HP 75000 Series C

HP E1422A Remote Channel Multi-function DAC Module with

HP E1529A 32ch Remote Strain Conditioning Unit and HP E1539A Remote Channel Signal Conditioning Plug-on

User's and SCPI Programming Manual

Where to Find it - Online and Printed Information:

System installation (hardware/software) .... VXIbus Configuration Guide*
HP VIC (VXI installation software)*
Getting Started with VXI Guide**

Module configuration and wiring .... This Manual
SCPI programming .... This Manual
SCPI example programs .... This Manual, Driver Disk
SCPI command reference .... This Manual

VXI*plug&play* programming .... VXI*plug&play* Online Help
VXI*plug&play* example programs .... VXI*plug&play* Online Help
VXI*plug&play* function reference .... VXI*plug&play* Online Help
Soft Front Panel information .... VXI*plug&play* Online Help

VISA language information .... HP VISA User's Guide

HP VEE programming information .... HP VEE User's Manual

**Supplied with HP Command Modules , Embedded Controllers, and VXLink*
** Supplied with HP VXI Mainframes and on WEB at www.tmo.hp.com then search for "getting started with vxi". Search result will be "Getting Started With VXI Guide", click on this to download .PDF file.

HEWLETT® PACKARD

Manual Part Number: E1422-90000
Printed in U.S.A. E0199

HEWLETT-PACKARD WARRANTY STATEMENT

HP PRODUCT: HP E1422A Remote Channel Multi-function DAC Module with — **DURATION OF WARRANTY:** 3 years
HP E1529A Remote Strain Conditioning Module and — **DURATION OF WARRANTY:** 3 years
HP E1539A Remote Channel Signal Conditioning Plug-on and
all other applicable Signal Conditioning Plug-ons — **DURATION OF WARRANTY:** 3 years

1. HP warrants HP hardware, accessories and supplies against defects in materials and workmanship for the period specified above. If HP receives notice of such defects during the warranty period, HP will, at its option, either repair or replace products which prove to be defective. Replacement products may be either new or like-new.

2. HP warrants that HP software will not fail to execute its programming instructions, for the period specified above, due to defects in material and workmanship when properly installed and used. If HP receives notice of such defects during the warranty period, HP will replace software media which does not execute its programming instructions due to such defects.

3. HP does not warrant that the operation of HP products will be interrupted or error free. If HP is unable, within a reasonable time, to repair or replace any product to a condition as warranted, customer will be entitled to a refund of the purchase price upon prompt return of the product.

4. HP products may contain remanufactured parts equivalent to new in performance or may have been subject to incidental use.

5. The warranty period begins on the date of delivery or on the date of installation if installed by HP. If customer schedules or delays HP installation more than 30 days after delivery, warranty begins on the 31st day from delivery.

6. Warranty does not apply to defects resulting from (a) improper or inadequate maintenance or calibration, (b) software, interfacing, parts or supplies not supplied by HP, (c) unauthorized modification or misuse, (d) operation outside of the published environmental specifications for the product, or (e) improper site preparation or maintenance.

7. TO THE EXTENT ALLOWED BY LOCAL LAW, THE ABOVE WARRANTIES ARE EXCLUSIVE AND NO OTHER WARRANTY OR CONDITION, WHETHER WRITTEN OR ORAL, IS EXPRESSED OR IMPLIED AND HP SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTY OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE.

8. HP will be liable for damage to tangible property per incident up to the greater of $300,000 or the actual amount paid for the product that is the subject of the claim, and for damages for bodily injury or death, to the extent that all such damages are determined by a court of competent jurisdiction to have been directly caused by a defective HP product.

9. TO THE EXTENT ALLOWED BY LOCAL LAW, THE REMEDIES IN THIS WARRANTY STATEMENT ARE CUSTOMER'S SOLE AND EXLUSIVE REMEDIES. EXCEPT AS INDICATED ABOVE, IN NO EVENT WILL HP OR ITS SUPPLIERS BE LIABLE FOR LOSS OF DATA OR FOR DIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL (INCLUDING LOST PROFIT OR DATA), OR OTHER DAMAGE, WHETHER BASED IN CONTRACT, TORT, OR OTHERWISE.

FOR CONSUMER TRANSACTIONS IN AUSTRALIA AND NEW ZEALAND: THE WARRANTY TERMS CONTAINED IN THIS STATEMENT, EXCEPT TO THE EXTENT LAWFULLY PERMITTED, DO NOT EXCLUDE, RESTRICT OR MODIFY AND ARE IN ADDITION TO THE MANDATORY STATUTORY RIGHTS APPLICABLE TO THE SALE OF THIS PRODUCT TO YOU.

U.S. Government Restricted Rights

The Software and Documentation have been developed entirely at private expense. They are delivered and licensed as "commercial computer software" as defined in DFARS 252.227- 7013 (Oct 1988), DFARS 252.211-7015 (May 1991) or DFARS 252.227-7014 (Jun 1995), as a "commercial item" as defined in FAR 2.101(a), or as "Restricted computer software" as defined in FAR 52.227-19 (Jun 1987)(or any equivalent agency regulation or contract clause), whichever is applicable. You have only those rights provided for such Software and Documentation by the applicable FAR or DFARS clause or the HP standard software agreement for the product involved.

HEWLETT® PACKARD

HP E1422A Remote Channel DAC Unit User's Manual and SCPI Programming Guide
Edition 1

Documentation History

All Editions and Updates of this manual and their creation date are listed below. The first Edition of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct or add additional information to the current Edition of the manual. Whenever a new Edition is created, it will contain all of the Update information for the previous Edition. Each new Edition or Update also includes a revised copy of this documentation history page.

Edition 1 . . . . .May 1999

Safety Symbols

Instruction manual symbol affixed to product. Indicates that the user must refer to the manual for specific WARNING or CAUTION information to avoid personal injury or damage to the product.

Indicates the field wiring terminal that must be connected to earth ground before operating the equipmentÅprotects against electrical shock in case of fault.

or Frame or chassis ground terminal—typically connects to the equipment's metal

Alternating current (AC)

Direct current (DC).

Indicates hazardous voltages.

WARNING Calls attention to a procedure, practice, or condition that could cause bodily injury or death.

CAUTION Calls attention to a procedure, practice, or condition that could possibly cause damage to equipment or permanent loss of data.

WARNINGS

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground the equipment: For Safety Class 1 equipment (equipment having a protective earth terminal), an uninterruptible safety earth ground must be provided from the mains power source to the product input wiring terminals or supplied power cable.

DO NOT operate the product in an explosive atmosphere or in the presence of flammable gases or fumes.

For continued protection against fire, replace the line fuse(s) only with fuse(s) of the same voltage and current rating and type. DO NOT use repaired fuses or short-circuited fuse holders.

Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers or shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.

DO NOT operate damaged equipment: Whenever it is possible that the safety protection features built into this product have been impaired, either through physical damage, excessive moisture, or any other reason, REMOVE POWER and do not use the product until safe operation can be verified by service-trained personnel. If necessary, return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

DO NOT service or adjust alone: Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

DO NOT substitute parts or modify equipment: Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Operating Location: Sheltered location where air temperature and humidity are controlled within this product’s specifications and the product is protected against direct exposure to climatic conditions such as direct sunlight, wind, rain, snow, sleet, and icing, water spray or splash, hoarfrost or dew. (Typically, indoor.) Pollution environment for which this product may be operated is IEC 664 Pollution degree 2.

CLEANING THE FRONT PANEL AND TOP/BOTTOM SHIELDS: Clean the outside surfaces of this module with a cloth slightly dampened with water. Do not attempt to clean the interior of this module.

Declaration of Conformity

according to ISO/IEC Guide 22 and EN 45014

Manufacturer's Name: Hewlett-Packard Company
Loveland Manufacturing Center

Manufacturer's Address: 815 14th Street S.W.
Loveland, Colorado 80537

declares, that the product:

Product Names: Remote Channel Multi-function DAC Module
Remote Channel Signal Conditioning Plug-on
32-ch Remote Strain Conditioning Unit

Model Numbers: HP E1422A
HP E1539A
HP E1529A

Product Options: All options and all other applicable Signal Conditioning Plug-ons

conforms to the following Product Specifications:

Safety: IEC 1010-1 (1990) Incl. Amend 2 (1996)/EN61010-1 (1993)
CSA C22.2 #1010.1 (1992)
UL 3111-1 (1994)

EMC: CISPR 11:1990/EN55011 (1991): Group 1 Class A
EN61000-3-2:1995 Class A
EN61000-3-3:1995
EN50082-1:1992
IEC 1000-4-2:1995: 4kVCD, 8kVAD
IEC 1000-4-3:1995: 3 V/m
IEC 1000-4-4:1995: 1kV Power Line 0.5kV Signal Lines
ENV50141:1993/prEN50082-1 (1995): 3 Vrms
EN 61000-4-5:1995 1kV CM, 0.5kV DM
EN61000-4-8:1993/prEN50082-1 (1995): 3 A/M
EN61000-4-11:1994/prEN50082-1 (1995): 30%, 10mS 60%, 100mS

Supplementary Information: The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (inclusive 93/68/EEC) and carries the "CE" mark accordingly.

May 3, 1999 **Jim White, QA Manager**

For Compliance Information ONLY, contact:

Australia Contact: Product Regulations Manager, Hewlett-Packard Australia Ltd., 31-41 Joseph Street, Blackburn, Victoria 3130, Australia

European Contact: Your local Hewlett-Packard Sales and Service Office or Hewlett-Packard GmbH, Department HQ-TRE, Standards Europe, Herrenberger Straβe 130, D-71034 Boblingen (FAX: +49-7031-14-3143)

USA Contact: Product Regulations Manager, Hewlett-Packard Company, P.O. Box 301, Mail Stop BU212, Loveland, CO 80537

Notes:

Please fold and tape for mailing

Reader Comment Sheet

HP E1422A Remote Channel Multifunction DAC Module User's Manual

Edition 1

You can help us improve our manuals by sharing your comments and suggestions. **In appreciation of your time, we will enter you in a quarterly drawing for a Hewlett-Packard Palmtop Personal Computer** (U.S. government employees are not eligible for the drawing).

Your Name	City, State/Province
Company Name	Country
Job Title	Zip/Postal Code
Address	Telephone Number with Area Code

Please list the system controller, operating system, programming language, and plug-in modules you are using.

fold here

cut along this line

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 37 LOVELAND, CO

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY

Measurement Systems Division
Learning Products Department
P.O. Box 301
Loveland, CO 80539-9984

fold here

Please pencil-in one circle for each statement below:	Disagree				Agree
• The documentation is well organized.	O	O	O	O	O
• Instructions are easy to understand.	O	O	O	O	O
• The documentation is clearly written.	O	O	O	O	O
• Examples are clear and useful.	O	O	O	O	O
• Illustrations are clear and helpful.	O	O	O	O	O
• The documentation meets my overall expectations.	O	O	O	O	O

Please write any comments or suggestions below–be specific.

Contents

HP E1422A Remote Channel Multifunction DAC Module (Edition 1)

Chapter 1
Getting Started

About this Chapter

This chapter will explain hardware configuration before installation in a VXIbus mainframe. By attending to each of these configuration items, your HP E1422 won't have to be removed from its mainframe later. Chapter contents include:

Configuring the HP E1422

There are several aspects to configuring the module before installing it in a VXIbus mainframe. They are:

For most applications you will **only need to change the Logical Address switch** prior to installation. The other settings can be used as delivered.

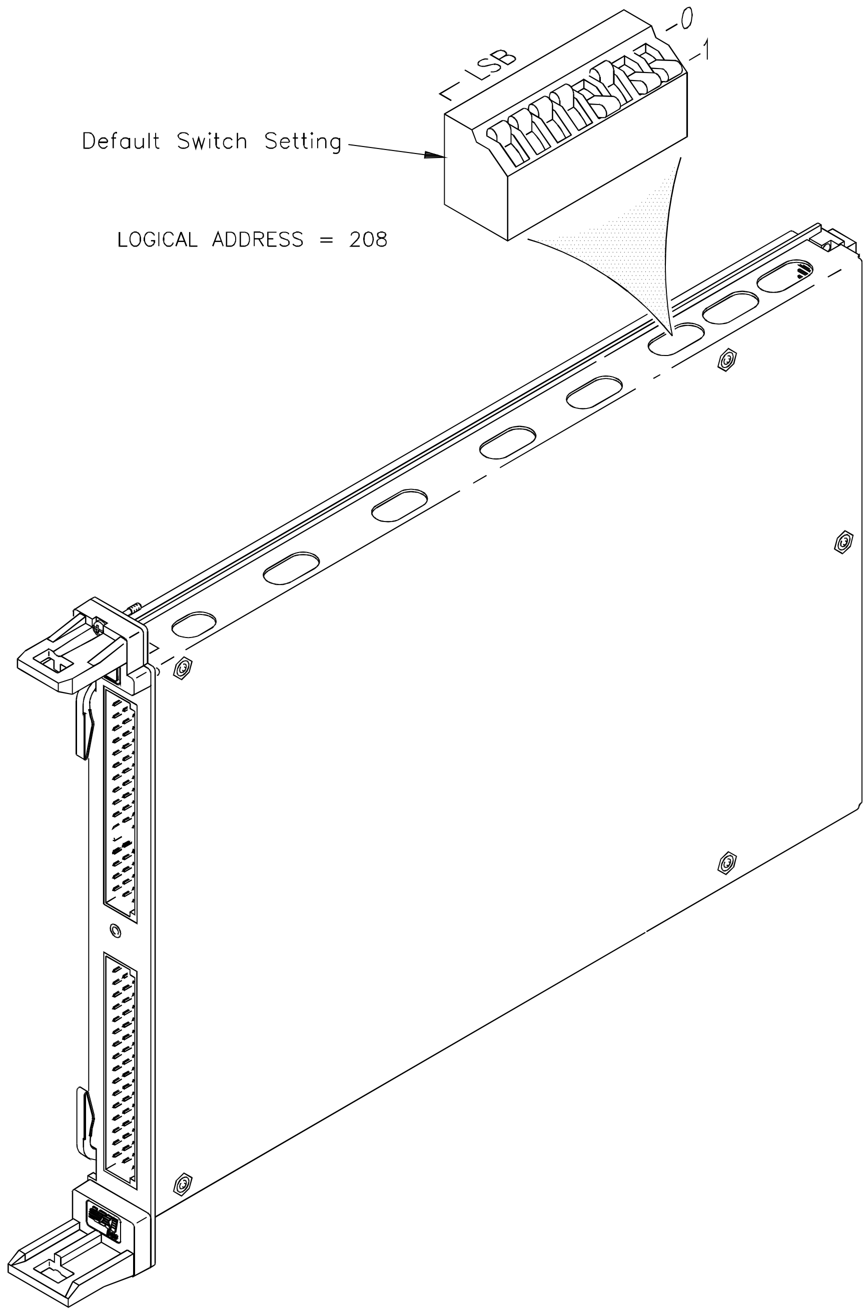
Switch/Jumper	Setting
Logical Address Switch	208
Input Protect Jumper	Protected
Flash Memory Protect Jumper	PROG

Note Setting the VXIbus Interrupt Level: The HP E1422 uses a default VXIbus interrupt level of 1. The default setting is made at power-on and after a *RST command. You can change the interrupt level by executing the DIAGnostic:INTerrupt[:LINe] command in your application program.

