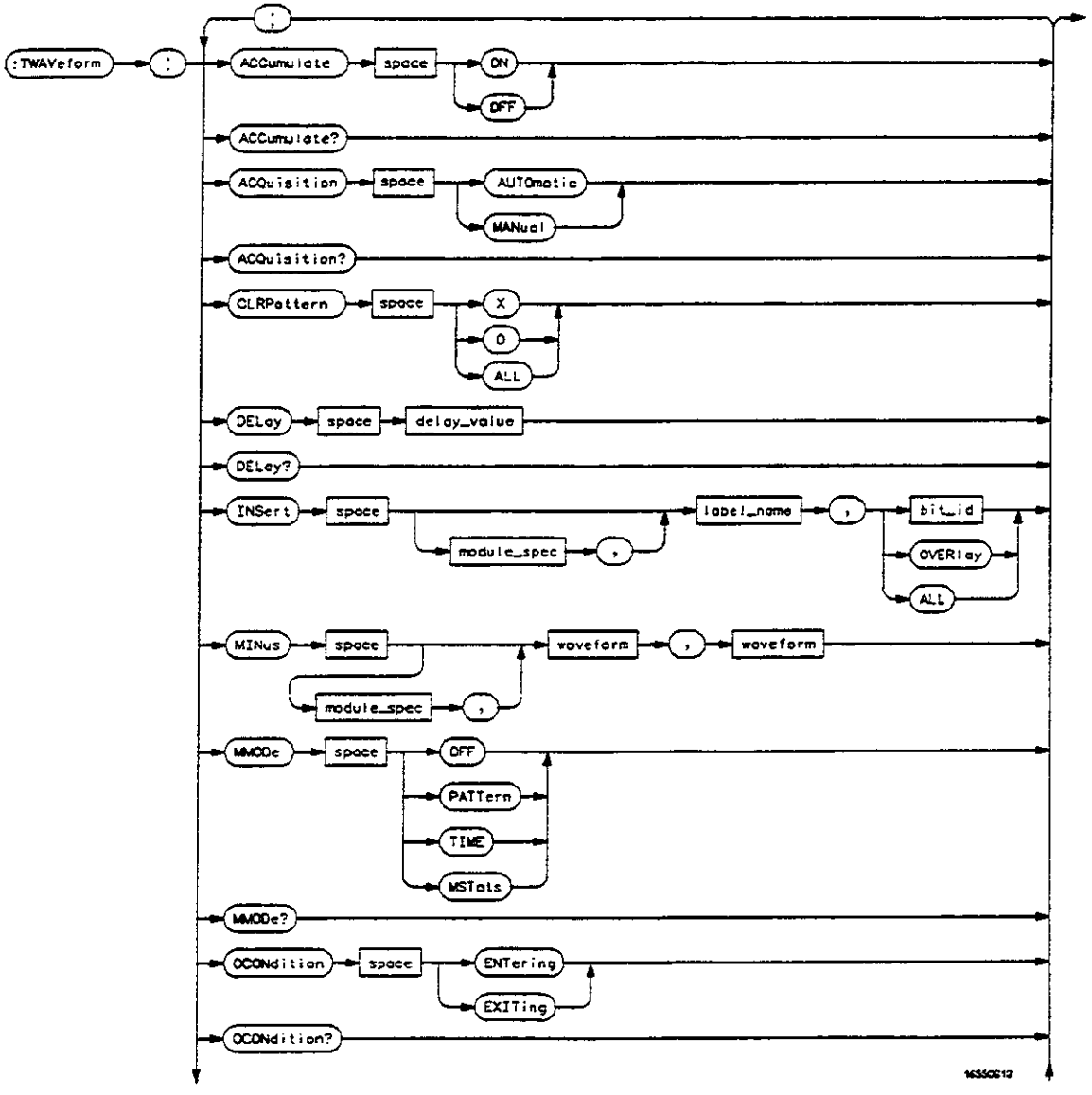


Figure 13-1. TWAVeform Subsystem Syntax Diagram

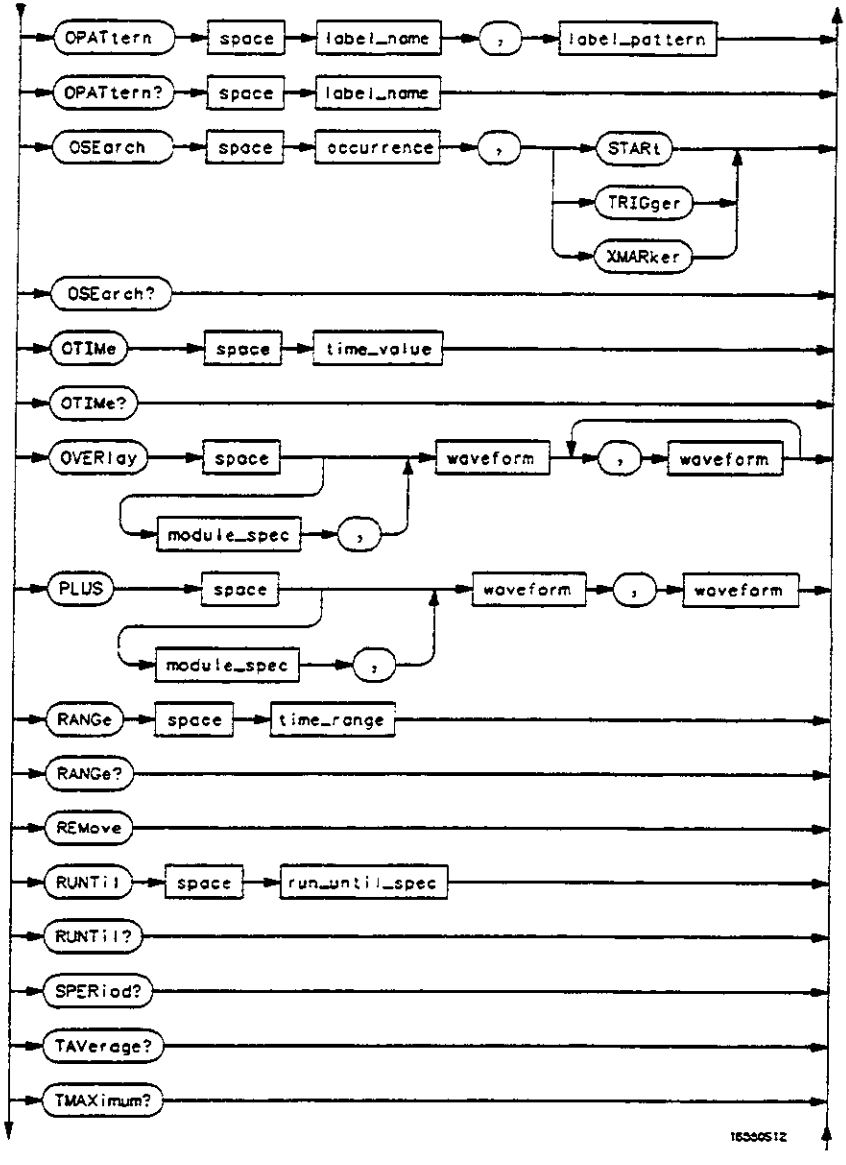


Figure 13-1. TWAVeform Subsystem Syntax Diagram (continued)

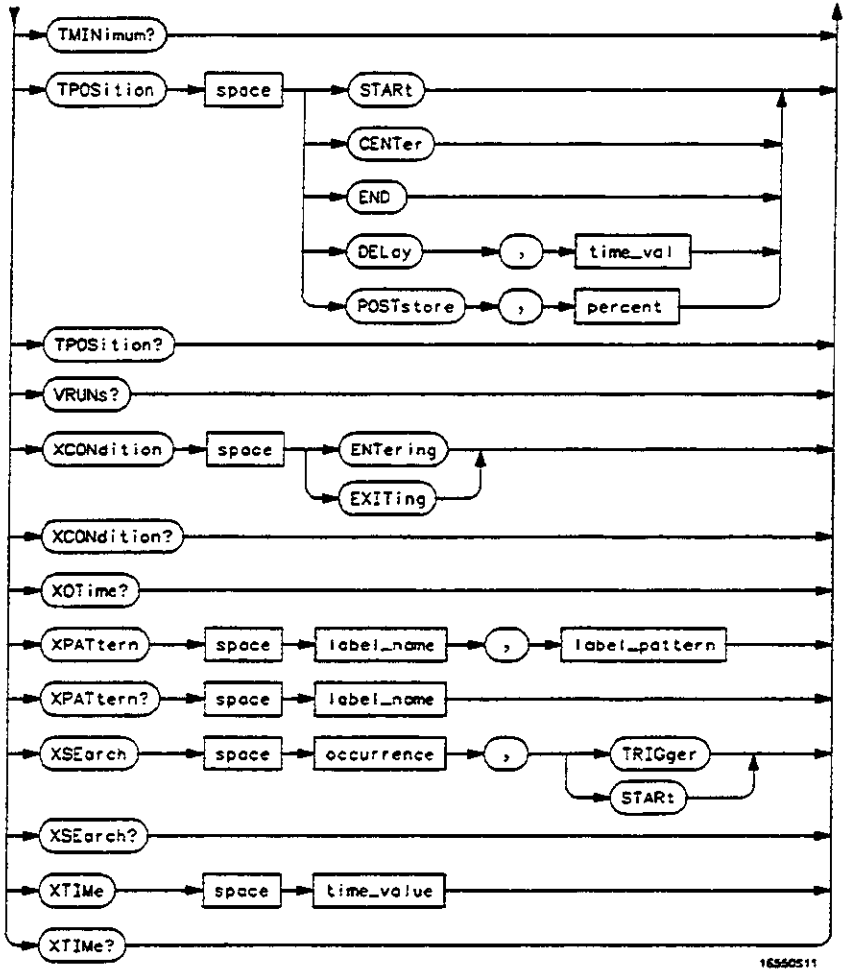


Figure 13-1. TWAVeform Subsystem Syntax Diagram (continued)

delay_value = *real number between* −2500 s and +2500 s
module_spec = *{1|2|3|4|5|6|7|8|9|10|11|12}*
bit_id = *integer from 0 to 31*
waveform = *string containing <acquisition_spec>{1|2}*
acquisition_spec = *{A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)*
label_name = *string of up to 6 alphanumeric characters*
label_pattern = "*{#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 time base card is installed*
time_range = *real number between 10 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*
time_val = *real number from 0 to 500 representing seconds*

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 characters, "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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:ACCUMULATE?"
```

ACQuisition command/query

The ACQuisition command allows you to specify the acquisition mode for the state analyzer. The acquisition modes are automatic and manual.

The ACQuisition query returns the current acquisition mode.

Command Syntax: `:MACHine{1|2}:TWAVeform:ACQuisition {AUTOmatic|MANual}`

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

Returned Format: `[MACHine{1|2}:TWAVeform:ACQuisition] {AUTOmatic|MANual}<NL>`

Example:
```
OUTPUT XXX;":MACHINE2:TWAVEFORM:ACQUISITION?"
```

CENTer command

The CENTer command allows you to center the waveform display about the specified markers. The markers are placed on the waveform in the SLISt subsystem.

Command Syntax: `:MACHine{1|2}:Twaveform:CENTer <marker_type>`

where:

<marker_type> ::= {X|O|XO|TRIGger}

Example:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:CENTER X"
```

CLRPattern command

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Command Syntax: `:MACHine{1|2}:TWAVeform:CLRPattern {X|0|ALL}`

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:CLRPATTERN ALL"
```

CLRStat command

The CLRStat command allows you to clear the waveform statistics without having to stop and restart the acquisition.

Command Syntax: `:MACHine{1|2}:Twaveform:CLRStat`

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:CLRSTAT"
```

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. If the acquisition mode is automatic, then 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 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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:DELAY?"
```

INSert command

The INSert command inserts waveforms in the timing waveform display. The waveforms are added from top to bottom up to a maximum of 96 waveforms. Once 96 waveforms are present, each time you insert another waveform, it 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, which is the module specifier. 1 through 10 corresponds to modules A through J. If you do not specify the module, 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 you specify OVERlay, all the bits of the label are displayed as a composite overlaid waveform. If you specify ALL, all the bits are displayed sequentially. If you do not specify the third parameter, 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|6|7|8|9|10|11|12}
<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|6|7|8|9|10|11|12} 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, where 1 through 10 refers to slots A through J. 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|6|7|8|9|10}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)

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

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

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 (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'"
```

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

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

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OPATTERN? 'A'"
```

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> ::= {STARt|TRIGger|XMARker}
<occurrence> ::= integer from −8192 to +8192

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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OSEARCH?"
```

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 −2.5 ks to +2.5 ks

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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:OTIME?"
```

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|6|7|8|9|10}
<waveform> ::= string containing <acquisition_spec> {1|2}
<acquisition_spec> ::= {A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)

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, where 1 through 10 refers to slots A through J. 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|6|7|8|9|10}
<waveform> ::= string containing <acquisition_spec>{1|2}
<acquisition_spec> ::= {A|B|C|D|E|F|G|H|I|J} (slot where acquisition card is located)

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 10 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 10 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: `OUTPUT XXX;":MACHINE1:TWAVEFORM:RANGE?"`

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 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 2 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"
```

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

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

Example:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:RUNTIL?"
```

SPERiod command/query

The SPERiod command allows you to set the sample period of the timing analyzer in the Conventional and Glitch modes. The sample period range depends on the mode selected and is as follows:

2 ns to 8 ms for Conventional Half Channel 500 MHz
4 ns to 8 ms for Conventional Full Channel 250 MHz
8 ns to 8 ms for Glitch Half Channel 125 MHz

The SPERiod query returns the current sample period.

Command Syntax: `:MACHine{1|2}:TWAVeform:SPERiod <sample_period>`

where:

<sample_period> ::= real number from 2 ns to 8 ms depending on mode

Example:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:SPERIOD 50E-9"
```

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

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

Example:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:SPERIOD?"
```

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:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:TAVERAGE?"
```

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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:TMAXIMUM?"
```

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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:TMINIMUM?"
```

TPOSition command/query

The TPOSition command allows you to control where the trigger point is placed. The trigger point can be placed at the start, center, end, at a percentage of post store, or at a value specified by delay. The post store option is the same as the User Defined option when setting the trigger point from the front panel.

The TPOSition command is only available when the acquisition mode is set to manual.

The TPOSition query returns the current trigger setting.

Command Syntax: `MACHine{1|2}:TWAVeform:TPOSition {STARt|CENTer|END|DELay, <time_val>|POSTstore,<percent>}`

where:

<time_val> ::= real number from 0 to 500 seconds
<percent> ::= integer from 1 to 100

Example:
```
OUTPUT XXX;":MACHINE2:TWAVEFORM:TPOSITION CENTER"
```

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

Returned Format: `[MACHine{1|2}:TWAVeform:TPOSition] {STARt|CENTer|END|DELay, <time_val>|POSTstore,<percent>}<NL>`

Example:
```
OUTPUT XXX;":MACHINE2:TWAVEFORM:TPOSition?"
```

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:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:VRUNS?"
```

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

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:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XOTIME?"
```

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 iterations.

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

The XPATtern query, in pattern marker mode, returns the pattern specification for a given label name. In the time marker mode, the query returns the pattern under the X marker for a given label. If the X marker is not placed on valid data, don't cares (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'"
```

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

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

Example:

```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XPATTERN? 'A*'"
```

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|STARt}
<occurrence> ::= integer from −8192 to +8192

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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XSEARCH?"
```

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 −2.5 ks to +2.5 ks

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:
```
OUTPUT XXX;":MACHINE1:TWAVEFORM:XTIME?"
```

TLISt Subsystem 14

Introduction

The TLISt subsystem contains the commands available for the Timing Listing menu in the HP 16550A logic analyzer module and is the same as the SLISt subsystem with the exception of the OCONdition and XCONdition commands. The TLISt subsystem commands are:

- COLumn
- CLRPattern
- DATA
- LINE
- MMODe
- OCONdition
- OPATtern
- OSEarch
- OSTate
- OTAG
- REMove
- RUNTil
- TAVerage
- TMAXimum
- TMINimum
- VRUNs
- XCONdition
- XOTag
- XOTime
- XPATtern
- XSEarch
- XSTate
- XTAG