Setting the Logical Address Switch

Follow the next figure and ignore any switch numbering printed on the Logical Address switch. When installing more than one HP E1422 in a single VXIbus Mainframe, set each instrument to a different Logical Address.

Setting the Logical Address Switch

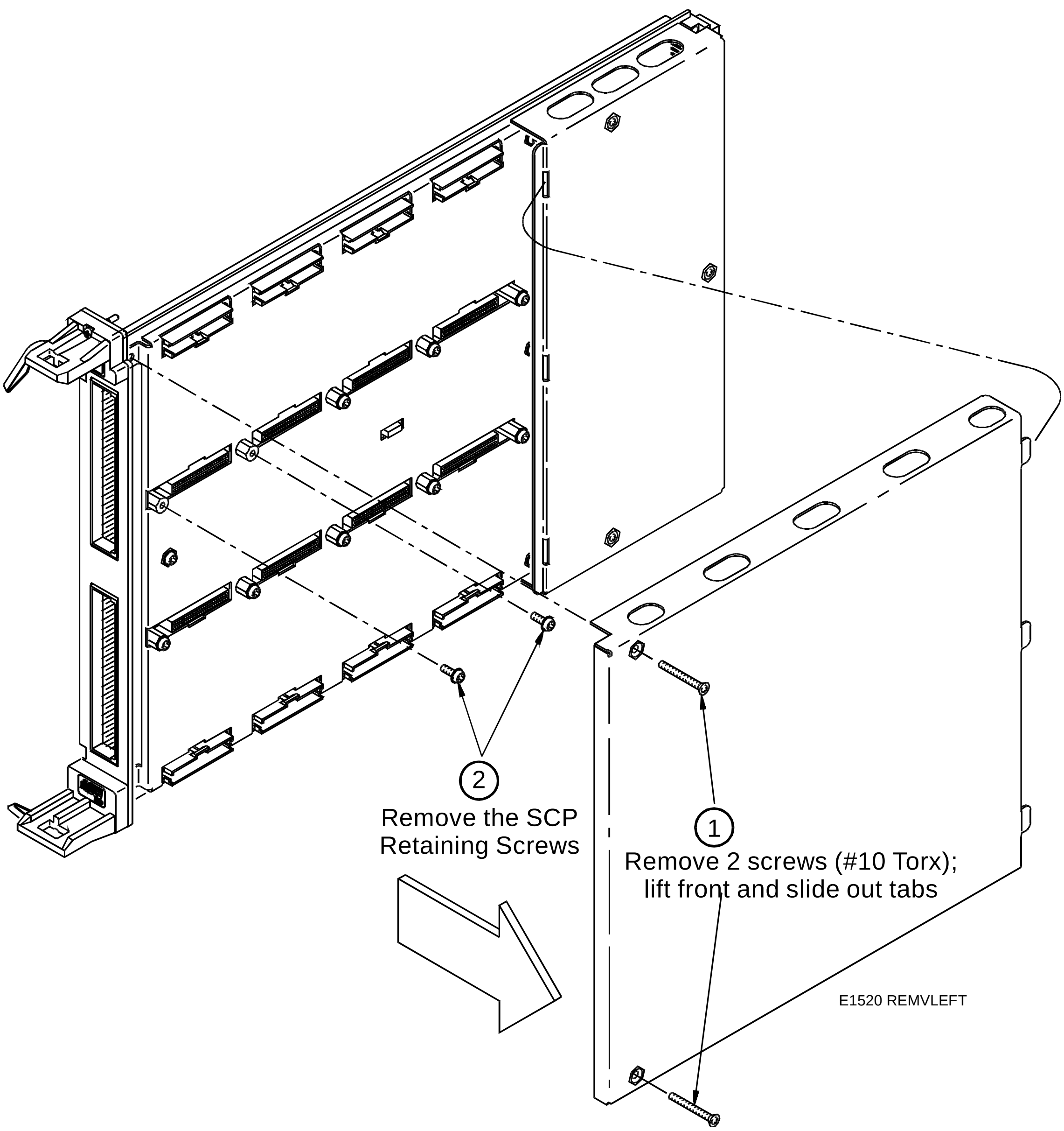


Installing Signal Conditioning Plug-ons

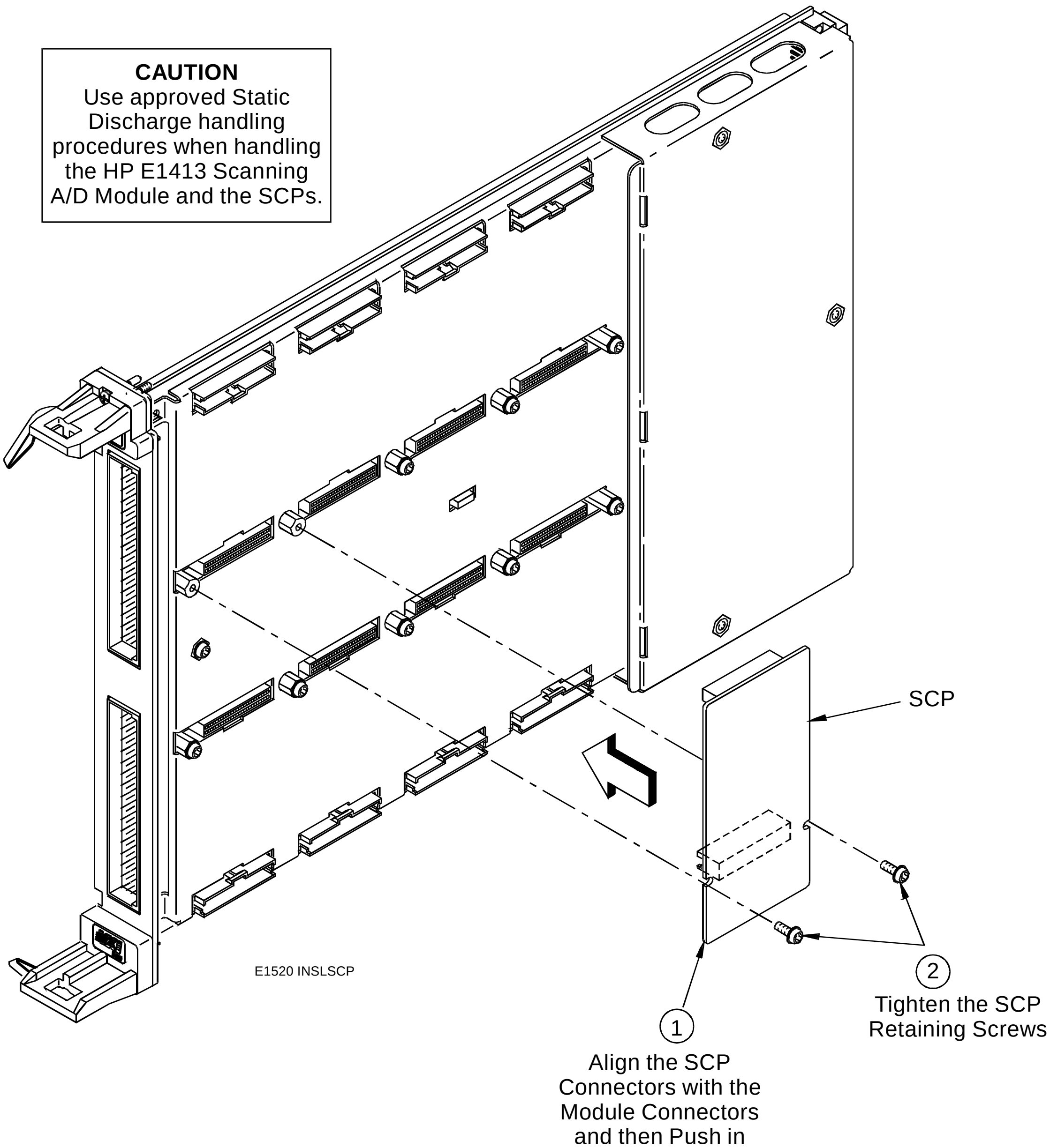
The following illustrations show the steps you'll use to install Signal Conditioning Modules. Before you install your SCPs, you should read the "Separating Digital and Analog SCP Signals" in Appendix D page 415.

Caution **Use approved Static Discharge Safe handling procedures anytime you have the covers removed from the HP E1422 or are handling SCPs.**

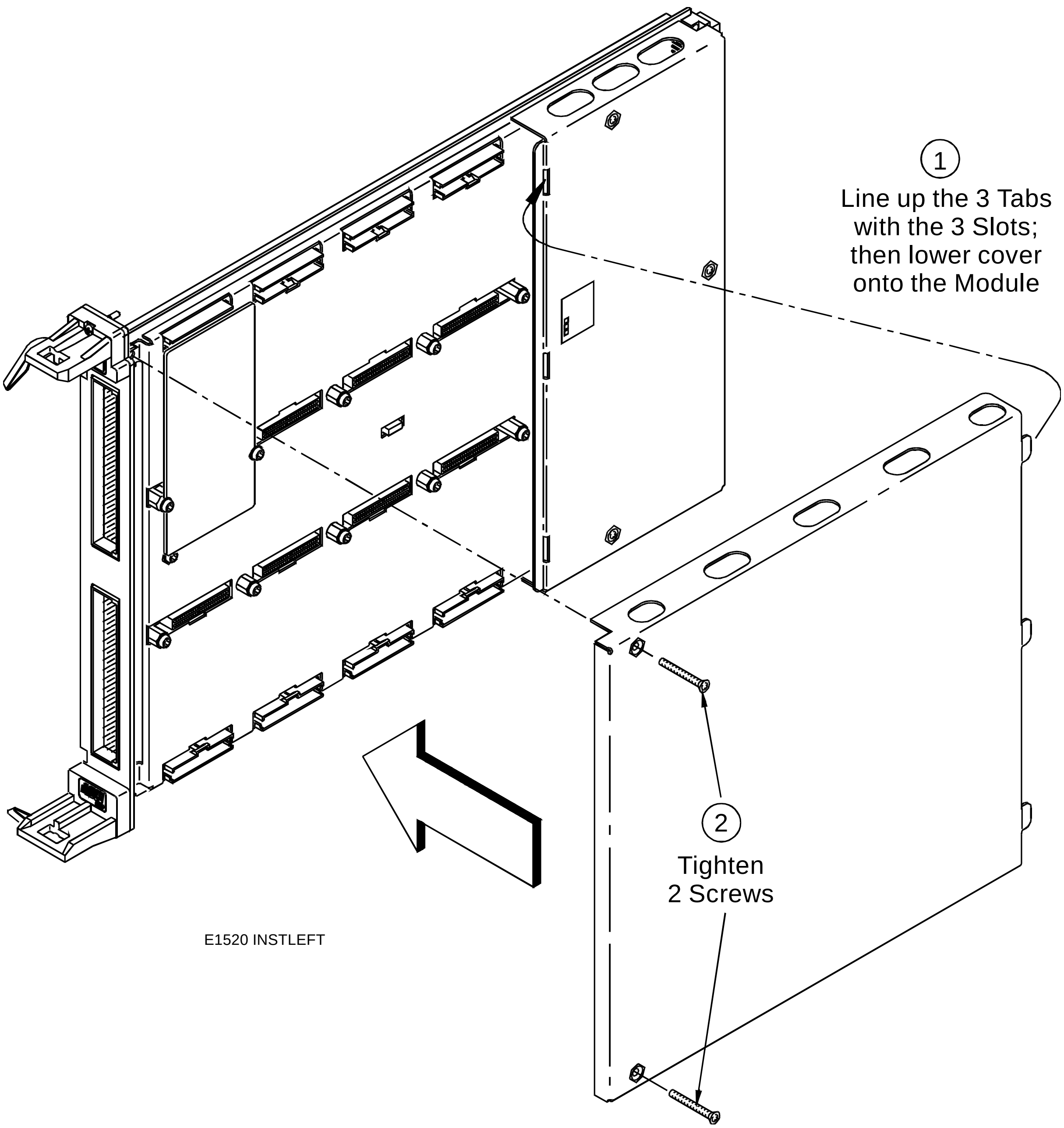
Installing SCPs: Step 1, Removing the Cover HP E1422



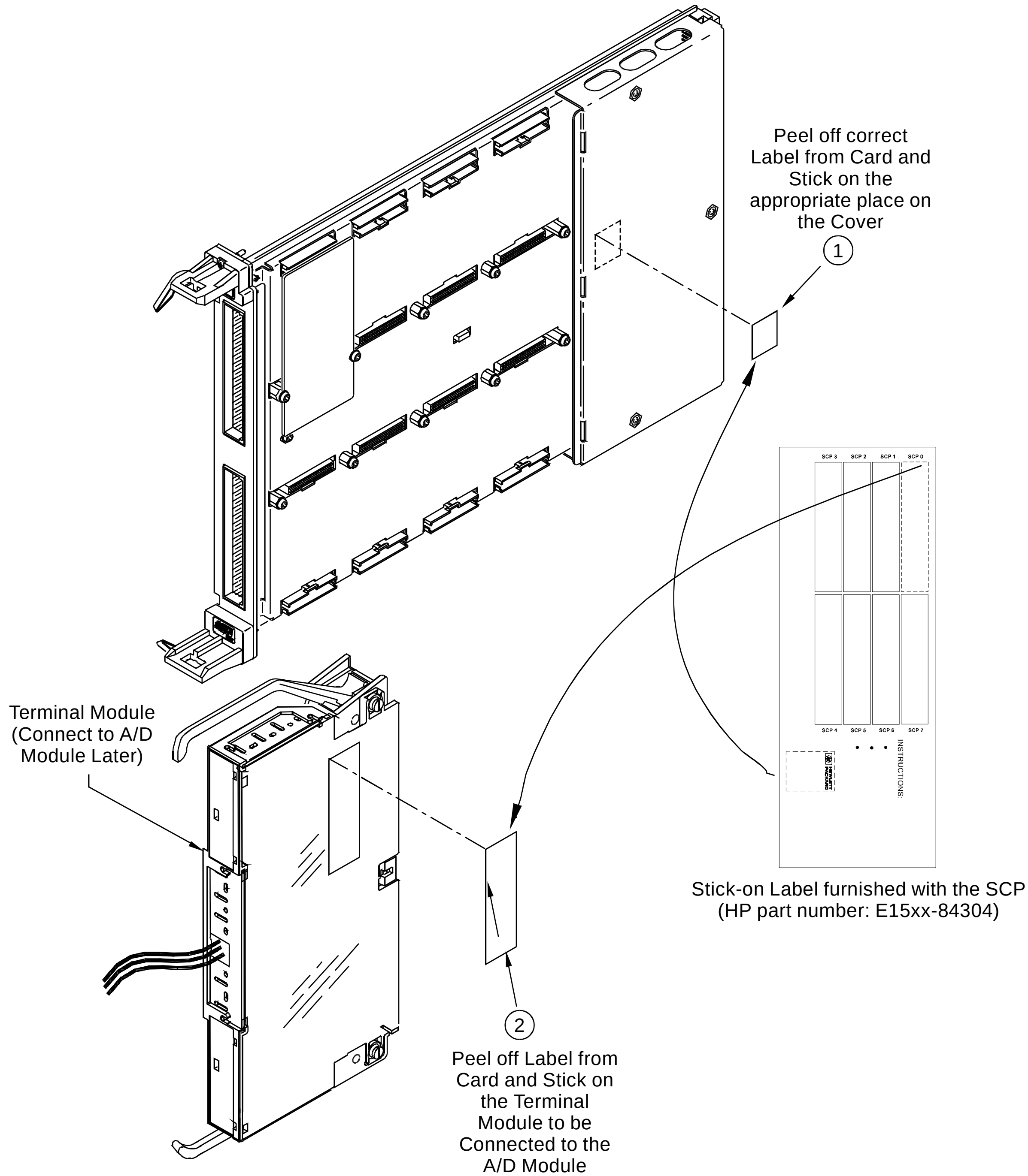
Installing SCPs: Step 2, Mounting an SCP



Installing SCPs: Step 3, Reinstalling the Cover HP E1422



Installing SCPs: Step 4, Labeling



Disabling the Input Protect Feature (optional)

Disabling the Input Protect feature voids the HP E1422's warranty. The Input Protect feature allows the HP E1422 to open all channel input relays if any input's voltage exceeds ±19 volts (±6 volts for digital I/O SCPs). This feature will help to protect the card's Signal Conditioning Plug-ons, input multiplexer, ranging amplifier, and A/D from destructive voltage levels. The level that trips the protection function has been set to provide a high probability of protection. The voltage level that is certain to cause damage is somewhat higher. **If in your application the importance of completing a measurement run outweighs the added risk of damage to your HP E1422, you may choose to disable the Input Protect feature.**

Voids Waranty! Disabling the Input Protection Feature voids the HP E1422's warranty.

To disable the Input Protection feature, locate and cut JM2202. Make a single cut in the jumper and bend the adjacent ends apart. See following illustration for location of JM2202.

Disabling Flash Memory Access (optional)

The Flash Memory Protect Jumper (JM2201) is shipped in the "PROG" position. We recommend that you leave the jumper in this position so that all of the calibration commands can function. Changing the jumper to the protect position will mean you won't be able to execute:

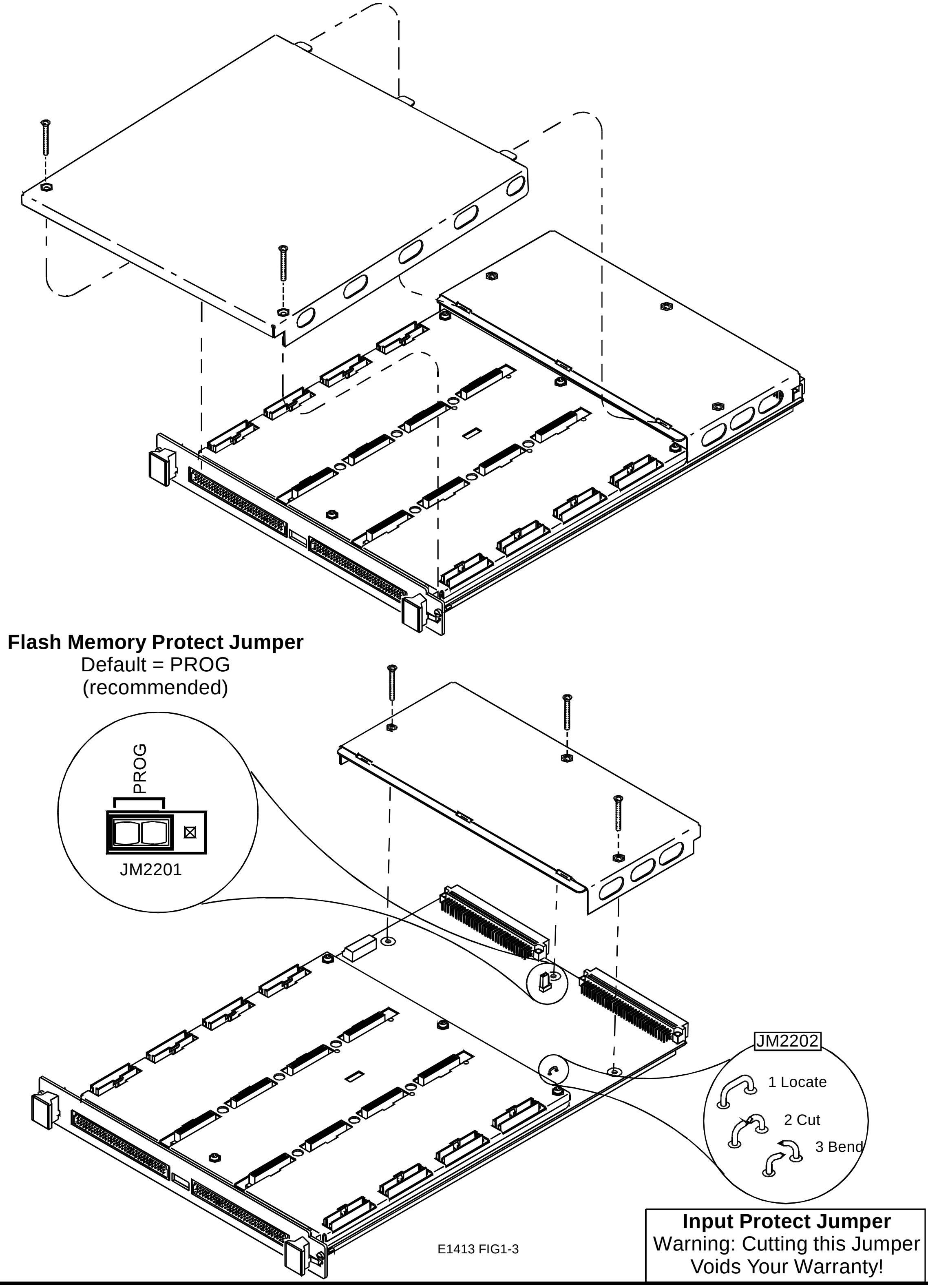
- The SCPI calibration command CAL:STORE ADC | TARE
- The register-based calibration commands STORECAL, and STORETAR
- Any application that installs firmware-updates or makes any other modification to Flash Memory through the A24 window.

With the jumper in the "PROG" position, you can completely calibrate one or more HP E1422s without removing them from the application system. An HP E1422 calibrated in its working environment will in general be better calibrated than if it were calibrated separate from its application system.

The multimeter you use during the periodic calibration cycle should be considered your calibration transfer standard. Have your Calibration Organization control unauthorized access to its calibration constants. See the *HP E1422 Service Manual* for complete information on HP E1422 periodic calibration.

If you must limit access to the HP E1422's calibration constants, you can place JM2201 in the protected position and cover the shield retaining screws with calibration stickers. See following illustration for location of JM2201.

Accessing and Locating JM2201 and JM2202 HP E1422



Installing the Module

Installation of the HP E1422 VXI module is covered in your HP Mainframe manual.

WARNING **All instruments within the VXI mainframe are grounded through the mainframe chassis. During installation, tighten the instrument's retaining screws to secure the instrument to the mainframe and to make the ground connection.**

WARNING **SHOCK HAZARD. Only qualified, service-trained personnel who are aware of the hazards involved should install, configure, or remove the VXI Module. Disconnect all power sources from the mainframe, the Terminal Modules, and installed modules before installing or removing a module.**

Instrument Drivers

Two driver types are supplied on the HP Universal Drivers CD that comes with your Instrument. There is a VXIplug&play driver which includes a front panel program and help file. In addition there is also a down-loadable driver for the HP E1406A Command Module. Follow the instructions that are presented by the CD setup program. Also view the readme.txt file provided with the VXIplug&play driver for possible update information.

About Example Programs

Examples on CD

All example programs mentioned by file name in this manual are available on the HP Universal Drivers CD supplied with your HP E1422. Again see the readme.txt file for the specific file locations of these examples.

Example Command Sequences

Where programming concepts are discussed in this manual, the commands to send to the HP E1422 are shown in the form of command sequences. These are not example programs because they are not written in any computer language. They are meant to show the HP E1422 SCPI commands in the sequence they should be sent. Where necessary these sequences include comments to describe program flow and control such as loop - end loop, and if - end if. See "Example SCPI Command Sequence" on page 131. for an example. For VXIplug&play users, there is an "Example VXIplug&play Driver Function Sequence" on page 132.

Typical Example program

The Verify program (file name verif.cpp) is printed below to show a typical VXIplug&play program for the HP E1422.

Verifying a Successful Configuration

An example 'C' program source is shown on the following pages. This program is included on your HP Universal Drivers CD that comes with your HP E1422A (file name *verif.cpp*). The program uses the *IDN? query command to verify the HP E1422 is operational and responding to commands. The program also has an error checking function (*check()*). It is important to include an instrument error checking routine in your programs, particularly your first trial programs so you get instant feedback while you are learning about the HP E1422. Compile this program according to the plug&play help file (hpe1422.hlp) topics "Introduction to Programming"→"Compiling and Linking Programs Using Integrated Environments".

```
/*****************************************************************************/
/* verif.cpp                                                                 */
/*                                                                           */
/*   This example program uses an instrument Plug&Play driver, the VISA I/O */
/*   Library and Visual C/C++.and an HP VXI Command Module                   */
/*                                                                           */
/*   The program resets the module and reads the module's ID string.         */
/*                                                                           */
/*   Program should be compiled in the 'large' memory model.                 */
/*                                                                           */
/*   This program requires the following header and library files:          */
/*                                                                           */
/*      e1422.h - your instrument Plug&Play header file                      */
/*      visatype.h - header file to define for Visa types                    */
/*      vpptype.h - header file to define Plug&Play types                    */
/*      hpe1422.lib - library file                                           */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <e1422.h>    /* include the driver header file */

 /* GPIB-VXI addressing (0 is the interface number, 208 is the */
 /* instrument logical address, INSTR is a VISA resource type) */
#define INSTR_ADDRESS "GPIB-VXI0::208::INSTR"

ViSession  addr;
ViStatus  errStatus;

 /* Function Prototypes */
void main (void);  /* Main function */
void rst_inst(void); /* Resets the instrument and sends a device clear */
void reads_instrument_id(void); /* reads instrument software revision */
void check (ViSession addr, ViStatus errStatus); /* checks module errors */

/*****************************************************************************/
void main (void) /* Main function */
{
 ViChar  err_message[256];

 /* Set the session and status variables */
 #if defined(__BORLANDC__) && !defined(__WIN32__)
     _InitEasyWin();
 #endif
```

```
 /* open device session and reset the instrument; check if successful */
 errStatus = hpe1422_init(INSTR_ADDRESS,0,0,&addr);
     if( VI_SUCCESS > errStatus)
     {
         hpe1422_error_message( addr, errStatus, err_message);
         printf("Unable to open %s\n", INSTR_ADDRESS);
         printf("hpe1422_init() returned error message %s\n", err_message);
         return;
     }

 rst_inst();     /* Resets the instrument and sends a device clear */

 reads_instrument_id();  /* Reads instrument software revision */

 read_integer_long_data();   /* Reads integer and long data */

 /* close the device session */
 hpe1422_close(addr);
}

/*****************************************************************************/
void rst_inst(void)
 /* Function to set the interface timeout period, resets the instrument, */
 /* waits for completion of reset, and sends a device clear to enable */
 /* the instrument to receive a new command */
{
    ViInt32 result;

 /* set timeout to allow completion of reset */
 errStatus = hpe1422_timeOut(sessn, 5000);
 check(sessn, errStatus);

 /* reset the instrument */
 errStatus = hpe1422_reset(sessn);
 check(sessn, errStatus);

 /* wait for completion of *RST */
 errStatus = hpe1422_cmdInt32_Q(sessn,"*OPC?",&result);
 check(sessn, errStatus);

 /* send a device clear to enable new commands to be sent to the instrument */
 errStatus = hpe1422_dcl(sessn);
 check(sessn, errStatus);
}

/*****************************************************************************/
void reads_instrument_id(void)
 /* Function uses a hpe1422__revision_query to read the software revision */
 /* string. */
{
 ViChar  driver_rev[256];
 ViChar  instr_rev[256];

     /* Query the instrument for its firmware revision */
 errStatus = hpe1422_revision_query(addr,  driver_rev, instr_rev);

     /* Print the results */
 printf("The instrument driver’s revision is %s\n", driver_rev);
 printf("The instrument’s firmware revision is %s\n", instr_rev);
}

/*****************************************************************************/
/*****************************************************************************/
        /* error checking routine */
```

```
void check (ViSession addr, ViStatus errStatus)
{
 ViInt32 err_code;
 ViChar err_message[256];

 if(VI_SUCCESS > errStatus)
 {
  hpe1422_dcl(addr);    /* send a device clear */
  if(hpe1422_INSTR_ERROR_DETECTED == errStatus)
  {
                        /* query the instrument */
    hpe1422_error_query( addr, &err_code, err_message);
                        /* display the error */
    printf("Instrument Error : %ld, %s\n", err_code, err_message);
  }
  else
  {
                        /* query the instrument */
   hpe1422_error_message( addr, errStatus, err_message);
                        /* display the error */
   printf("Driver Error : %ld, %s\n", errStatus, err_message);
  }
 hpe1422_reset(addr);                  /* reset the instrument */
 hpe1422_close(addr);                  /* close the instrument handle */
 exit(1);
}

        return;
}
```

Chapter 2
Field Wiring

About This Chapter

This chapter shows how to plan and connect field wiring to the HP E1422's Terminal Module. The chapter explains proper connection of analog signals to the HP E1422, both two-wire voltage type and four-wire resistance type measurements. Connections for other measurement types (e.g., strain using the Bridge Completion SCPs) refer to the specific SCP manual. Chapter contents include:

Planning Your Wiring Layout

The first point to understand is that the HP E1422 makes no assumptions about the relationship between Signal Conditioning Plug-on (SCP) function and the position in the HP E1422 that it can occupy. You can put any type of SCP into any SCP position. There are, however, some factors you should consider when planning what mix of SCPs should be installed in each of your HP E1422s. The following discussions will help you understand these factors.