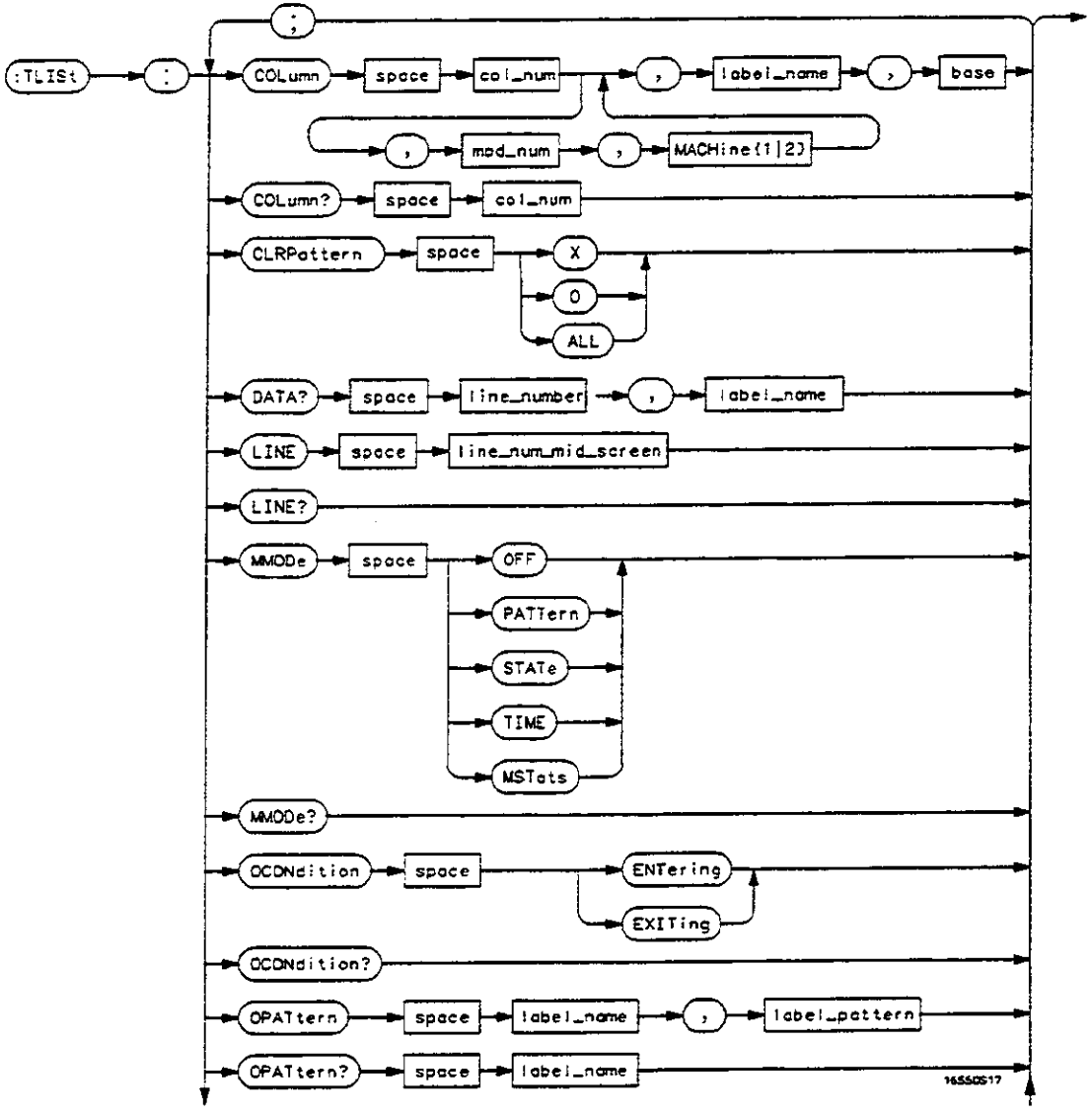


Figure 14-1. TLISt Subsystem Syntax Diagram

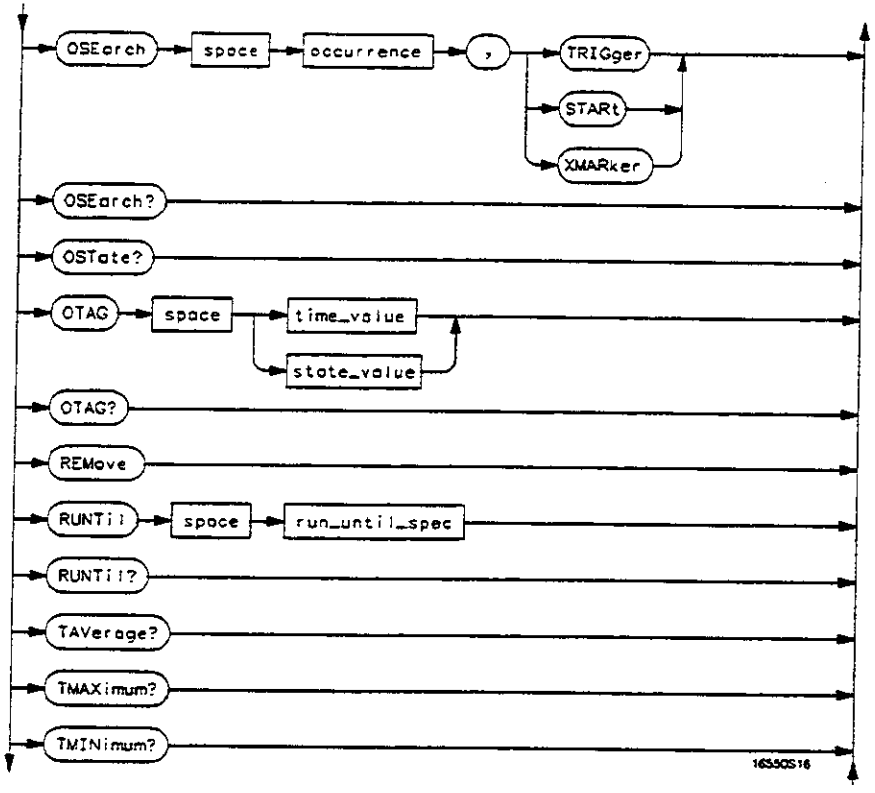
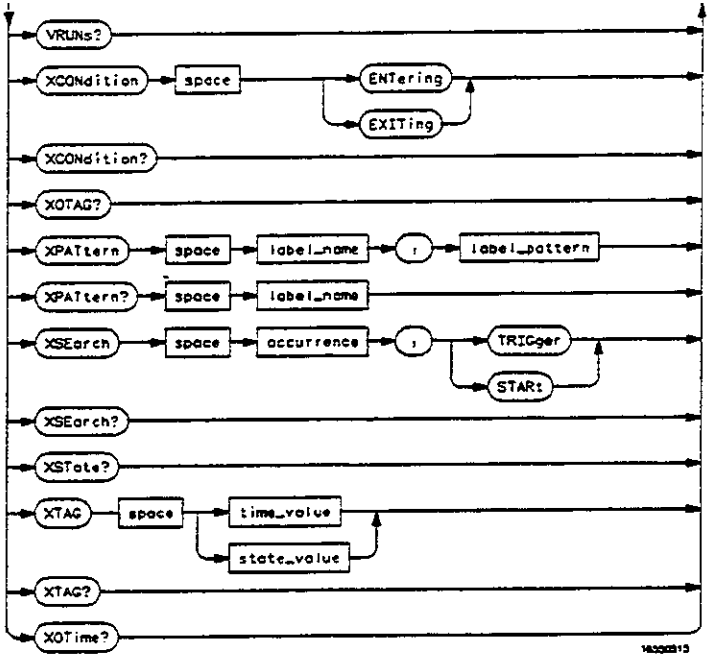


Figure 14-1. TLISt Subsystem Syntax Diagram (continued)



module_num = {*1*|*2*|*3*|*4*|*5*|*6*|*7*|*8*|*9*|*10*}
mach_num = {*1*|*2*}
col_num = *integer from 1 to 61*
line_number = *integer from* −8191 to +8191
label_name = *a string of up to 6 alphanumeric characters*
base = {*BINary*|*HEXadecimal*|*OCTal*|*DECimal*|*TWOS*|*ASCii*|*SYMBol*|*LASSembler*} *for labels or*
{*ABSolute*|*RELative*} *for tags*
line_num_mid_screen = *integer from* −8191to +8191
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}... }"
currence = *integer from* −8191 to +8191
e_value = *real number*
e_value = *real number*
until_spec = {*OFF*|*LT*, <*value*>|*GT*, <*value*>|*INRange*, <*value*>, <*value*>|
OUTRange, <*value*>, <*value*>}
= *real number*

Figure 14-1. TLISt Subsystem Syntax Diagram (continued)

TLISt

selector

The TLISt selector is used as part of a compound header to access those settings normally found in the Timing 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}:TLISt`

Example:

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

COLumn command/query

The COLumn command allows you to configure the timing analyzer list display by assigning a label name and base to one of the 61 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.

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

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.

Command Syntax:

```
:MACHine{1|2}:TLISt:COLumn <col_num>[,<module_num>,MACHine{1|2}],
<label_name>,<base>
```

where:

<col_num> ::= integer from 1 to 61

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

<base> ::= {BINary|HEXadecimal|OCTal|DECimal|TWOS|ASCii|SYMBol|IASSembler} for labels

or

::= {ABSolute|RELative} for tags

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:COLUMN 4,2,'A',HEX"
```

Query Syntax:

```
:MACHine{1|2}:TLISt:COLumn? <col_num>
```

Returned Format:

```
[:MACHine{1|2}:TLISt:COLumn] <col_num>,<module_num>,MACHine{1|2},
<label_name>,<base><NL>
```

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:COLUMN? 4"
```

CLRPattern command

The CLRPattern command allows you to clear the patterns in the selected Specify Patterns menu.

Command Syntax: `:MACHine{1|2}:TLISt:CLRPattern {X|0|ALL}`

Example: `OUTPUT XXX;":MACHINE1:TLIST:CLRPATTERN 0"`

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}:TLISt:DATA? <line_number>,<label_name>
```

Returned Format:

```
[:MACHine{1|2}:TLISt:DATA] <line_number>,<label_name>,
<pattern_string><NL>
```

where:

<line_number> ::= integer from −8191 to +8191

<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:

```
OUTPUT XXX;":MACHINE1:TLIST:DATA? 512, 'RAS'"
```

LINE

command/query

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

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

Command Syntax: `:MACHine{1|2}:TLISt:LINE <line_num_mid_screen>`

where:

<line_num_mid_screen> ::= integer from −8191 to +8191

Example: `OUTPUT XXX;":MACHINE1:TLIST:LINE 0"`

Query Syntax: `:MACHine{1|2}:TLISt:LINE?`

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

Example: `OUTPUT XXX;":MACHINE1:TLIST:LINE?"`

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}:TLISt:MMODe <marker_mode>`

where:

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:MMODE TIME"
```

Query Syntax: `:MACHine{1|2}:TLISt:MMODe?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:MMODE?"
```

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}:TLISt:OCONdition {ENTering|EXITing}`

Example:
```
OUTPUT XXX; ":MACHINE1:TLIST:OCONDITION ENTERING"
```

Query Syntax: `:MACHine{1|2}:TLISt:OCONdition?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:OCONDITION?"
```

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 iterations.

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

The OPATtern query returns the pattern specification for a given label name.

Command Syntax:

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

Query Syntax:

```
:MACHine{1|2}:TLISt:OPATtern? <label_name>
```

Returned Format:

```
[:MACHine{1|2}:TLISt:OPATtern] <label_name>,<label_pattern><NL>
```

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:OPATTERN? 'A'"
```

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 OSEarch 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}:TLISt:OSEarch <occurrence>,<origin>`

where:

`<occurrence>` ::= integer from −8191 to +8191

`<origin>` ::= {TRIGger|STARt|XMARker}

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

Query Syntax: `:MACHine{1|2}:TLISt:OSEarch?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:OSEARCH?"
```

OSTate query

The OSTate query returns the line number in the listing where the O marker resides (−8191 to +8191). If data is not valid, the query returns 32767.

Query Syntax: `:MACHine{1|2}:TLISt:OSTate?`

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

where:

<state_num> ::= an integer from −8191 to +8191, or 32767

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:OSTATE?"
```

OTAG

command/query

The OTAG command specifies the tag value on which the O Marker should be placed. The tag value is time. If the data is not valid tagged data, no action is performed.

The OTAG query returns the O Marker position in time 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, or returns 32767 for state tagging.

Command Syntax: `:MACHine{1|2}:TLISt:OTAG <time_value>`

where:

<time_value> ::= real number

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

Query Syntax: `:MACHine{1|2}:TLISt:OTAG?`

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

Example: `OUTPUT XXX;":MACHINE1:TLIST:OTAG?"`

REMove command

The REMove command removes all labels, except the leftmost label, from the listing menu.

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

Example:

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

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, until 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 8 ns apart since this is the minimum time resolution of the time tag counter.

The RUNTil query returns the current stop criteria.

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

where:

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

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:RUNTIL GT,800.0E-6"
```

Query Syntax: `:MACHine{1|2}:TLISt:RUNTil?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:RUNTIL?"
```

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}:TLISt:TAVerage?`

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

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:TAVERAGE?"
```

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}:TLISt:TMAXimum?`

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

where:

<time_value> ::= real number

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:TMAXIMUM?"
```

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}:TLISt:TMINimum?`

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

where:

<time_value> ::= real number

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:TMINIMUM?"
```

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}:TLISt:VRUNs?`

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

where:

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

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:VRUNS?"
```

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}:TLISt:XCONdition {ENTering|EXITing}`

Example:
```
OUTPUT XXX; ":MACHINE1:TLIST:XCONDITION ENTERING"
```

Query Syntax: `:MACHine{1|2}:TLISt:XCONdition?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:XCONDITION?"
```

XOTag

query

The XOTag query returns the time from the X to O markers. If there is no data in the time mode the query returns 9.9E37.

Query Syntax: `:MACHine{1|2}:TLISt:XOTag?`

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

where:

<XO_time> ::= real number

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:XOTAG?"
```

XOTime query

The XOTime query returns the time from the X to O markers. If there is no data in the time mode the query returns 9.9E37.

Query Syntax: `:MACHine{1|2}:TLISt:XOTime?`

Returned Format: `[:MACHine{1|2}:TLISt:XOTime] <XO_time><NL>`

where:

<XO_time> ::= real number

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:XOTIME?"
```

XPATtern

command/query

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

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

The XPATtern query returns the pattern specification for a given label name.