SCP Positions and Channel Numbers

The HP E1422 has a fixed relationship between Signal Conditioning Plug-on positions and the channels they connect to. Each of the eight SCP positions can connect to eight channels. Figure 2-1 shows the channel number to SCP relationship.

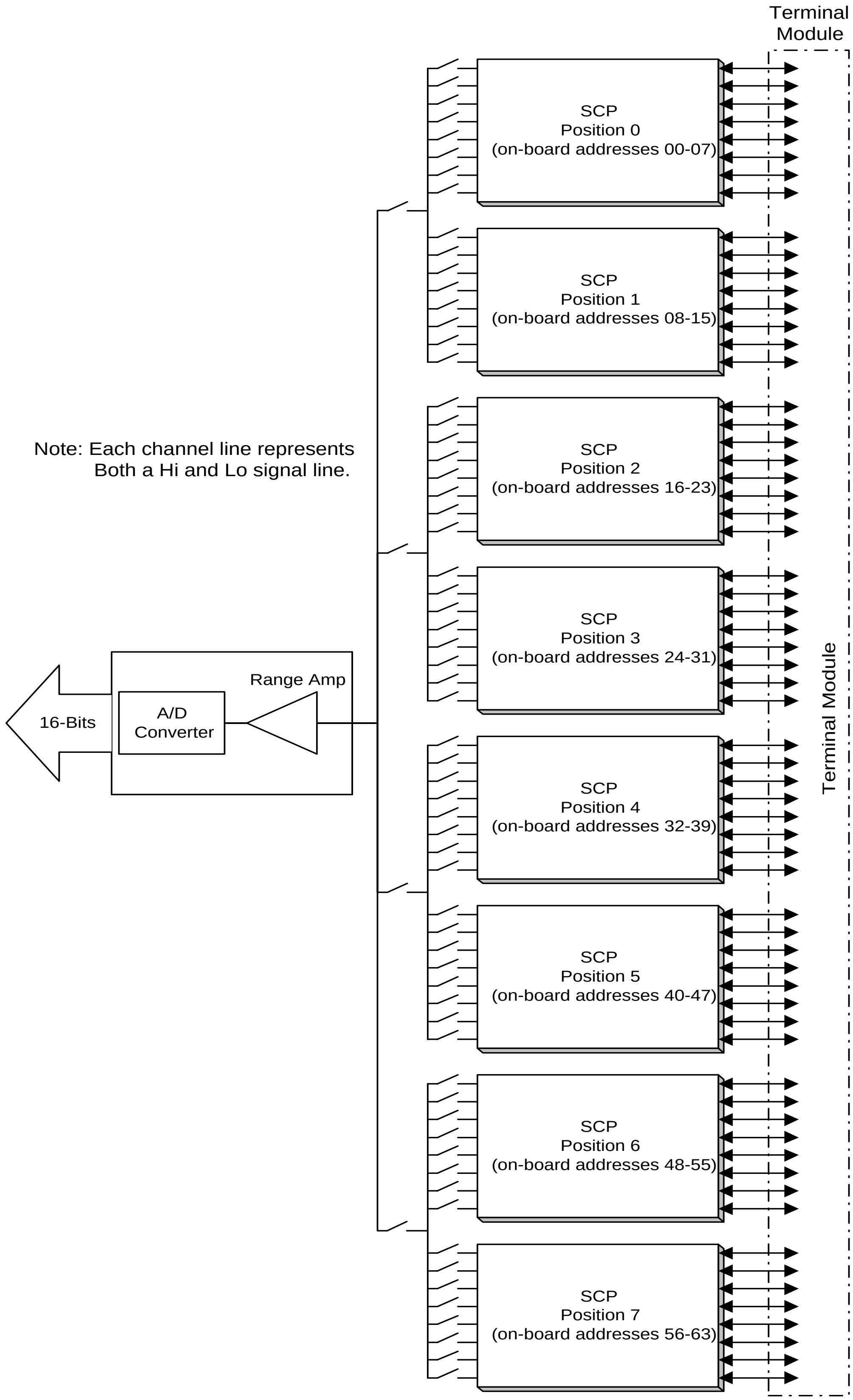


Figure 2-1. Channel Numbers at SCP Positions

Sense SCPs and Output SCPs

Some SCPs provide input signal conditioning (sense SCPs such as filters and amplifiers) while others provide stimulus to your measurement circuit (output SCPs such as current sources and strain bridge completion). In general, channels at output SCP positions are not used for external signal sensing but are paired with channels of a sense SCP. Two points to remember about mixing output and sense SCPs:

1. Paired SCPs (an output and a sense SCP) may reside in separate HP E1422s. SCP outputs are adjusted by *CAL? to be within a specific limit. The Engineering Unit (EU) conversion used for a sense channel will assume the calibrated value for the output channel.

2. Output SCPs while providing stimulus to your measurement circuit reduce the number of external sense channels available to your HP E1422.

Figure 2-2 illustrates an example of "pairing" output SCP channels with sense SCP channels (in this example, four-wire resistance measurements).

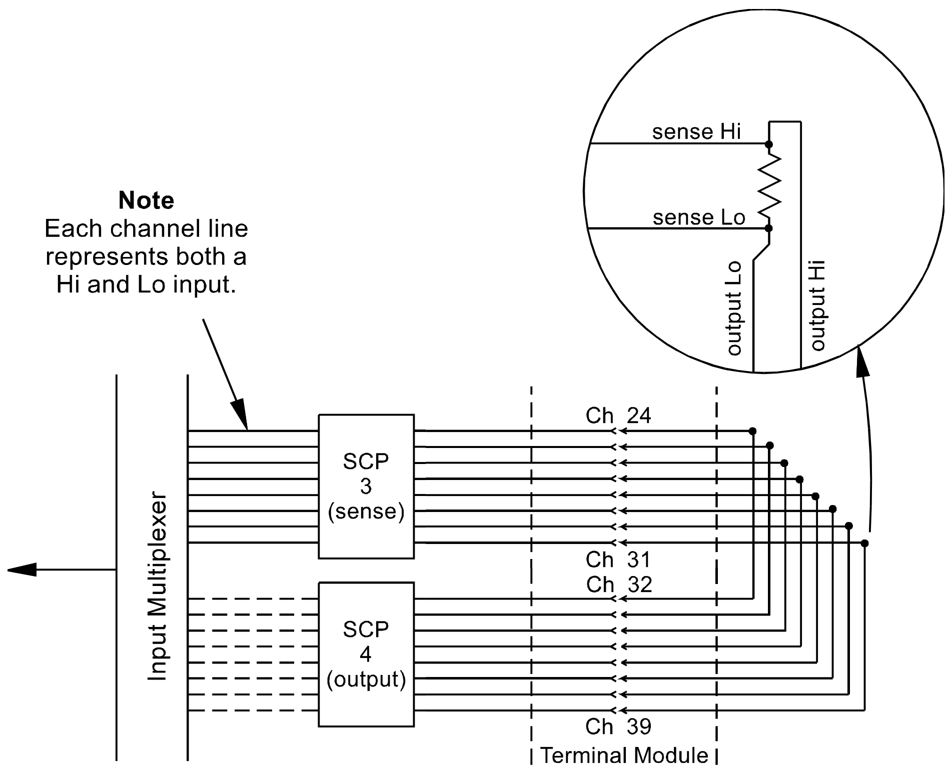


Figure 2-2. Pairing Output and Sense SCP Channels

Planning for Thermocouple Measurements

Using either the Screw Terminal or Spring Terminal Modules you can wire your thermocouples and your thermocouple reference temperature sensor to any of the HP E1422's channels. When you execute your scan list, you only have to make sure that the reference temperature sensor is specified in the channel sequence before any of the associated thermocouple channels.

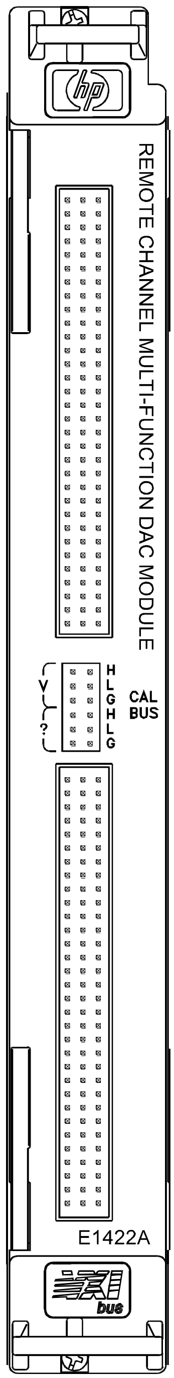
External wiring and connections to the HP E1422 are made using Terminal Modules (see Figures 2-4 through 2-6).

Note The isothermal reference temperature measurement made by an HP E1422 applies only to thermocouple measurements made by that instrument. In systems with multiple HP E1422s, each instrument must make its own reference measurements. The reference measurement made by one HP E1422 can not be used to compensate thermocouple measurements made by another HP E1422.

Note To make good low-noise measurements you must use shielded wiring from the device under test to the Terminal Module at the HP E1422. The shield must be continuous through any wiring panels or isothermal reference connector blocks and must be grounded at a single point to prevent ground loops. See "Preferred Measurement Connections" later in this section and "Wiring and Noise Reduction Methods" on page 415.

Faceplate Connector Pin-Signal Lists

Figure 2-3 shows the Faceplate Connector Pin Signal List for the HP E1422.



Count From Top				Count From Bottom
1	GND	H00	L00	32
2	GND	H01	L01	31
3	GND	H02	L02	30
4	GND	H03	L03	29
5	GND	H04	L04	28
6	G0	H05	L05	27
7	G1	H06	L06	26
8	H08	H07	L07	25
9	L08	H09	L09	24
10	H11	H10	L10	23
11	L11	H12	L12	22
12	H14	H13	L13	21
13	L14	H15	L15	20
14	H17	H16	L16	19
15	L17	H18	L18	18
16	H20	H19	L19	17
17	L20	H21	L21	16
18	H23	H22	L22	15
19	L23	H24	L24	14
20	G2	H25	L25	13
21	H27	H26	L26	12
22	L27	H28	L28	11
23	H30	H29	L29	10
24	L30	H31	L31	9
25	G3	G3	G3	8
26	G3	G3	G3	7
27	GND	GND	GND	6
28	GND	HOHM	LOHM	5
29	GND	HCAL	LCAL	4
30	GND	GND	GND	3
31	GND	H_I	L_I	2
32	GND	GND	GND	1

1	HCAL	HCAL	6
2	LCAL	LCAL	5
3	GND	GND	4
4	HOHM	HOHM	3
5	LOHM	LOHM	2
6	GND	GND	1

1	GND	GND	GND	32
2	GND	TRG	GND	31
3	GND	GND	GND	30
4	TST-A	SYSF	GND	29
5	GND	GND	GND	28
6	G4	G4	G4	27
7	G4	G4	G4	26
8	G4	G4	G4	25
9	L33	H32	L32	24
10	H33	H34	L34	23
11	L36	H35	L35	22
12	H36	H37	L37	21
13	G5	H38	L38	20
14	L40	H39	L39	19
15	H40	H41	L41	18
16	L43	H42	L42	17
17	H43	H44	L44	16
18	L46	H45	L45	15
19	H46	H47	L47	14
20	L49	H48	L48	13
21	H49	H50	L50	12
22	L52	H51	L51	11
23	H52	H53	L53	10
24	L55	H54	L54	9
25	H55	H56	L56	8
26	G6	H57	L57	7
27	G7	H58	L58	6
28	GND	H59	L59	5
29	GND	H60	L60	4
30	GND	H61	L61	3
31	GND	H62	L62	2
32	GND	H63	L63	1

Figure 2-3. HP E1422A Faceplate Connector Pin Signals

Optional Terminal and Connector Modules

The HP E1422 is comprised of the main A/D module and optionally, a Connector or Terminal Module. The Option 001 Connector Module provides 16 RJ-45 jacks to allow easy connection of the HP E1422A to Remote Signal Conditioning Units (RSCUs) like the HP E1529A Remote Strain Conditioning Unit. Optional conventional terminal modules include a Terminal Module with screw clamped terminal blocks (Option 011), and another with spring clamped terminal blocks (Option 013).

The Spring Terminal Module and Screw Terminal Module provide:

- Terminal connections to field wiring.
  -- Allows a mix of direct field wiring with some connections to Remote Signal Conditioning Units
- Strain relief for the wiring bundle.
- Reference junction temperature sensing for thermocouple measurements.
- Ground to Guard connections for each channel.

The RJ-45 Connector Module provides:

- Easy mass terminated plug-in connection to HP Remote Signal Conditioning Units (RSCUs).
- Allows some direct analog or digital field wiring to be connected to RJ-45 modular plugs that are then plugged into the Connector Module.
- **Note:** Since the RJ-45 Connector Module was designed for connection to RSCUs, it doesn't provide an on-board isothermal reference thermistor or connection to the HP E1422A's on-board current source. This means that the RJ-45 Terminal Module is not suitable for direct connection to thermocouples.

The SCPs and Terminal Module

Each SCP includes a set of labels to map that SCP's channels to the Terminal Module's terminal blocks. See "Installing SCPs: Step 4, Labeling" on page 26.

Terminal Module Layout

Figures 2-4 through 2-6 show the layout and feature location of the Terminal Modules available for the HP E1422A.

WARNING **When handling user wiring connected to the Terminal Module, consider the highest voltage present accessible on any terminal. Use only wire with an insulation rating greater than the highest voltage which will be present on the Terminal Module. Do not touch any circuit element connected to the Terminal Module if any other connector to the Terminal Module is energized to more than 30VACRMS or 60VDC.**

The RJ-45 Connector Module

Figure 2-4 shows the HP E1422A Option 001 RJ-45 Connector Module with connector pin numbering.

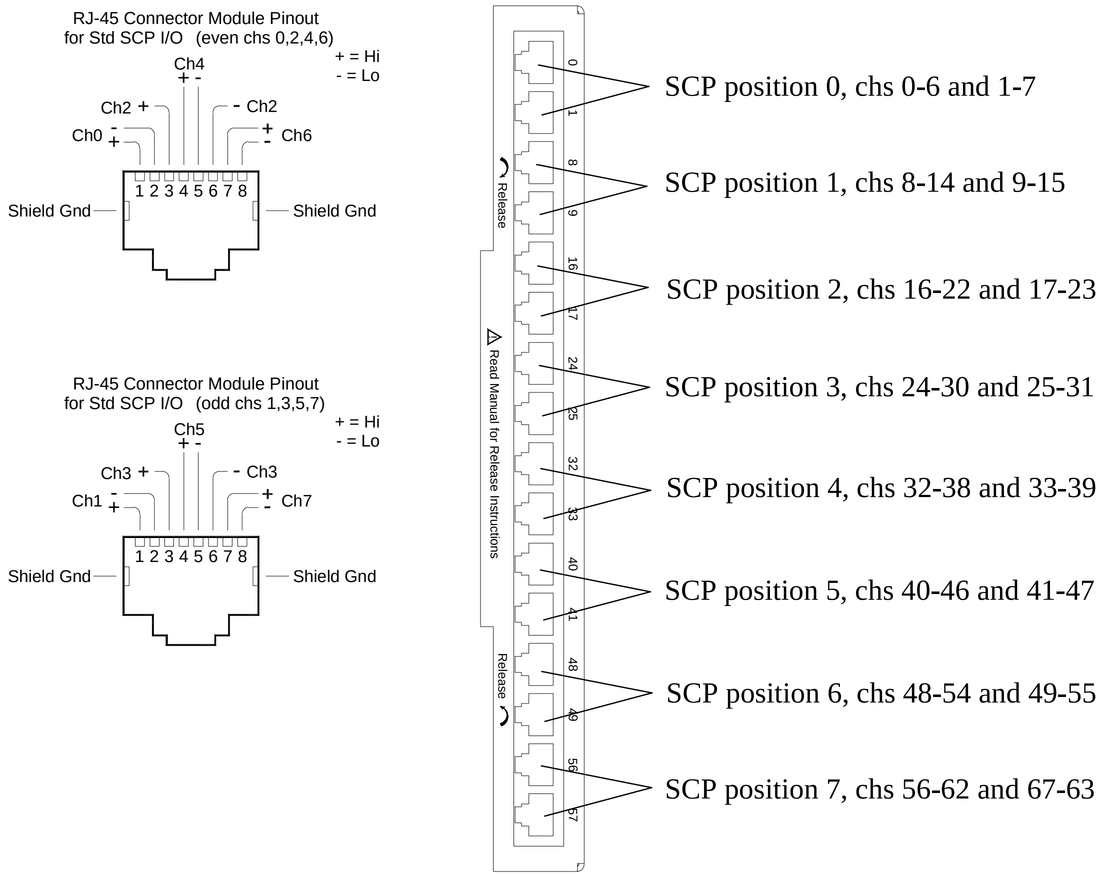


Figure 2-4. RJ-45 Connector Module and Pin-out

Spring Terminal Module Layout

Figure 2-5 shows the HP E1422A Option 013 Spring Terminal Module features and jumper locations.

Caution **WIRING THE TERMINAL MODULES. When wiring to the terminal connectors on the screw clamp and spring clamp Terminal Module, be sure not to exceed a 5mm strip back of insulation to prevent the possibility of shorting to other wiring on adjacent terminals.**

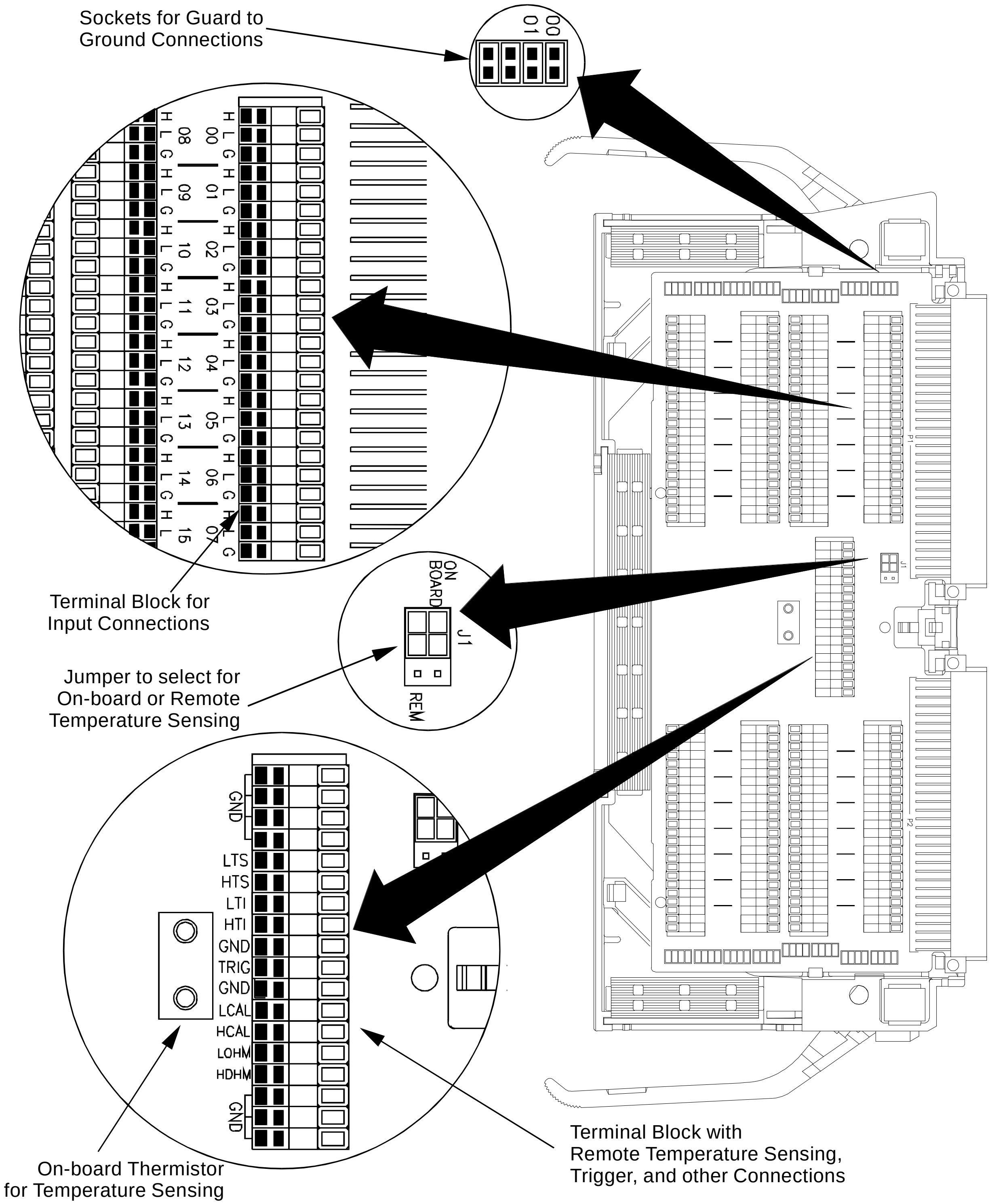


Figure 2-5. HP E1422A Spring Terminal Module

Screw Terminal Module Layout

Figure 2-6 shows the HP E1422A Option 011 Screw Terminal Module features and jumper locations.

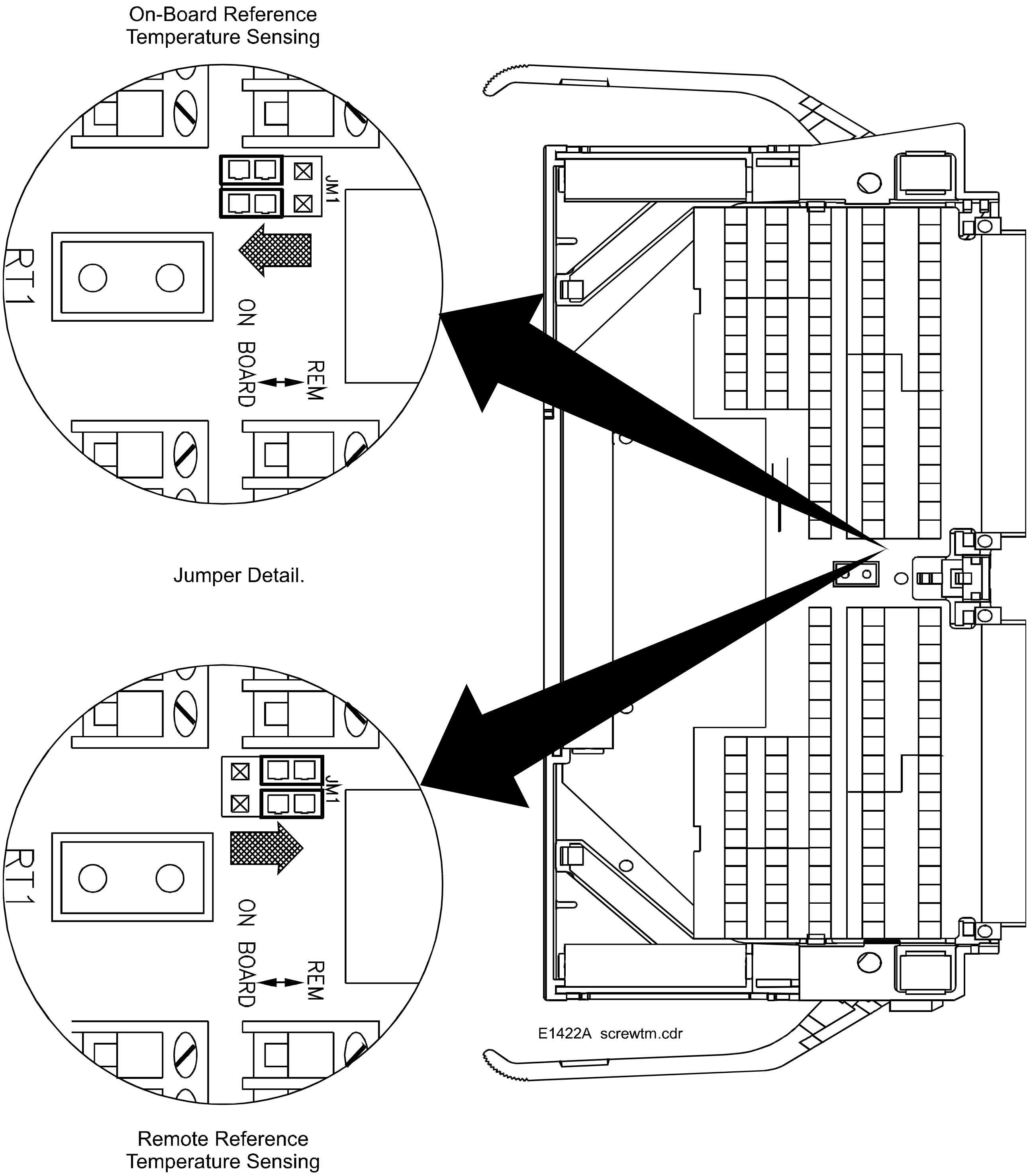


Figure 2-6. HP E1422A Screw Terminal Module

Reference Temperature Sensing with the HP E1422

The Screw Terminal and Spring Terminal Modules provides an on-board thermistor for sensing isothermal reference temperature of the terminal blocks. Also provided is a jumper set (J1 in Figures 2-7 and 2-8) to route the HP E1422's on-board current source to a thermistor or RTD on a remote isothermal reference block. Figure 2-7 and Figure 2-8 show connections for both local and remote sensing. See "Connecting the On-board Thermistor" on page 47. for location of J1.