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

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

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

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:XPATTERN? 'A'"
```

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}:TLISt:XSEarch <occurrence>,<origin>`

where:

<occurrence> ::= integer from −8191 to +8191
<origin> ::= {TRIGger|STARt}

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

Query Syntax: `:MACHine{1|2}:TLISt:XSEarch?`

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

Example:
```
OUTPUT XXX;":MACHINE1:TLIST:XSEARCH?"
```

XSTate

query

The XSTate query returns the line number in the listing where the X marker resides (−8191 to +8191). If data is not valid, the query returns 32767.

Query Syntax: `:MACHine{1|2}:TLISt:XSTate?`

Returned Format: `[:MACHine{1|2}:TLISt:XSTate] <state_num><NL>`

where:

<state_num> ::= an integer from −8191 to +8191, or 32767

Example:

```
OUTPUT XXX;":MACHINE1:TLIST:XSTATE?"
```

XTAG command/query

The XTAG command specifies the tag value on which the X Marker should be placed. The tag value is time. If the data is not valid tagged data, no action is performed.

The XTAG query returns the X Marker position in time 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.

Command Syntax: `:MACHine{1|2}:TLISt:XTAG <time_value>`

where:

<time_value> ::= real number

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

Query Syntax: `:MACHine{1|2}:TLISt:XTAG?`

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

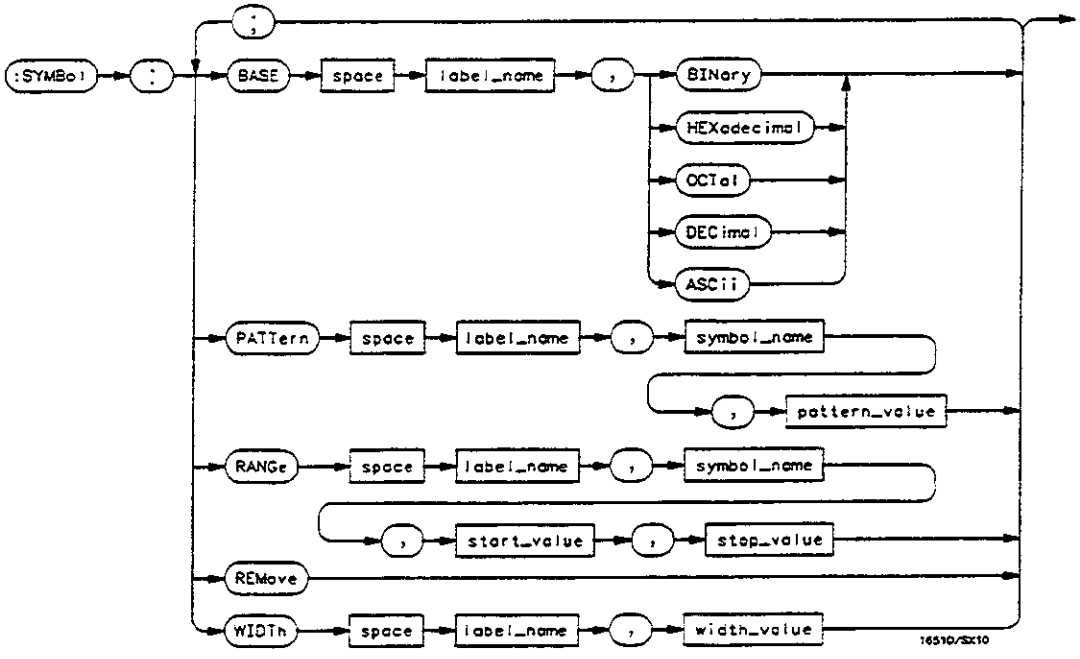
Example: `OUTPUT XXX;":MACHINE1:TLIST:XTAG?"`

SYMBol Subsystem 15

Introduction

The SYMBol subsystem contains the commands that allow you to define symbols on the controller and download them to the HP 16550A 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 15-1. 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 can 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 16

Introduction

The DATA and SETup commands are SYSTem commands that allow you to send and receive block data between the HP 16550A and a controller. Use the DATA instruction to transfer acquired timing and state 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 chapter explains how to use these commands.

The format and length of block data depends on the instruction being used, the configuration of the instrument, and the amount of acquired data. The length of the data block can be up to 606,384 bytes in a two-card configuration.

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.

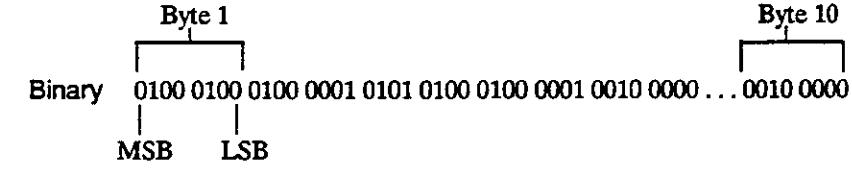
Note  Do not change the block data in the controller if you intend to send the block data back into the logic analyzer for later processing. Changes made to the block data in the controller could have unpredictable results when sent back to the logic analyzer.

Data Format

To understand the format of the data within the block data, there are four important things to keep in mind.

- Data is sent to the controller in binary form.
- Each byte, as described in this chapter, contains 8 bits.
- The first bit of each byte is the MSB (most significant bit).
- Byte descriptions are printed in binary, decimal, or ASCII depending on how the data is described.

For example, the first ten bytes that describe the section name contain a total of 80 bits as follows:



Decimal 68 65 84 65 32 32 32 32 32 32

ASCII DATA space space space space space space

SYSTem:DATA

command/query

The SYSTem:DATA command transmits the acquisition memory data from the controller to the HP 16550A logic analyzer.

The SYSTem:DATA query returns the block data to the controller.

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 a variable number of bytes containing information captured by the acquisition chips. The information will be in one of three formats, depending on the type of data captured. The three formats are glitch, transitional, conventional timing or state. 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 16550A.

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 has a variable number of bytes depending on configuration and amount of acquired data.

SYSTem:DATA

Command Syntax: :SYSTem:DATA <block data>

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

where:

<block data>	::= <block length specifier> <section> ...
<block length specifier	::= #8<length>
<length>	::= the total length of all sections in byte format (must be represented with 8 digits)
<section>	::= <section header> <section data>
<section header>	::= 16 bytes, described on the following page
<section data>	::= format depends on the 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: See "Transferring the logic analyzer acquired data" on page 17-18 in Chapter 17, "Program Examples" for an example.

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 space space space space space space" in ASCII for the DATA instruction).

11 **1 byte** - Reserved

12 **1 byte** - Module ID (0010 0000 binary or 32 decimal for the HP 16550A)

13 **4 bytes** - Length of block in number of bytes that when converted to decimal, specifies the number of bytes contained in the data block.

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 a variable number of bytes of data. 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.

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

17 **2 bytes** - Instrument ID (always 16500 decimal for HP 16550A)

19 **1 byte** - Revision Code

20 **1 byte** - number of acquisition chips used in last acquisition

Note

The values stored in the preamble represent the captured data currently stored in this structure and not the current analyzer configuration. For example, the mode of the data (bytes 21 and 49) may be STATE with tagging, while the current setup of the analyzer is TIMING.

The next 40 bytes are for Analyzer 1 Data Information.

21 **1 byte** - Machine data mode, one of the following decimal values:
- −1 = off
- 0 = state data without tags
- 1 = state data with each chip assigned to a machine (2kB memory) and either time or state tags
- 2 = state data with unassigned pod used to store tag data (4kB memory)
- 8 = state data at half channel (8kB memory with no tags)
- 10 = conventional timing data at full channel
- 11 = transitional timing data at full channel
- 12 = glitch timing data
- 13 = conventional timing data at half channel
- 14 = transitional timing data at half channel

22 **1 byte** - Unused.

23 **2 bytes** - List of pods in this analyzer, where a binary 1 indicates that the corresponding pod is assigned to this analyzer

bit 15	bit 14	bit 13	bit 12	bit 11	bit 10	bit 9	bit 8
unused	unused	always 1	Pod 12	Pod 11	Pod 10	Pod 9	Pod 8
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
Pod 7	Pod 6	Pod 5	Pod 4	Pod 3	Pod 2	Pod 1	unused

For example, xx10 0000 0001 111x indicates pods 1 through 4 are assigned to this analyzer (x = unused bit).

25 **1 byte** - This byte returns which chip is used to store the time or state tags when an unassigned pod is available to store tag data. This chip is available in state data mode with an unassigned pod and state or time tags on. Byte 21 = 2 in this mode.

26 **1 byte** - Master chip for this analyzer. This decimal value returns which chip's time tag data is valid in a non-transitional mode; for example, state with time tags.

Master card	Expansion card
5 - pods 1 and 2	2 - pods 1 and 2
4 - pods 3 and 4	1 - pods 3 and 4
3 - pods 5 and 6	0 - pods 5 and 6
	– 1 - no chip

27 **6 bytes** - Unused

33 **8 bytes** - A decimal integer representing sample period in picoseconds (timing only). For example, the following 64 bits in binary:

```
00000000 00000000 00000000 00000000 00000000 00000000 00011111 01000000
```

would equal 8,000 picoseconds or, 8 nanoseconds.

41 **8 bytes** - Unused

49 **1 byte** - Tag type for state only in one of the following decimal values:
0 = off
1 = time tags
2 = state tags

50 **1 bytes** - Unused

51 **8 bytes** - A decimal integer representing the time offset in picoseconds from when this analyzer is triggered and when this analyzer provides an output trigger to the IMB or port out. The value for one analyzer is always zero and the value for the other analyzer is the time between the triggers of the two analyzers.

59 **2 bytes** - Unused

61 **40 bytes** - The next 40 bytes are for Analyzer 2 Data Information. They are organized in the same manner as Analyzer 1 above, but they occupy bytes 61 through 100.

101 **26 bytes** - Number of valid rows of data (starting at byte 177) for each pod. The 26 bytes of this group are organized as follows:
Bytes 1 and 2 - Unused
Bytes 3 and 4 contain the number of valid rows of data for pod 6 of the expansion card in a two-card configuration. These two bytes are not used for a single-card configuration.
Bytes 5 and 6 contain the number of valid rows of data for pod 5 of the expansion card in a two-card configuration. These two bytes are not used for a single-card configuration.
Bytes 7 and 8 contain the number of valid rows of data for pod 4 of the expansion card in a two-card configuration. These two bytes are not used for a single-card configuration.
Bytes 9 and 10 contain the number of valid rows of data for pod 3 of the expansion card in a two-card configuration. These two bytes are not used for a single-card configuration.
Bytes 11 and 12 contain the number of valid rows of data for pod 2 of the expansion card in a two-card configuration. These two bytes are not used for a single-card configuration.
Bytes 13 and 14 contain the number of valid rows of data for pod 1 of the expansion card in a two-card configuration. These two bytes are not used for a single-card configuration.
Bytes 15 and 16 contain the number of valid rows of data for pod 6 of the master card in a two-card configuration or for a single-card configuration.
Bytes 17 and 18 contain the number of valid rows of data for pod 5 of the master card in a two-card configuration or for a single-card configuration.
Bytes 19 and 20 contain the number of valid rows of data for pod 4 of the master card in a two-card configuration or for a single-card configuration.
Bytes 21 and 22 contain the number of valid rows of data for pod 3 of the master card in a two-card configuration or for a single-card configuration.
Bytes 23 and 24 contain the number of valid rows of data for pod 2 of the master card in a two-card configuration or for a single-card configuration.
Bytes 25 and 26 contain the number of valid rows of data for pod 1 of the master card in a two-card configuration or for a single-card configuration.

127 **26 bytes** - Row of data containing the trigger point. This byte group is organized in the same way as the data rows (starting at byte 101 above). These binary numbers are base zero numbers which start from the first sample stored for a specific pod. For example, if bytes 151 and 152 contained a binary number with a decimal equivalent of +1018, the data row having the trigger is the 1018th data row on pod 1. There are 1018 rows of pre-trigger data as shown below.

row 0
row 1
.
.
.
row 1017
row 1018 – trigger row

153 **24 bytes** - Unused

Acquisition Data Description

The acquisition data section consists of a variable number of bytes depending on a one- or two-card configuration, the acquisition mode and the tag setting (time, state, or off). The data is grouped in 14-byte rows for a single card analyzer or in 26-byte rows for a two-card analyzer. The number of rows for each pod is stored in byte positions 101 through 126.

	clock lines	pod 6	pod 5	pod 4	pod 3	pod 2	pod 1*
177	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
191	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.
(x)	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

The clock line bytes for a one-card configuration are organized as follows:
`xxxx xxxx xxPN MLKJ`

In the following two-card analyzer example the data appears to be two rows; however, it is one continuous row.

	clock lines	pod 12	pod 11	pod 10	pod 9	pod 8	pod 7*
177	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
	pod 6	pod 5	pod4	pod 3	pod 2	pod 1*	
	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	

203 Row two - same as above

. .

. .

. .

(x) Row (x) - same as above

The clock line bytes for a two-card configuration are organized as follows:
`xxPN MLKJ xxPN MLKJ`

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

Row (x) is the highest number of valid rows specified by the bytes in byte positions 101 through 126 in all modes except glitch. In the glitch mode, row (x) is the larger of:

1. The highest number of valid rows specified by the bytes in byte positions 101 through 126; or,
2. 2048 + the highest number of valid rows for the pods assigned to the timing analyzer.

Time Tag Data Description

The time tag data starts at the end of the acquired data. Each data row has an 8-byte time tag for each chip (2-pod set). The starting location of the time tag data is immediately after the last row of valid data (maximum data byte + 1). If an analyzer is in a non-transitional mode, the master chip (byte 26) is the only chip with valid time-tag data. The time tag data is a decimal integer representing time in picoseconds for both timing and state time tags. For state tags in the state analyzer, tag data is a decimal integer representing the number of states.

Time Tag Block (One-card)

Byte 1 through 8 (64 bits starting with the MSB) - First sample tag for pods 1 and 2.

Byte 9 through 16 (64 bits starting with the MSB) - Second sample tag for pods 1 and 2.

.
.
.

Byte (x) through (x + 7) (64 bits starting with the MSB) - Last sample tag for pods 1 and 2.

Byte (x + 8) through (x + 15) (64 bits starting with the MSB) - First sample tag for pods 3 and 4.

Byte (x + 16) through (x + 23) (64 bits starting with the MSB) - Second sample tag for pods 3 and 4.

.
.
.

Byte (y) through (y + 7) (64 bits starting with the MSB) - Last sample tag for pods 3 and 4.

Byte (y + 8) through (y + 15) (64 bits starting with the MSB) - First sample tag for pods 5 and 6.

Byte (y + 16) through (y + 23) (64 bits starting with the MSB) - Second sample tag for pods 5 and 6.

.
.
.

Byte (z) through (z + 7) (64 bits starting with the MSB) - Last sample tag for pods 5 and 6.

Time Tag Block (Two-cards)

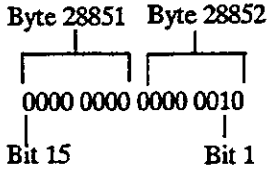
The description of the one-card time tag block is the same for the master card in a two-card configuration. The time tag block for the expansion card of a two-card configuration follows the master card as follows:

Master card
Pods 1 and 2
Pods 3 and 4
Pods 5 and 6
Expansion card
Pods 1 and 2
Pods 3 and 4
Pods 5 and 6

Glitch Data Description

In the glitch mode, each pod has two bytes assigned to indicate where glitches occur in the acquired data. For each row of acquired data there will be a corresponding row of glitch data. The glitch data is organized in the same way as the acquired data. The glitch data is grouped in 14-byte rows for a single card analyzer or in 26-byte rows for a two-card analyzer. The number of rows is stored in byte positions 101 through 126. The starting byte of the glitch data is an absolute starting point regardless of the number of rows of acquired data.

A binary 1 in the glitch data indicates a glitch was detected. For example, if a glitch occurred on bit 1 of pod 6 in data row 1 of a one-card configuration, bytes 28851 and 28852 would contain:



First byte of glitch data		clock lines	pod 6	pod 5	pod 4	pod 3	pod 2	pod 1*
———	28849	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
	28863	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	(x)	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes

In the following two-card analyzer example the glitch data appears to be two rows; however, it is one continuous row.

First byte of glitch data		clock lines	pod 12	pod 11	pod 10	pod 9	pod 8	pod 7*
———	53425	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes
		pod 6	pod 5	pod4	pod 3	pod 2	pod 1*	
		2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	2 bytes	
	53451	Row two - same as above						
	•		•					
	•		•					
	•		•					
	(x)	Row (x) - same as above						

*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. The length of the configuration data block can be up to 877,016 bytes in a two-card configuration.

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

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

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

- "SYMBOLS A "
- "SYMBOLS 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 (32 for the HP 16550A 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: See "Transferring the logic analyzer configuration" on page 17-14 in Chapter 17, "Program Examples" for an example.

Programming Examples 17

Introduction

This chapter contains short, usable, and tested program examples that cover the most asked for examples. The examples are written in HP BASIC 6.0.

- Making a Timing analyzer measurement
- Making a State analyzer measurement
- Making a State Compare measurement
- Transferring Logic Analyzer configuration between the logic analyzer and the controller
- Transferring Logic Analyzer data between the logic analyzer and the controller
- Checking for measurement completion
- Sending queries to the logic analyzer

Making a Timing analyzer measurement

This program sets up the logic analyzer to make a simple timing analyzer measurement. This example can be used with E2422-60002 Logic Analyzer Training board to acquire and display the output of the ripple counter. It can also be modified to make any timing analyzer measurement.

```
10   ! ****************** TIMING ANALYZER EXAMPLE ******************
20   !                for the HP 16550A Logic Analyzer
30   !
40   ! ***************************************************************
50   ! Select the module slot in which the HP 16550A is installed.  In
60   ! this example, the HP 16550A is in slot B of the mainframe.
70   !
80   OUTPUT 707;":SELECT 2"
90   !
100  ! ***************************************************************
110  ! Name Machine 1 "TIMING," configure Machine 1 as a timing analyzer,
120  ! and assign pod 1 to Machine 1.
130  !
140  OUTPUT 707;":MACH1:NAME 'TIMING'"
150  OUTPUT 707;":MACH1:TYPE TIMING"
160  OUTPUT 707;":MACH1:ASSIGN 1"
170  !
180  ! ***************************************************************
190  ! Make a label "COUNT," give the label a positive polarity, and
200  ! assign the lower 8 bits.
210  !
220  OUTPUT 707;":MACHINE1:TFORMAT:REMOVE ALL"
230  OUTPUT 707;":MACH1:TFORMAT:LABEL 'COUNT',POS,0,0,#B00000000111111111"
240  !
250  ! ***************************************************************
260  ! Specify FF hex for resource term A, which is the default trigger term for
270  ! the timing analyzer.
280  !
290  OUTPUT 707;":MACH1:TTRACE:TERM A, 'COUNT', '#HFF'"
300  !
```

```
310  ! ****************************************************************
320  ! Remove any previously inserted labels, insert the "COUNT"
330  ! label, change the seconds-per-division to 100 ns, and display the
340  ! waveform menu.
350  !
360  OUTPUT 707;":MACH1:TWAVEFORM:REMOVE"
370  OUTPUT 707;":MACH1:TWAVEFORM:INSERT 'COUNT', ALL"
380  OUTPUT 707;":MACH1:TWAVEFORM:RANGE 1E-6"
390  OUTPUT 707;":MENU 2,5"
400  !
410  ! ****************************************************************
420  ! Run the timing analyzer in single mode.
430  !
440  OUTPUT 707;":RMODE SINGLE"
450  OUTPUT 707;":START"
460  !
470  ! ****************************************************************
480  ! Set the marker mode (MMODE) to time so that time tags are available
490  ! for marker measurements.  Place the X-marker on 03 hex and the O-
500  ! marker on 07 hex.  Then tell the timing analyzer to find the first
510  ! occurrence of 03h after the trigger and the first occurrence of 07h
520  ! after the X-marker is found.
530  !
540  OUTPUT 707;":MACHINE1:TWAVEFORM:MMODE TIME"
550  !
560  OUTPUT 707;":MACHINE1:TWAVEFORM:XPATTERN 'COUNT','#H03'"
570  OUTPUT 707;":MACHINE1:TWAVEFORM:OPATTERN 'COUNT','#H07'"
580  !
590  OUTPUT 707;":MACHINE1:TWAVEFORM:XCONDITION ENTERING"
600  OUTPUT 707;":MACHINE1:TWAVEFORM:OCONDITION ENTERING"
610  !
620  OUTPUT 707;":MACHINE1:TWAVEFORM:XSEARCH +1, TRIGGER"
630  OUTPUT 707;":MACHINE1:TWAVEFORM:OSEARCH +1, XMARKER"
```