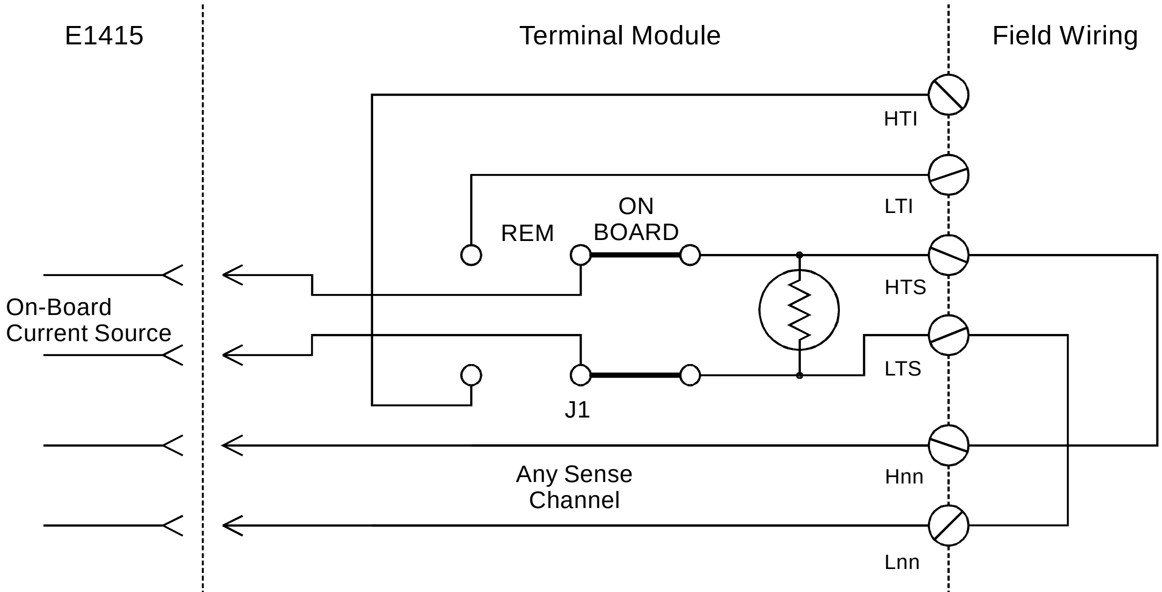


Figure 2-7. On-Board Thermistor Connection

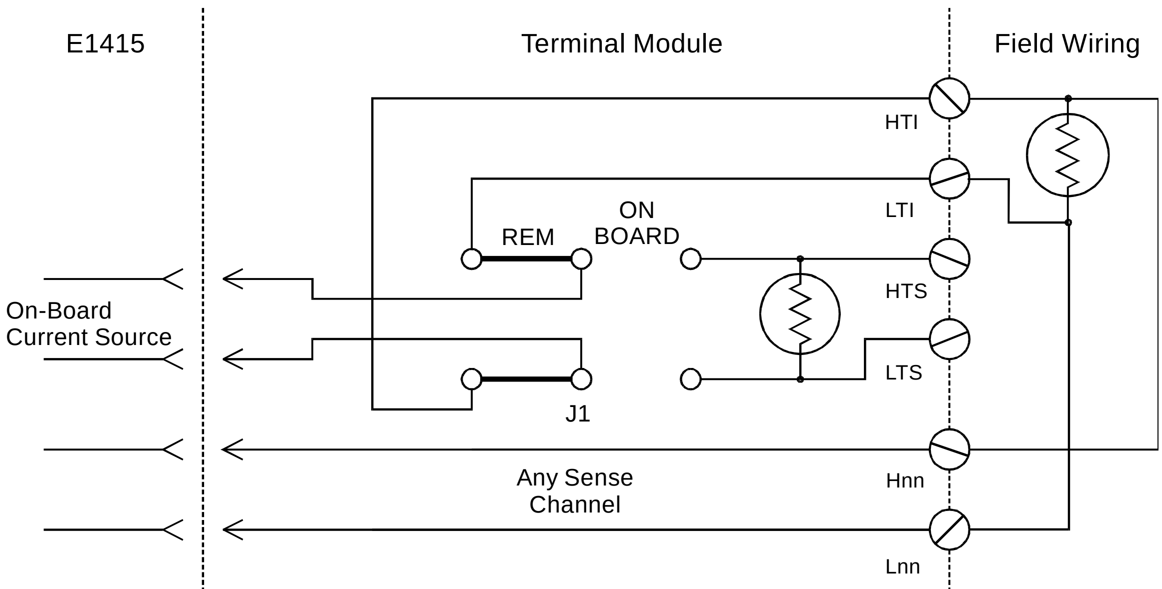


Figure 2-8. Remote Thermistor or RTD Connections

Terminal Module Considerations for TC Measurements

The isothermal characteristics of the HP E1422 Terminal Module are crucial for good TC readings and can be affected by any of the following factors:

1. The clear plastic cover must be on the Terminal Module.
2. The thin white mylar thermal barrier must be inserted over the Terminal Module connector. This prevents airflow from the HP E1422 A/D Module into the Terminal Module.
3. The Terminal Module must also be in a fairly stable temperature environment, and it is best to minimize the temperature gradient between the HP E1422 module and the Terminal Module.
4. The VXI mainframe cooling fan filters must be clean and there should be as much clear space in front of the fan intakes as possible.
5. Recirculating warm air inside a closed rack cabinet can cause a problem if the Terminal Module is suspended into ambient air that is significantly warmer or cooler. If the mainframe recess is mounted in a rack with both front and rear doors, closing both doors helps keep the entire HP E1422 at a uniform temperature. If there is no front door, try opening the back door.
6. HP recommends that the cooling fan switch on the back of the of an HP E1401 Mainframe is in the "High" position. The normal variable speed cooling fan control can make the internal HP E1422 module temperature cycle up and down, which affects the amplifiers with these uV level signals.

Preferred Measurement Connections

IMPORTANT!

For any A/D Module to scan channels at high speeds, it must use a very short sample period (<10μsecond for the HP E1422). If significant normal mode noise is presented to its inputs, that noise will be part of the measurement. To make quiet, accurate measurements in electrically noisy environments, use properly connected shielded wiring between the A/D and the device under test. Figure 2-9 shows recommended connections for powered transducers, thermocouples, and resistance transducers. (See Appendix D page 415 for more information on Wiring Techniques).

Notes

1. Try to install Analog SCPs relative to Digital I/O as shown in "Separating Digital and Analog Signals" in Appendix .
2. Use individually shielded, twisted-pair wiring for each channel.
3. Connect the shield of each wiring pair to the corresponding Guard (G) terminal on the Terminal Module (see Figure 2-10 for schematic of Guard to Ground circuitry on the Terminal Module).
4. The Terminal Module is shipped with the Ground-Guard (GND-GRD) shorting jumper installed for each channel. These may be left installed or removed (see Figure 2-11 to remove the jumper), dependent on the following conditions:
   a. **Grounded Transducer with shield connected to ground at the transducer:** Low frequency ground loops (DC and/or 50/60Hz) can result if the shield is also grounded at the Terminal Module end. To prevent this, remove the GND-GRD jumper for that channel (Figure 2-9 A/C).
   b. **Floating Transducer with shield connected to the transducer at the source:** In this case, the best performance will most likely be achieved by leaving the GND-GRD jumper in place (Figure 2-9 B/D).
3. In general, the GND-GRD jumper can be left in place unless it is necessary to remove to break low frequency (below 1 kHz) ground loops.
4. Use good quality foil or braided shield signal cable.
5. Route signal leads as far as possible from the sources of greatest noise.
6. In general, don't connect Hi or Lo to Guard or Ground at the HP E1422.
7. It is best if there is a D.C. path somewhere in the system from Hi or Lo to Guard/Ground.
8. The impedance from Hi to Guard/Ground should be the same as from Lo to Guard/Ground (balanced).
9. Since each system is different, don't be afraid to experiment using the suggestions presented here until you find an acceptable noise level.

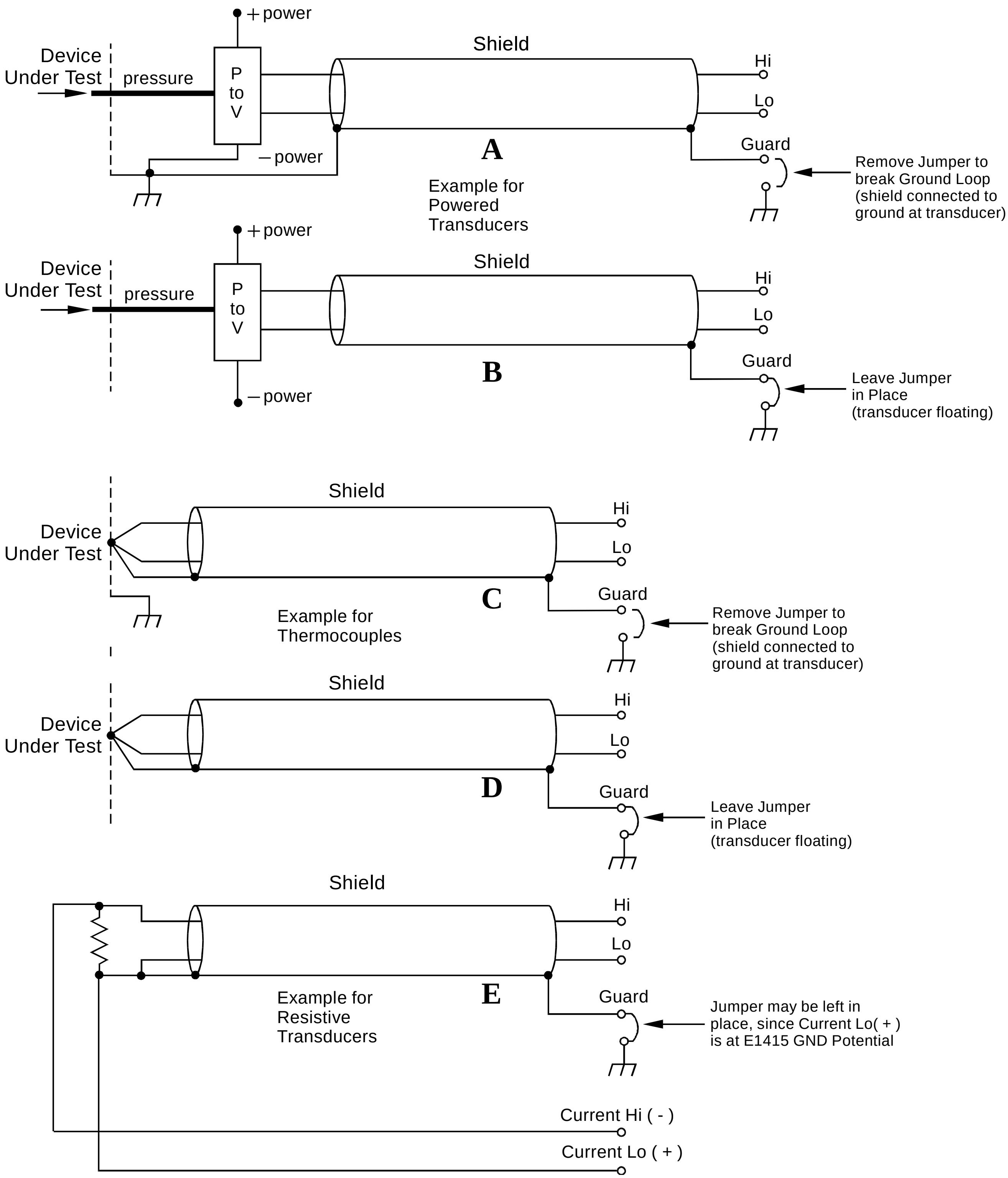


Figure 2-9. Preferred Signal Connections

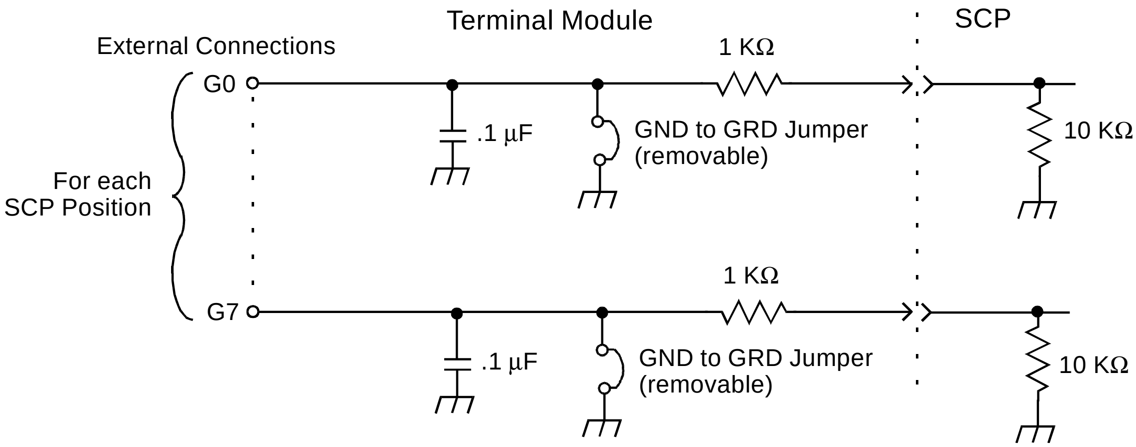


Figure 2-10. GRD/GND Circuitry on Terminal Module

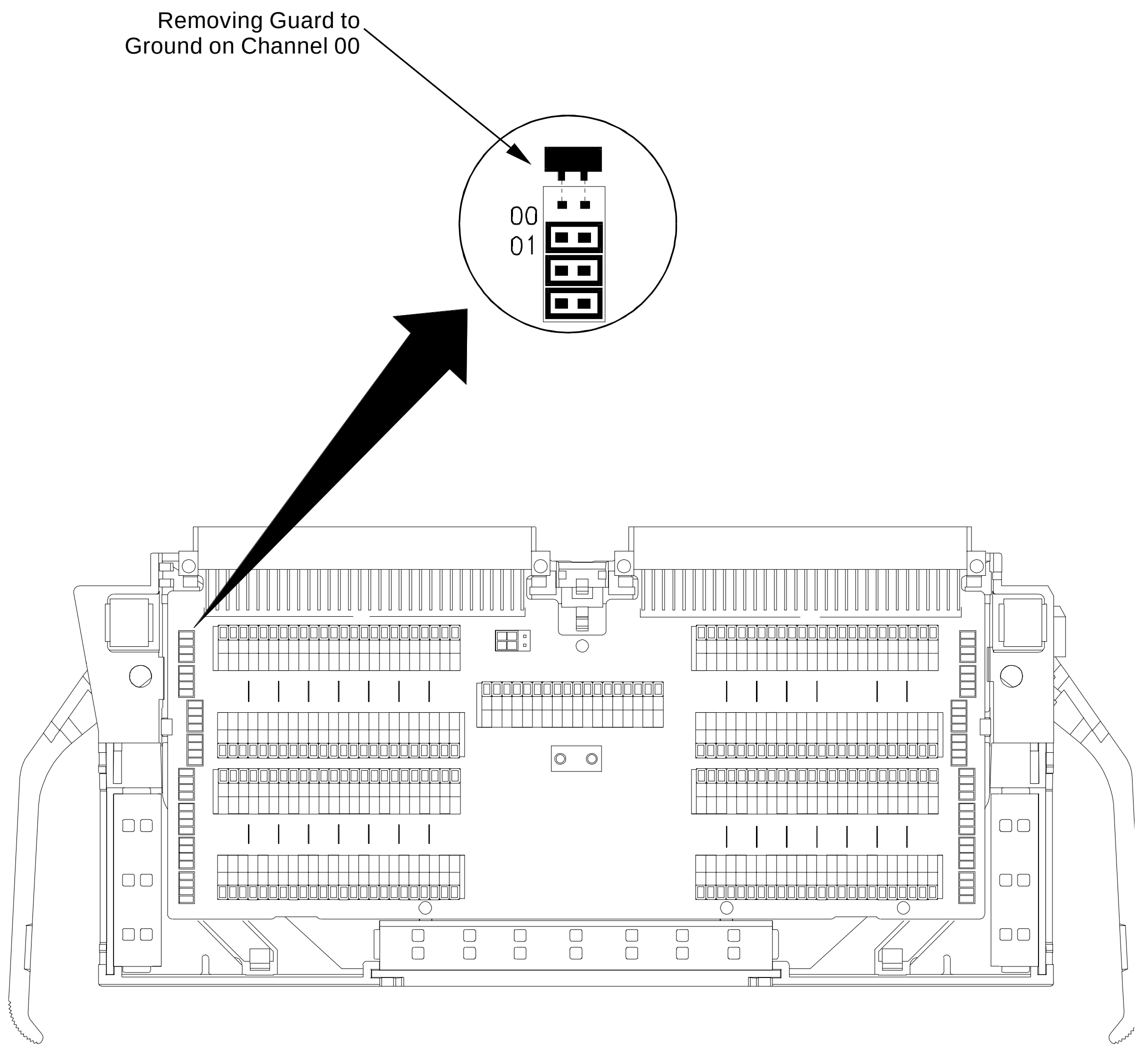


Figure 2-11. Grounding the Guard Terminals

Connecting the On-board Thermistor

The following figures show how to use the HP E1422 to make temperature measurements using the on-board Thermistor or a remote reference sensor. The Thermistor is used for reference junction temperature sensing for thermocouple measurements. Figure 2-12 shows the configuration for the HP E1422A's Spring Terminal Module, Figure 2-6 shows the configuration for the Screw Terminal Module. See "Reference Temperature Sensing with the HP E1422" on page 42. for a schematic diagram of the reference connections.