```
640   !
650   ! ********************************************************************
660   ! Turn the longform and headers on, dimension a string for the query
670   ! data, send the XOTIME query and print the string containing the
680   ! XOTIME query data.
690   !
700   OUTPUT 707;":SYSTEM:LONGFORM ON"
710   OUTPUT 707;":SYSTEM:HEADER ON"
720   !
730   DIM Mtime$[100]
740   OUTPUT 707;":MACHINE1:TWAVEFORM:XOTIME?"
750   ENTER 707;Mtime$
760   PRINT Mtime$
770   END
```

Making a State analyzer measurement

This state analyzer program selects the HP 16550A card, displays the configuration menu, defines a state machine, displays the state trigger menu, sets a state trigger for multilevel triggering. This program then starts a single acquisition measurement while checking for measurement completion.

This program is written in such a way you can run it with the HP E2433-60002 Logic Analyzer Training Board. This example is the same as the "Multilevel State Triggering" example in chapter 9 of the *HP E2433-90902 Logic Analyzer Training Guide.*

```
10   ! ********************* STATE ANALYZER EXAMPLE *************************
20   !                  for the HP 16550A Logic Analyzer
30   !
40   ! ******************* SELECT THE HP 16550A MODULE ***********************
50   ! Select the module slot in which the HP 16550A is installed.  In this
60   ! example, the HP 16550A is in slot B of the mainframe.
70   !
80   OUTPUT 707;":SELECT 2"
90   !
100  ! ********************* CONFIGURE THE STATE ANALYZER ***********************
110  ! Name Machine 1 "STATE," configure Machine 1 as a state analyzer, assign
120  ! pod 1 to Machine 1, and display System Configuration menu of the
130  ! HP 16550A.
140  !
150  OUTPUT 707;":MACHINE1:NAME 'STATE'"
160  OUTPUT 707;":MACHINE1:TYPE STATE"
170  OUTPUT 707;":MACHINE1:ASSIGN 1"
180  OUTPUT 707;":MENU 2,0"
190  !
200  ! ******************** SETUP THE FORMAT SPECIFICATION ***********************
210  ! Make a label "SCOUNT," give the label a positive polarity, and
220  ! assign the lower 8 bits.
230  !
240  OUTPUT 707;":MACHINE1:SFORMAT:REMOVE ALL"
250  OUTPUT 707;":MACHINE1:SFORMAT:LABEL 'SCOUNT', POS, 0,0,255"
260  !
270  ! ******************** SETUP THE TRIGGER SPECIFICATION **********************
280  ! The trigger specification will use five sequence levels with the trigger
```

```
290  ! level on level four.  Resource terms A through E, and RANGE1 will be
300  ! used to store only desired counts from the 8-bit ripple counter.
310  !
320  ! Display the state trigger menu.
330  !
340  OUTPUT 707;":MENU 2,3"
350  !
360  ! Create a 5 level trigger specification with the trigger on the
370  ! fourth level.
380  !
390  OUTPUT 707;":MACHINE1:STRIGGER:SEQUENCE 5,4"
400  !
410  ! Define pattern terms A, B, C, D, and E to be 11, 22, 33, 44 and 59
420  ! decimal respectively.
430  !
440  OUTPUT 707;":MACHINE1:STRIGGER:TERM A,'SCOUNT','11'"
450  OUTPUT 707;":MACHINE1:STRIGGER:TERM B,'SCOUNT','22'"
460  OUTPUT 707;":MACHINE1:STRIGGER:TERM C,'SCOUNT','33'"
470  OUTPUT 707;":MACHINE1:STRIGGER:TERM D,'SCOUNT','44'"
480  OUTPUT 707;":MACHINE1:STRIGGER:TERM E,'SCOUNT','59'"
490  !
500  ! Define a Range having a lower limit of 50 and an upper limit of 58.
510  !
520  OUTPUT 707;":MACHINE1:STRIGGER:RANGE1 'SCOUNT','50','58'"
530  !
540  ! ***************** CONFIGURE SEQUENCE LEVEL 1 ***************************
550  ! Store NOSTATE in level 1 and Then find resource term "A" once.
560  !
570  OUTPUT 707;":MACHINE1:STRIGGER:STORE1 'NOSTATE'"
580  OUTPUT 707;":MACHINE1:STRIGGER:FIND1 'A',1"
590  !
600  ! ***************** CONFIGURE SEQUENCE LEVEL 2 ***************************
610  ! Store RANGE1 in level 2 and Then find resource term "E" once.
620  !
630  OUTPUT 707;":MACHINE1:STRIGGER:STORE2 'IN_RANGE1'"
640  OUTPUT 707;":MACHINE1:STRIGGER:FIND2 'E',1"
650  !
660  ! ***************** CONFIGURE SEQUENCE LEVEL 3 ***************************
670  ! Store NOSTATE in level 3 and Then find term "B" once.
680  !
690  OUTPUT 707;":MACHINE1:STRIGGER:STORE3 'NOSTATE'"
```

```
700  OUTPUT 707;":MACHINE1:STRIGGER:FIND3 'B',1"
710  !
720  ! ***************** CONFIGURE SEQUENCE LEVEL 4 ***************************
730  ! Store a combination of resource terms (C or D or RANGE1) in level 4 and
740  ! Then Trigger on resource term "E."
750  !
760  OUTPUT 707;":MACHINE1:STRIGGER:STORE4 '(C OR D OR IN_RANGE1)'"
770  !
780  ! ************************ NOTE ************************
790  !      The FIND command selects the trigger in the
800  !      sequence level specified as the trigger level.
810  ! ******************************************************
820  !
830  OUTPUT 707;":MACHINE1:STRIGGER:FIND4 'E',1"
840  !
850  ! ***************** CONFIGURE SEQUENCE LEVEL 5 ***************************
860  ! Store anystate on level 5
870  !
880  OUTPUT 707;":MACHINE1:STRIGGER:STORE5 'ANYSTATE'"
890  !
900  ! ****************** START ACQUISITION ***********************************
910  ! Place the logic analyzer in single acquisition mode, then determine when
920  ! the acquisition is complete.
930  !
940  OUTPUT 707;":RMODE SINGLE"
950  !OUTPUT 707;"*CLS"
960  OUTPUT 707;":START"
970  !
980  ! ******************* CHECK FOR MEASUREMENT COMPLETE *********************
990  ! Enable the MESR register and query the register for a measurement
1000 ! complete condition.
1010 !
1020 OUTPUT 707;":SYSTEM:HEADER OFF"
1030 OUTPUT 707;":SYSTEM:LONGFORM OFF"
1040 !
1050 Status=0
1060 OUTPUT 707;":MESE2 1"
1070 OUTPUT 707;":MESR2?"
1080 ENTER 707;Status
1090 !
1100 ! Print the MESR register status.
```

```
1110  !
1120  CLEAR SCREEN
1130  PRINT "Measurement complete status is ";Status
1140  PRINT "0 = not complete, 1 = complete"
1150  ! Repeat the MESR query until measurement is complete.
1160  WAIT 1
1170  IF Status=1 THEN GOTO 1190
1180  GOTO 1070
1190  PRINT TABXY(30,15);"Measurement is complete"
1200  !
1210  ! ************************ VIEW THE RESULTS ******************************
1220  ! Display the State Listing and select a line number in the listing that
1230  ! allows you to see the beginning of the listing on the logic analyer
1240  ! display.
1250  !
1260  OUTPUT 707;":MACHINE1:SLIST:COLUMN 1, 'SCOUNT', DECIMAL"
1270  OUTPUT 707;":MENU 2,7"
1280  OUTPUT 707;":MACHINE1:SLIST:LINE -16"
1290  !
1300  END
```

Making a State Compare measurement

This program example acquires a state listing, copies the listing to the compare listing, acquires another state listing, and compares both listings to find differences.

This program is written in such a way you can run it with the HP E2433-60002 Logic Analyzer Training Board. This example is the same as the "State Compare" example in chapter 3 of the *HP E2433-90902 Logic Analyzer Training Guide*.

```
10   ! ********** STATE COMPARE EXAMPLE ********************************
20   !            for the HP 16550A Logic Analyzer
30   !
40   !
50   !************** SELECT THE HP 16550A MODULE ************************
60   ! Select the module slot in which the HP 16550A is installed.  In this
70   ! example, the HP 16550A is in slot B of the mainframe.
80   !
90   OUTPUT 707;":SELECT 2"
100  !
110  !************** CONFIGURE THE STATE ANALYZER ************************
120  ! Name Machine 1 "STATE," configure Machine 1 as a state analyzer, and
130  ! assign pod 1 to Machine 1.
140  !
150  OUTPUT 707;":MACHINE1:NAME 'STATE'"
160  OUTPUT 707;":MACHINE1:TYPE STATE"
170  OUTPUT 707;":MACHINE1:ASSIGN 1"
180  !
190  ! ********************************************************************
200  ! Remove all labels previously set up, make a label "SCOUNT," specify
210  ! positive logic, and assign the lower 8 bits of pod 1 to the label.
220  !
230  OUTPUT 707;":MACHINE1:SFORMAT:REMOVE ALL"
240  OUTPUT 707;":MACHINE1:SFORMAT:LABEL 'SCOUNT', POS, 0,0,255"
250  !
260  ! ********************************************************************
270  ! Make the "J" clock the Master clock and specify the falling edge.
280  !
```

```
290   OUTPUT 707;":MACHINE1:SFORMAT:MASTER J, FALLING"
300   !
310   ! ********************************************************************
320   ! Specify two sequence levels, the trigger sequence level, specify
330   ! FF hex for the "a" term which will be the trigger term, and store
340   ! no states until the trigger is found.
350   !
360   OUTPUT 707;":MACHINE1:STRIGGER:SEQUENCE 2,1"
370   OUTPUT 707;":MACHINE1:STRIGGER:TERM A,'SCOUNT','#HFF'"
380   OUTPUT 707;":MACHINE1:STRIGGER:STORE1 'NOSTATE'"
390   OUTPUT 707;":MENU 2,3"
400   !
410   ! ********************************************************************
420   ! Change the displayed menu to the state listing and start the state
430   ! analyzer in repetitive mode.
440   !
450   OUTPUT 707;":MENU 2,7"
460   OUTPUT 707;":RMODE REPETITIVE"
470   OUTPUT 707;":START"
480   !
490   ! ********************************************************************
500   ! The logic analyzer is now running in the repetitive mode
510   ! and will remain in repetitive until the STOP command is sent.
520   !
530   PRINT "The logic analyzer is now running in the repetitive mode"
540   PRINT "and will remain in repetitive until the STOP command is sent."
550   PRINT
560   PRINT "Press CONTINUE"
570   PAUSE
580   !
590   !*********************************************************************
600   ! Stop the acquisition and copy the acquired data to the compare reference
610   ! listing.
620   !
630   OUTPUT 707;":STOP"
640   OUTPUT 707;":MENU 2,10"
650   OUTPUT 707;":MACHINE1:COMPARE:MENU REFERENCE"
660   OUTPUT 707;":MACHINE1:COMPARE:COPY"
670   !
680   ! The logic analyzer acquistion is now stopped, the Compare menu
690   ! is displayed, and the data is now in the compare reference
```

```
700  ! listing.
710  !
720  !*************************************************************************
730  ! Display line 4090 of the compare listing and start the analyzer
740  ! in a repetitive mode.
750  !
760  OUTPUT 707;":MACHINE1:COMPARE:LINE 4090"
770  OUTPUT 707;":START"
780  !
790  ! Line 4090 of the listing is now displayed at center screen
800  ! in order to show the last four states acquired.  In this
810  ! example, the last four states are stable.  However, in some
820  ! cases, the end points of the listing may vary thus causing
830  ! a false failure in compare.  To eliminate this problem, a
840  ! partial compare can be specified to provide predicable end
850  ! points of the data.
860  !
870  PRINT "Press CONTINUE to send the STOP command."
880  PAUSE
890  OUTPUT 707;":STOP"
900  !
910  !*************************************************************************
920  ! The end points of the compare can be fixed to prevent false failures.
930  ! In addition, you can use partial compare to compare only sections
940  ! of the state listing you are interested in comparing.
950  !
960  OUTPUT 707;":MACHINE1:COMPARE:RANGE PARTIAL, 0, 508"
970  !
980  ! The compare range is now from line 0 to +508
990  !
1000 !*************************************************************************
1010 ! Change the Glitch jumper settings on the training board so that the
1020 ! data changes, reacquire the data and compare which states are different.
1030 PRINT "Change the glitch jumper settings on the training board so that the"
1040 PRINT "data changes, reacquire the data and compare which states are different."
1050 !
1060 PRINT "Press CONTINUE when you have finished changing the jumper."
1070 !
1080 PAUSE
1090 !
1100 !*************************************************************************
```

```
1110  ! Start the logic analyzer to acquire new data and then stop it to compare
1120  ! the data.  When the acquistion is stopped, the Compare Listing Menu will
1130  ! be displayed.
1140  !
1150  OUTPUT 707;":START"
1160  OUTPUT 707;":STOP"
1170  OUTPUT 707;":MENU 2,10"
1180  !
1190  !*************************************************************************
1200  ! Dimension strings in which the compare find query (COMPARE:FIND?)
1210  ! enters the line numbers and error numbers.
1220  !
1230  DIM Line$[20]
1240  DIM Error$[4]
1250  DIM Comma$[1]
1260  !
1270  ! ************************************************************************
1280  ! Display the Difference listing.
1290  !
1300  OUTPUT 707;":MACHINE1:COMPARE:MENU DIFFERENCE"
1310  !
1320  ! ************************************************************************
1330  ! Loop to query all 508 possible errors.
1340  !
1350  FOR Error=1 TO 508
1360  !
1370  ! Read the compare differences
1380  !
1390  OUTPUT 707;":MACHINE1:COMPARE:FIND? "&VAL$(Error)
1400  !
1410  ! ************************************************************************
1420  ! Format the Error$ string data for display on the controller screen.
1430  !
1440  IF Error>99 THEN GOTO 1580
1450  IF Error>9 THEN GOTO 1550
1460  !
1470  ENTER 707 USING "#,1A";Error$
1480  ENTER 707 USING "#,1A";Comma$
1490  ENTER 707 USING "K";Line$
1500  Error_return=IVAL(Error$,10)
1510  IF Error_return=0 THEN GOTO 1820
```

```
1520  !
1530  GOTO 1610
1540  !
1550  ENTER 707 USING "#,3A";Error$
1560  ENTER 707 USING "K";Line$
1570  GOTO 1610
1580  !
1590  ENTER 707 USING "#,4A";Error$
1600  ENTER 707 USING "K";Line$
1610  !
1620  ! ************************************************************************
1630  ! Test for the last error.  The error number of the last error is the same
1640  ! as the error number of the first number after the last error.
1650  !
1660  Error_line=IVAL(Line$,10)
1670  IF Error_line=Error_line2 THEN GOTO 1780
1680  Error_line2=Error_line
1690  !
1700  ! ************************************************************************
1710  ! Print the error numbers and the corresponding line numbers on the
1720  ! controller screen.
1730  !
1740  PRINT "Error number ",Error," is on line number ",Error_line
1750  !
1760  NEXT Error
1770  !
1780  PRINT
1790  PRINT
1800  PRINT "Last error found"
1810  GOTO 1850
1820  PRINT "No errors found"
1830  !
1840  !
1850  END
```

Transferring the logic analyzer configuration

This program uses the SYSTem:SETup query to transfer the configuration of the logic analyzer to your controller. This program also uses the SYSTem:SETup command to transfer a logic analyzer configuration from the controller back to the logic analyzer. The configuration data will set up the logic analyzer according to the data. It is useful for getting configurations for setting up the logic analyzer by the controller. This query differs from the SYSTem:DATA query because it only transfers the configuration and not the acquired data. The SYSTem:SETup command differs from the SYSTem:DATA command because it only transfers the configuration and not acquired data.

```
10   ! ****************** SETUP COMMAND AND QUERY EXAMPLE ********************
20   !                          for the HP 16550A
30   !
40   ! ********************** CREATE TRANSFER BUFFER *************************
50   ! Create a buffer large enough for the block data.  See page 16-9 for
55   ! maximum block length.
56   !
60   ASSIGN @Buff TO BUFFER [170000]
70   !
80   ! ***************** INITIALIZE HPIB DEFAULT ADDRESS ***********************
90   !
100  REAL Address
110  Address=707
120  ASSIGN @Comm TO Address
130  !
140  CLEAR SCREEN
150  !
160  ! ************* INTITIALIZE VARIABLE FOR NUMBER OF BYTES ******************
170  ! The variable "Numbytes" contains the number of bytes in the buffer.
180  !
190  REAL Numbytes
200  Numbytes=0
210  !
220  ! ************** RE-INITIALIZE TRANSFER BUFFER POINTERS *******************
230  !
240  CONTROL @Buff,3;1
250  CONTROL @Buff,4;0
```

```
260  !
270  ! ********************** SEND THE SETUP QUERY **************************
280  OUTPUT 707;":SYSTEM:HEADER ON"
290  OUTPUT 707;":SYSTEM:LONGFORM ON"
300  OUTPUT @Comm;"SELECT 2"
310  OUTPUT @Comm;":SYSTEM:SETUP?"
320  !
330  ! ******************** ENTER THE BLOCK SETUP HEADER ********************
340  ! Enter the block setup header in the proper format.
350  !
360  ENTER @Comm USING "#,B";Byte
370  PRINT CHR$(Byte);
380  WHILE Byte<>35
390    ENTER @Comm USING "#,B";Byte
400    PRINT CHR$(Byte);
410  END WHILE
420  ENTER @Comm USING "#,B";Byte
430  PRINT CHR$(Byte);
440  Byte=Byte-48
450  IF Byte=1 THEN ENTER @Comm USING "#,D";Numbytes
460  IF Byte=2 THEN ENTER @Comm USING "#,DD";Numbytes
470  IF Byte=3 THEN ENTER @Comm USING "#,DDD";Numbytes
480  IF Byte=4 THEN ENTER @Comm USING "#,DDDD";Numbytes
490  IF Byte=5 THEN ENTER @Comm USING "#,DDDDD";Numbytes
500  IF Byte=6 THEN ENTER @Comm USING "#,DDDDDD";Numbytes
510  IF Byte=7 THEN ENTER @Comm USING "#,DDDDDDD";Numbytes
520  IF Byte=8 THEN ENTER @Comm USING "#,DDDDDDDD";Numbytes
530  PRINT Numbytes
540  !
550  ! ******************** TRANSER THE SETUP ********************************
560  ! Transfer the setup from the logic analyzer to the buffer.
570  !
580  TRANSFER @Comm TO @Buff;COUNT Numbytes,WAIT
600  !
610  ENTER @Comm USING "-K";Length$
620  PRINT "LENGTH of Length string is";LEN(Length$)
630  !
640  PRINT "***** GOT THE SETUP ****"
650  PAUSE
```

```
660   ! ********** SEND THE SETUP **********
670   ! Make sure buffer is not empty.
680   !
690   IF Numbytes=0 THEN
700     PRINT "BUFFER IS EMPTY"
710     GOTO 1170
720   END IF
730   !
740   ! ********** SEND THE SETUP COMMAND **********
750   ! Send the Setup command
760   !
770   OUTPUT @Comm USING "#,15A";":SYSTEM:SETUP #"
780   PRINT "SYSTEM:SETUP command has been sent"
790   PAUSE
800   !
810   ! ********** SEND THE BLOCK SETUP **********
820   ! Send the block setup header to the HP 16550A in the proper format.
830   !
840   Byte=LEN(VAL$(Numbytes))
850   OUTPUT @Comm USING "#,8";(Byte+48)
860   IF Byte=1 THEN OUTPUT @Comm USING "#,A";VAL$(Numbytes)
870   IF Byte=2 THEN OUTPUT @Comm USING "#,AA";VAL$(Numbytes)
880   IF Byte=3 THEN OUTPUT @Comm USING "#,AAA";VAL$(Numbytes)
890   IF Byte=4 THEN OUTPUT @Comm USING "#,AAAA";VAL$(Numbytes)
900   IF Byte=5 THEN OUTPUT @Comm USING "#,AAAAA";VAL$(Numbytes)
910   IF Byte=6 THEN OUTPUT @Comm USING "#,AAAAAA";VAL$(Numbytes)
920   IF Byte=7 THEN OUTPUT @Comm USING "#,AAAAAAA";VAL$(Numbytes)
930   IF Byte=8 THEN OUTPUT @Comm USING "#,AAAAAAAA";VAL$(Numbytes)
940   !
950   ! ********** SAVE BUFFER POINTERS **********
960   ! Save the transfer buffer pointer so it can be restored after the
970   ! transfer.
980   !
990   STATUS @Buff,5;Streg
1000  !
```

```
1010  ! ****************** TRANSFER SETUP TO THE HP 16550 *********************
1020  ! Transfer the setup from the buffer to the HP 16550A.
1030  !
1040  TRANSFER @Buff TO @Comm;COUNT Numbytes,WAIT
1050  !
1060  ! ********************** RESTORE BUFFER POINTERS ************************
1070  ! Restore the transfer buffer pointer
1080  !
1090  CONTROL @Buff,5;Streg
1100  !
1110  ! ******************** SEND TERMINATING LINE FEED ***********************
1120  ! Send the terminating linefeed to properly terminate the setup string.
1130  !
1140  OUTPUT @Comm;""
1150  !
1160  PRINT "**** SENT THE SETUP ****"
1170  END
```

Transferring the logic analyzer acquired data

This program uses the SYSTem:DATA query to transfer acquired data to your controller. It is useful for getting acquired data for setting up the logic analyzer by the controller at a later time. This query differs from the SYSTem:SETup query because it transfers only the acquired data.

This program also uses the SYSTem:DATA command to transfer the logic analyzer data from the controller back to the logic analyzer and load the analyzer with the acquired data. The SYSTem:DATA command differs from the SYSTem:SETup command because it transfers both the configuration and the acquired data.

Note You should always precede the SYSTem:DATA query and command with the SYSTem:SETup query and command if the acquired data depends on a specific configuration. If you are only interested in the acquired data for post processing in the controller and the data is not dependent on the configuration, you can use the SYSTem:DATA query and command alone.