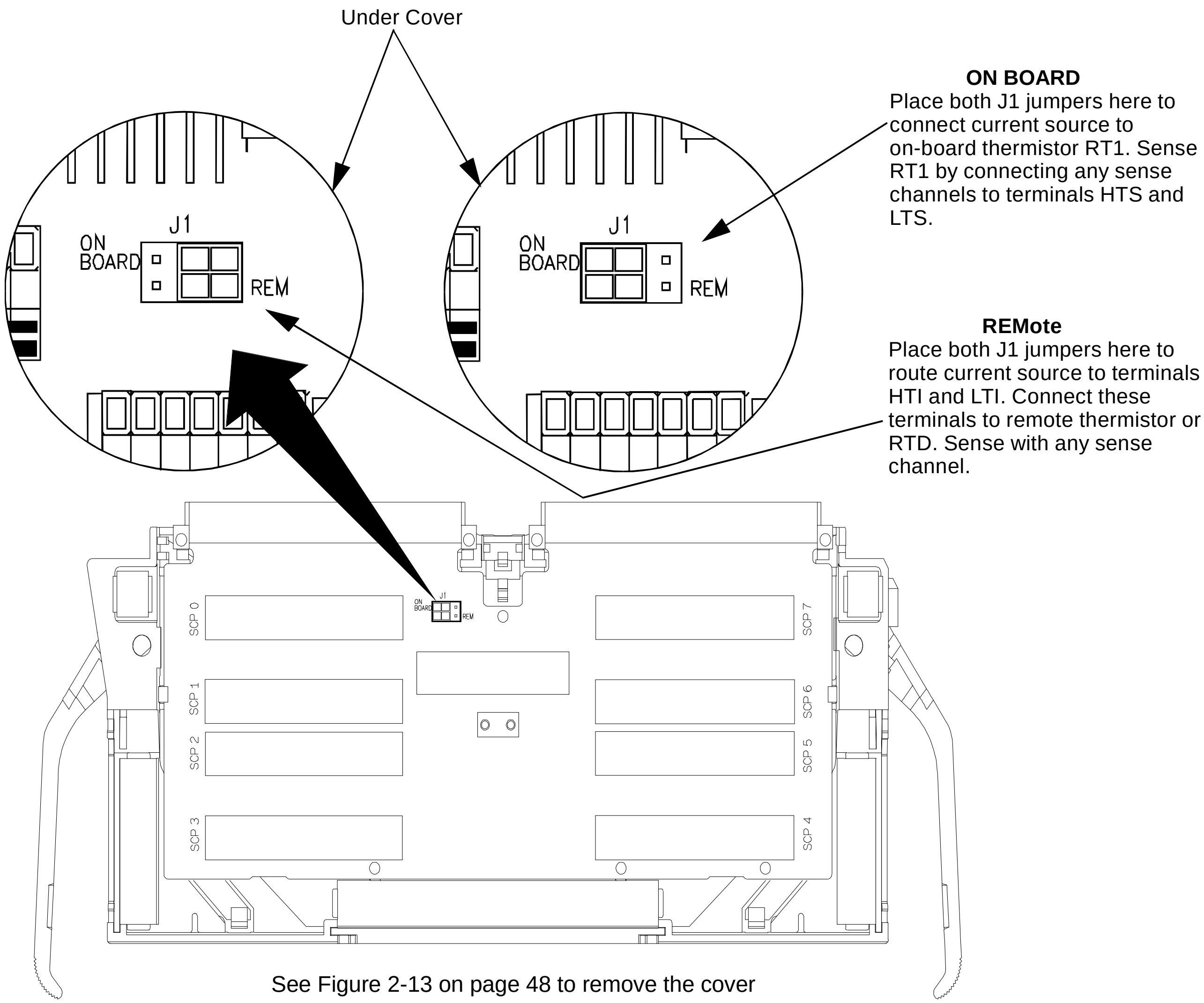


Figure 2-12. Temperature Sensing for the Terminal Module

Wiring and Attaching the Terminal Module

Figures 2-13 and 2-14 show how to open, wire, and attach the terminal module to an HP E1422.

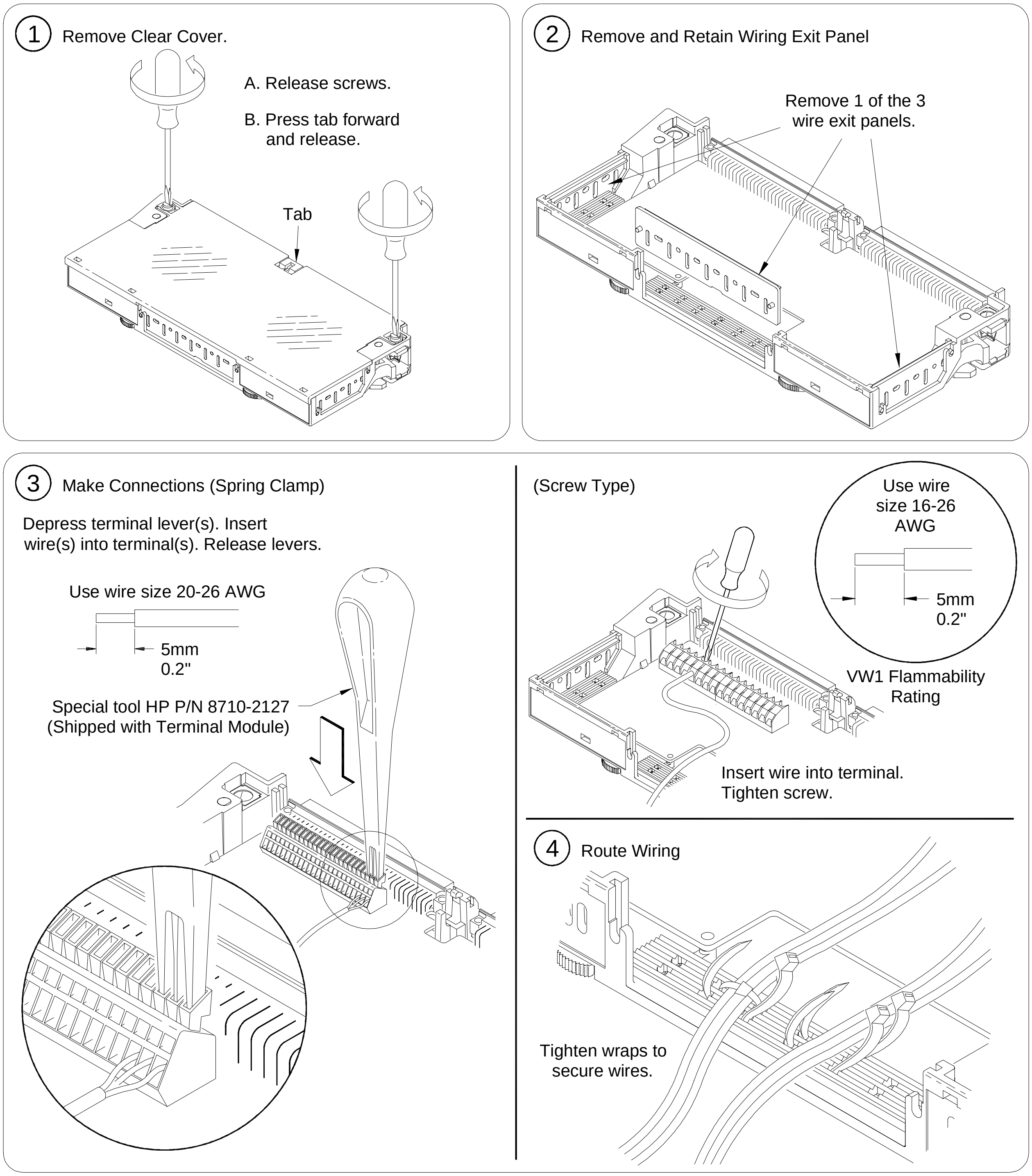


Figure 2-13. Opening and Wiring the E1422's Terminal Module

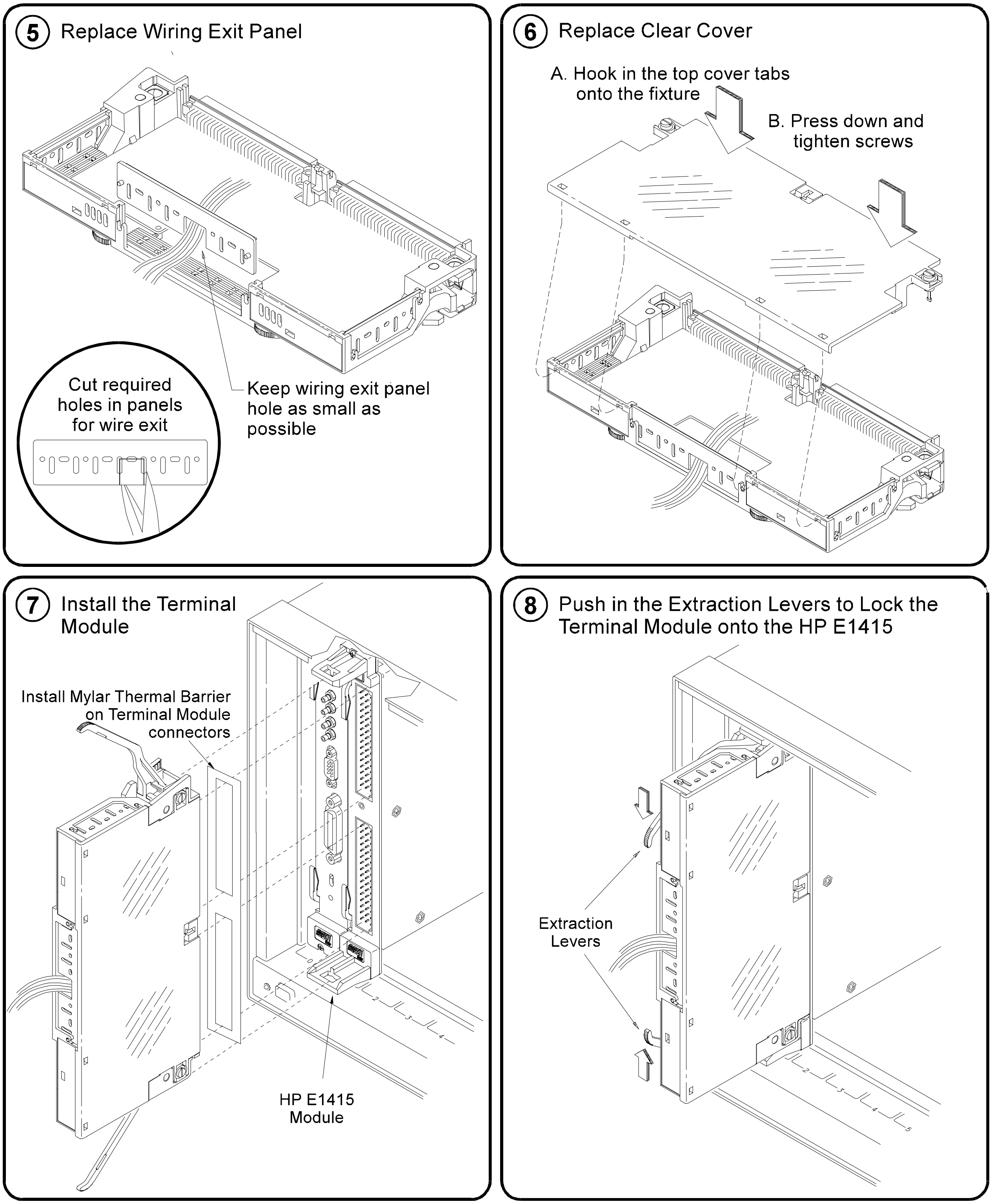


Figure 2-14. Closing and Attaching the HP E1422 Terminal Module

Removing the HP E1422 Terminal Modules

Figure 2-15 shows how to remove the Spring Terminal and Screw Terminal Modules from the HP E1422A.