```
10   ! ****************** DATA COMMAND AND QUERY EXAMPLE ********************
20   !                          for the HP 16550A
30   !
40   ! ********************** CREATE TRANSFER BUFFER *************************
50   ! Create a buffer large enough for the block data.  See page 16-1 for
55   ! maximum block length.
56   !
60   ASSIGN @Buff TO BUFFER [170000]
70   !
80   ! ***************** INITIALIZE HPIB DEFAULT ADDRESS **********************
90   !
100  REAL Address
110  Address=707
120  ASSIGN @Comm TO Address
130  !
140  CLEAR SCREEN
150  !
160  ! ************* INTITIALIZE VARIABLE FOR NUMBER OF BYTES ******************
170  ! The variable "Numbytes" contains the number of bytes in the buffer.
180  !
```

```
190  REAL Numbytes
200  Numbytes=0
210  !
220  ! ************** RE-INITIALIZE TRANSFER BUFFER POINTERS ******************
230  !
240  CONTROL @Buff,3;1
250  CONTROL @Buff,4;0
260  !
270  ! ********************** SEND THE DATA QUERY **************************
280  OUTPUT 707;":SYSTEM:HEADER ON"
290  OUTPUT 707;":SYSTEM:LONGFORM ON"
300  OUTPUT @Comm;"SELECT 2"
310  OUTPUT @Comm;":SYSTEM:DATA?"
320  !
330  ! ******************** ENTER THE BLOCK DATA HEADER *********************
340  ! Enter the block data header in the proper format.
350  !
360  ENTER @Comm USING "#,B";Byte
370  PRINT CHR$(Byte);
380  WHILE Byte<>35
390    ENTER @Comm USING "#,B";Byte
400    PRINT CHR$(Byte);
410  END WHILE
420  ENTER @Comm USING "#,B";Byte
430  PRINT CHR$(Byte);
440  Byte=Byte-48
450  IF Byte=1 THEN ENTER @Comm USING "#,D";Numbytes
460  IF Byte=2 THEN ENTER @Comm USING "#,DD";Numbytes
470  IF Byte=3 THEN ENTER @Comm USING "#,DDD";Numbytes
480  IF Byte=4 THEN ENTER @Comm USING "#,DDDD";Numbytes
490  IF Byte=5 THEN ENTER @Comm USING "#,DDDDD";Numbytes
500  IF Byte=6 THEN ENTER @Comm USING "#,DDDDDD";Numbytes
510  IF Byte=7 THEN ENTER @Comm USING "#,DDDDDDD";Numbytes
520  IF Byte=8 THEN ENTER @Comm USING "#,DDDDDDDD";Numbytes
530  PRINT Numbytes
540  !
550  ! ******************** TRANSER THE DATA ********************************
560  ! Transfer the data from the logic analyzer to the buffer.
570  !
580  TRANSFER @Comm TO @Buff;COUNT Numbytes,WAIT
600  !
```

```
610   ENTER @Comm USING "-K";Length$
620   PRINT "LENGTH of Length string is";LEN(Length$)
630   !
640   PRINT "**** GOT THE DATA ****"
650   PAUSE
660   ! ********************* SEND THE DATA ***********************************
670   ! Make sure buffer is not empty.
680   !
690   IF Numbytes=0 THEN
700     PRINT "BUFFER IS EMPTY"
710     GOTO 1170
720   END IF
730   !
740   ! ********************* SEND THE DATA COMMAND ***************************
750   ! Send the Setup command
760   !
770   OUTPUT @Comm USING "#,14A";":SYSTEM:DATA #"
780   PRINT "SYSTEM:DATA command has been sent"
790   PAUSE
800   !
810   ! ********************* SEND THE BLOCK DATA *****************************
820   ! Send the block data header to the HP 16550A in the proper format.
830   !
840   Byte=LEN(VAL$(Numbytes))
850   OUTPUT @Comm USING "#,B";(Byte+48)
860   IF Byte=1 THEN OUTPUT @Comm USING "#,A";VAL$(Numbytes)
870   IF Byte=2 THEN OUTPUT @Comm USING "#,AA";VAL$(Numbytes)
880   IF Byte=3 THEN OUTPUT @Comm USING "#,AAA";VAL$(Numbytes)
890   IF Byte=4 THEN OUTPUT @Comm USING "#,AAAA";VAL$(Numbytes)
900   IF Byte=5 THEN OUTPUT @Comm USING "#,AAAAA";VAL$(Numbytes)
910   IF Byte=6 THEN OUTPUT @Comm USING "#,AAAAAA";VAL$(Numbytes)
920   IF Byte=7 THEN OUTPUT @Comm USING "#,AAAAAAA";VAL$(Numbytes)
930   IF Byte=8 THEN OUTPUT @Comm USING "#,AAAAAAAA";VAL$(Numbytes)
940   !
950   ! ********************** SAVE BUFFER POINTERS ************************
960   ! Save the transfer buffer pointer so it can be restored after the
970   ! transfer.
980   !
990   STATUS @Buff,5;Streg
1000  !
```

```
1010  ! ****************** TRANSFER DATA TO THE HP 16550 **********************
1020  ! Transfer the data from the buffer to the HP 16550A.
1030  !
1040  TRANSFER @Buff TO @Comm;COUNT Numbytes,WAIT
1050  !
1060  ! ********************** RESTORE BUFFER POINTERS ************************
1070  ! Restore the transfer buffer pointer
1080  !
1090  CONTROL @Buff,5;Streg
1100  !
1110  ! ******************** SEND TERMINATING LINE FEED ***********************
1120  ! Send the terminating linefeed to properly terminate the data string.
1130  !
1140  OUTPUT @Comm;""
1150  !
1160  PRINT "**** SENT THE DATA ****"
1170  END
```

Checking for measurement completion

This program can be appended to or inserted into another program when you need to know when a measurement is complete. If it is at the end of a program it will tell you when measurement is complete. If you insert it into a program, it will halt the program until the current measurement is complete. In this example, the module installed in slot B is being checked for measurement complete.

This program is also in the state analyzer example program in "Making a State Analyzer Measurement" on pages 17-7 and 17-8. It is included in the state analyzer example program to show how it can be used in a program to halt the program until measurement is complete.

```
420  ! ****************** CHECK FOR MEASUREMENT COMPLETE ***********************
430  ! Enable the MESR register and query the register for a measurement
440  ! complete condition.
450  !
460  OUTPUT 707;":SYSTEM:HEADER OFF"
470  OUTPUT 707;":SYSTEM:LONGFORM OFF"
480  !
490  Status=0
500  OUTPUT 707;":MESE2 1"
510  OUTPUT 707;":MESR2?"
520  ENTER 707;Status
530  !
540  ! Print the MESR register status.
550  !
560  CLEAR SCREEN
570  PRINT "Measurement complete status is ";Status
580  PRINT "0 = not complete, 1 = complete"
590  ! Repeat the MESR query until measurement is complete.
600  WAIT 1
610  IF Status=1 THEN GOTO 630
620  GOTO 510
630  PRINT TABXY(30,15);"Measurement is complete"
640  !
650  END
```

Sending queries to the logic analyzer

This program example contains the steps required to send a query to the logic analyzer. Sending the query alone only puts the requested information in an output buffer of the logic analyzer. You must follow the query with an ENTER statement to transfer the query response to the controller. When the query response is sent to the logic analyzer, the query is properly terminated in the logic analyer. If you send the query but fail to send an ENTER statement, the logic analyzer will display the error message "Query Interrupted" when it receives the next command from the controller, and, the query response is lost.

```
10   !************************ QUERY EXAMPLE ************************
20   !                for the HP 16550A Logic Analyzer
30   !
40   ! ************************ OPTIONAL ****************************
50   ! The following two lines turn the headers and longform on so
60   ! that the query name, in its long form, is included in the
70   ! query response.
80   !
90   !          ************** NOTE ****************
100  !          If your query response includes real
110  !          or integer numbers that you may want
120  !          to do statistics or math on later, you
130  !          should turn both header and longform
140  !          off so only the number is returned.
150  !          ************************************
160  !
170  OUTPUT 707;":SYSTEM:HEADER ON"
180  OUTPUT 707;":SYSTEM:LONGFORM ON"
190  !
200  ! ***************************************************************
210  ! Select the slot in which the HP 16550A is located.
220  !
230  OUTPUT 707;":SELECT 2"
240  !
250  ! ***************************************************************
260  ! Dimension a string in which the query response will be entered.
270  !
```

```
280   DIM Query$[100]
290   !
300   ! ***************************************************************
310   ! Send the query.  In this example the MENU? query is sent.  All
320   ! queries except the SYSTem:DATA and SYSTem:SETup can be sent with
330   ! this program.
340   !
350   OUTPUT 707;"MENU?"
360   !
370   ! ***************************************************************
380   ! The two lines that follow transfer the query response from the
390   ! query buffer to the controller and then print the response.
400   !
410   ENTER 707;Query$
420   PRINT Query$
430   !
440   !
450   END
```

Index

D

E

Q

R

S

T

V

W

X