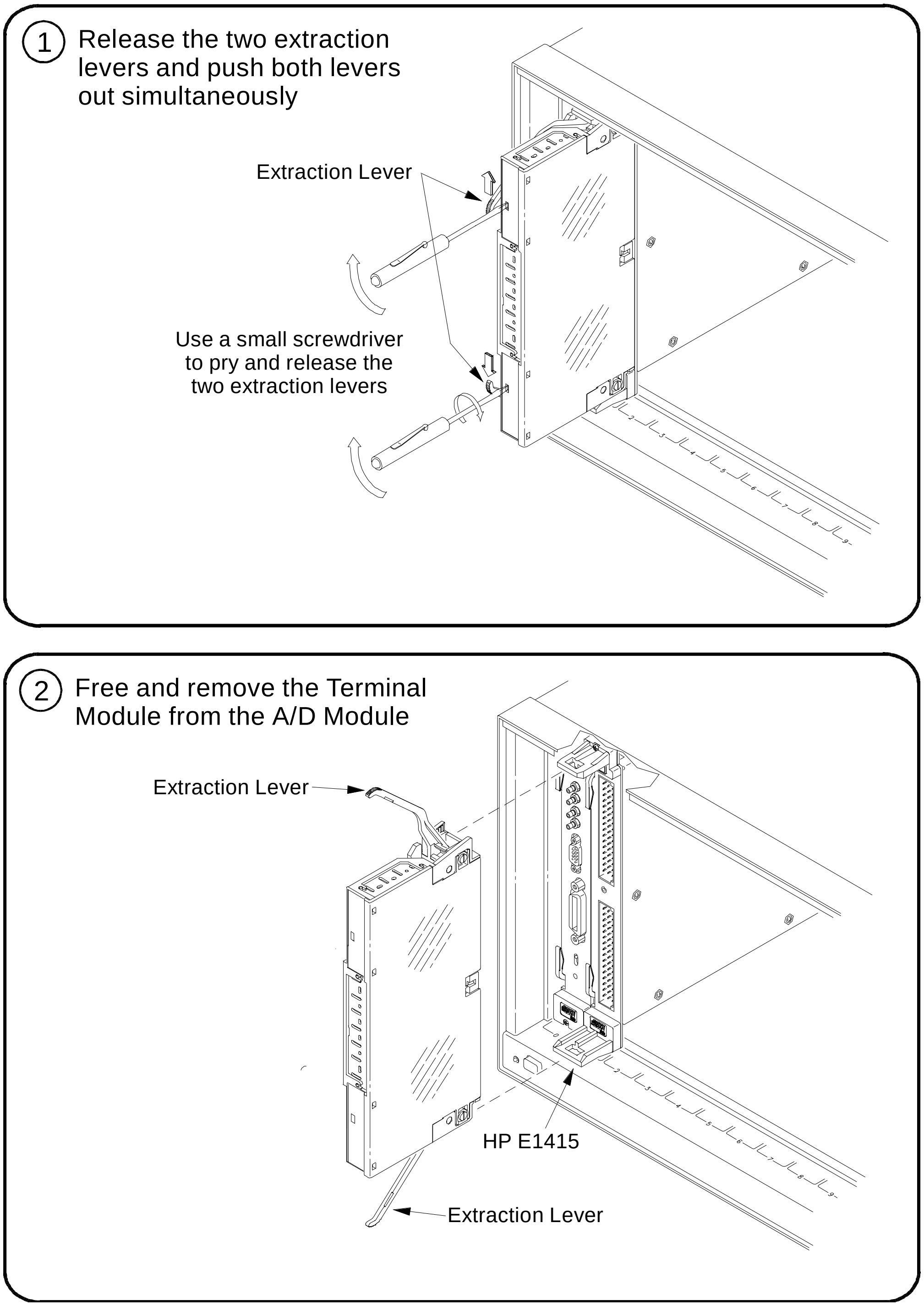


Figure 2-15. Removing the Screw and Spring Terminal Modules

Attaching and Removing the HP E1422 RJ-45 Module

Figure 2-16 shows how to remove the RJ-45 Terminal Module.

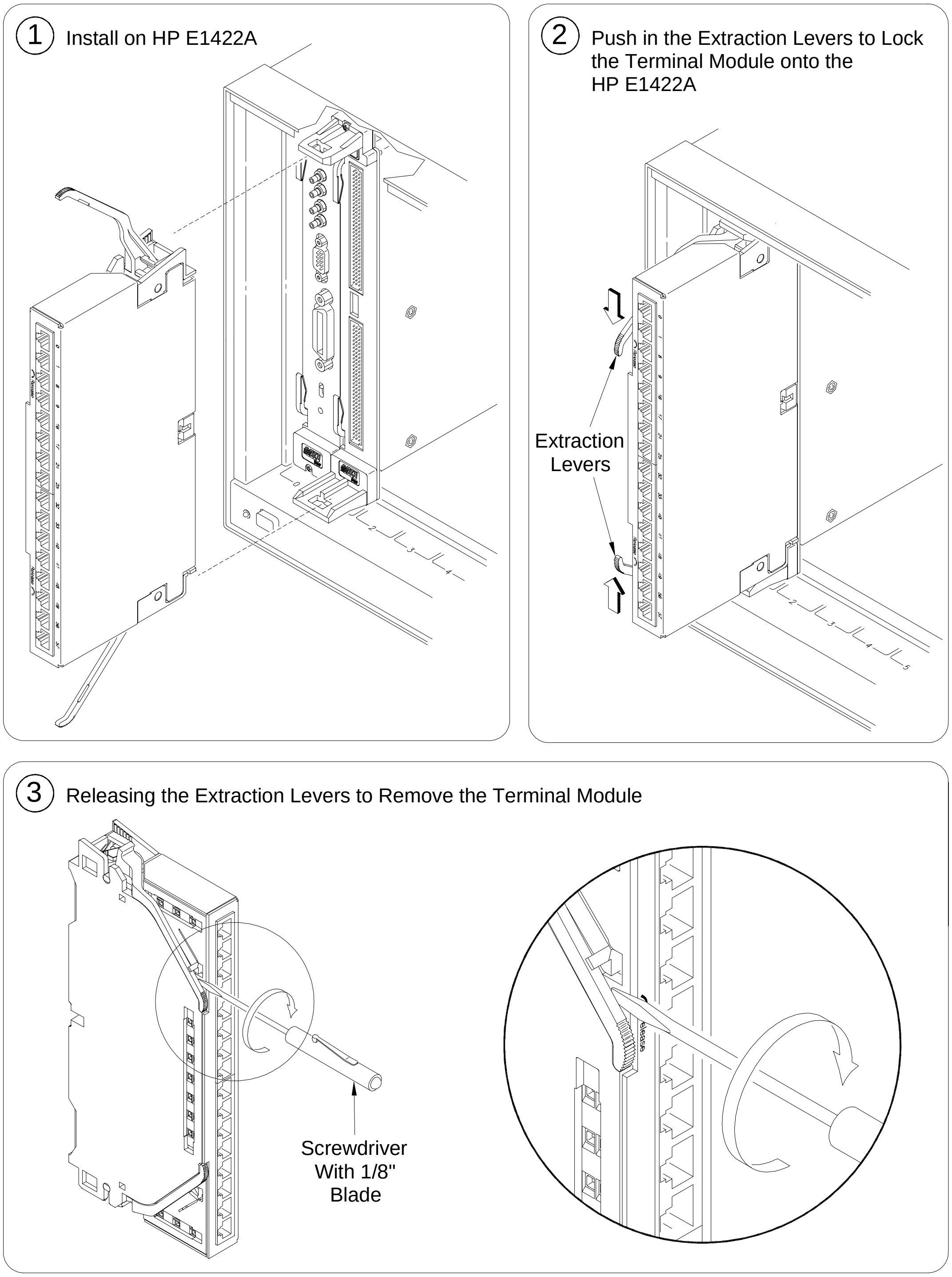


Figure 2-16. Removing the RJ-45 Terminal Module

Adding Components to the Terminal Module

The back of the terminal module P.C. board provides surface mount pads which you can use to add serial and parallel components to any channel's signal path. Figure 2-17 shows additional component locator information (see the schematic and pad layout information on the back of the teminal module P.C. board). Figure 2-18 shows some usage example schematics.

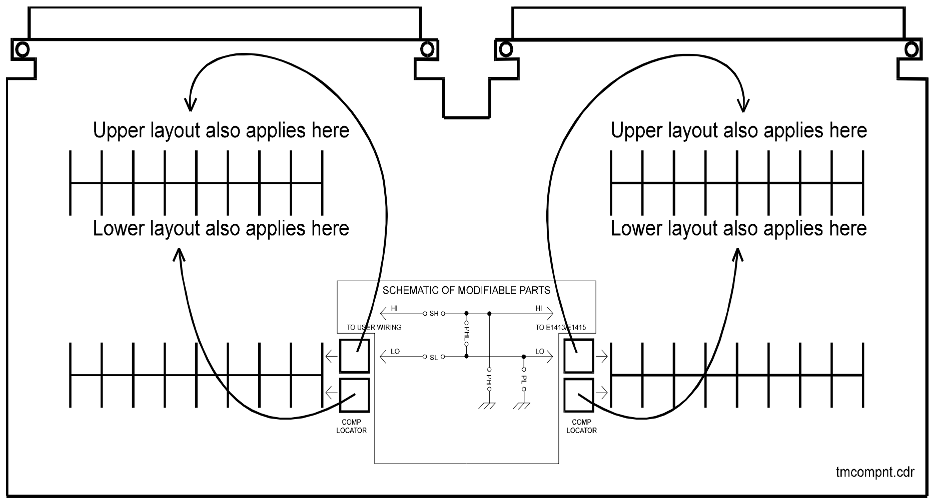


Figure 2-17. Additional Component Location

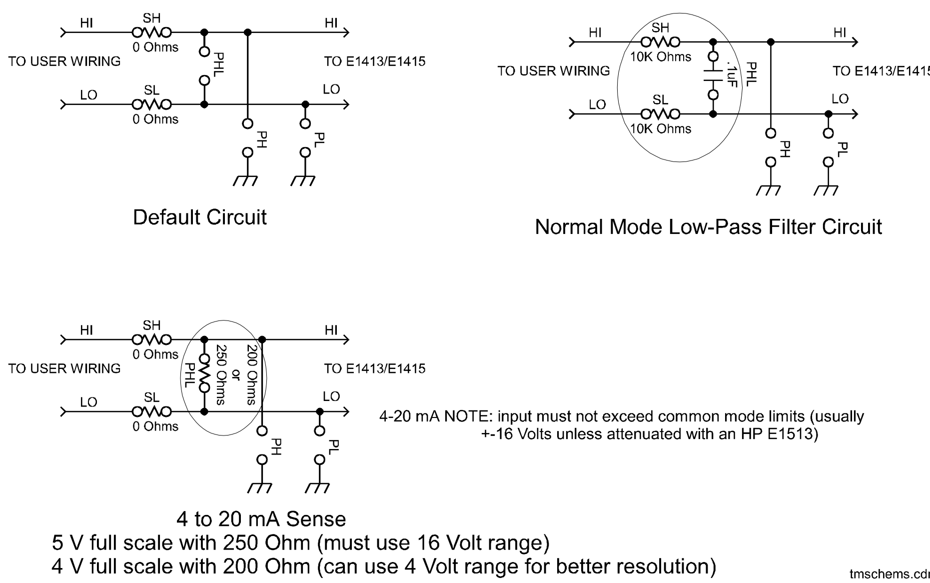


Figure 2-18. Series & Parallel Component Examples

Spring and Screw Terminal Module Wiring Maps

Figure 2-19 shows the Spring Terminal Module wiring map.

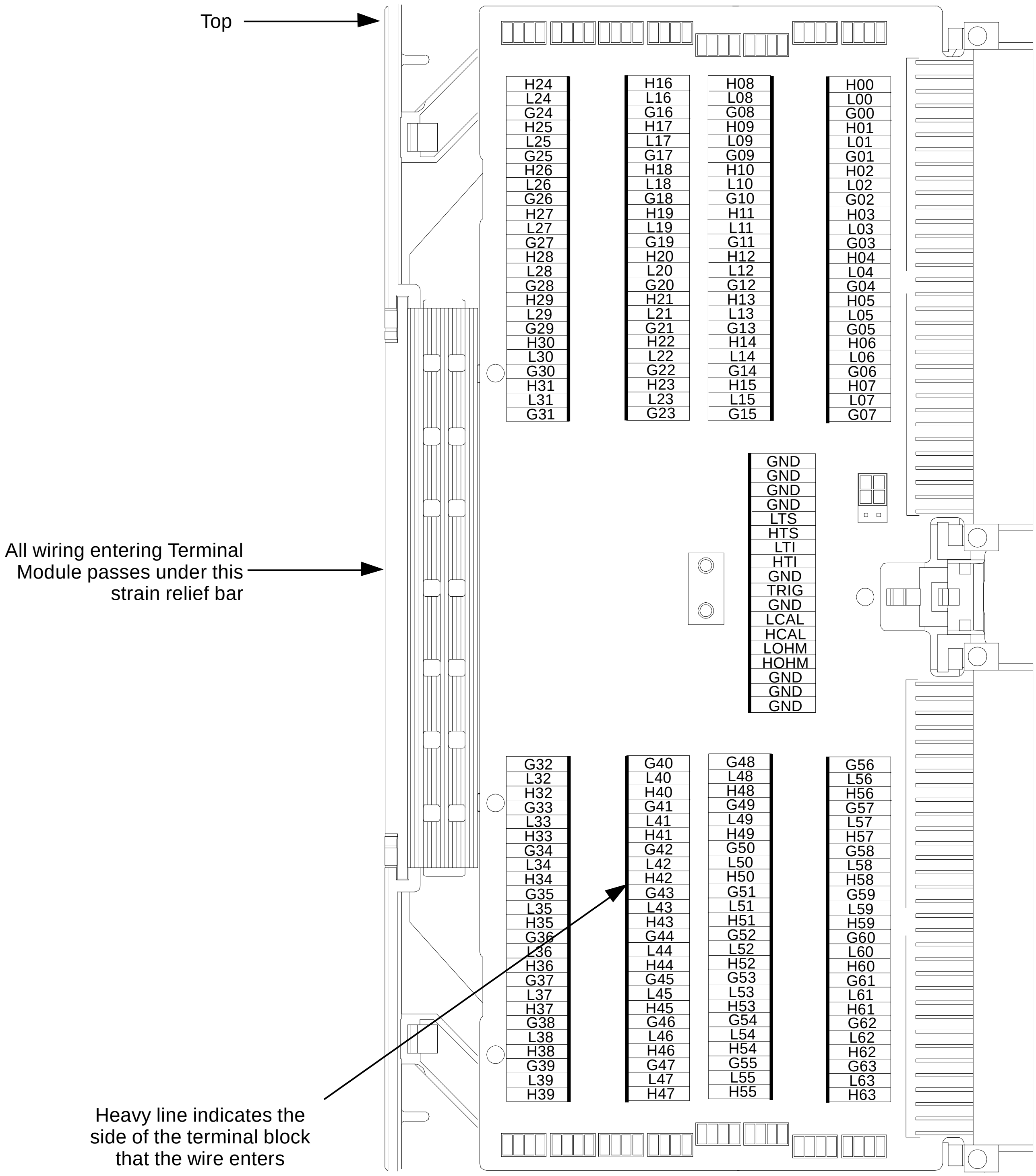


Figure 2-19. Spring Terminal Module Full-Size Wring Map

Figure 2-20 shows the Screw Terminal Module wiring map

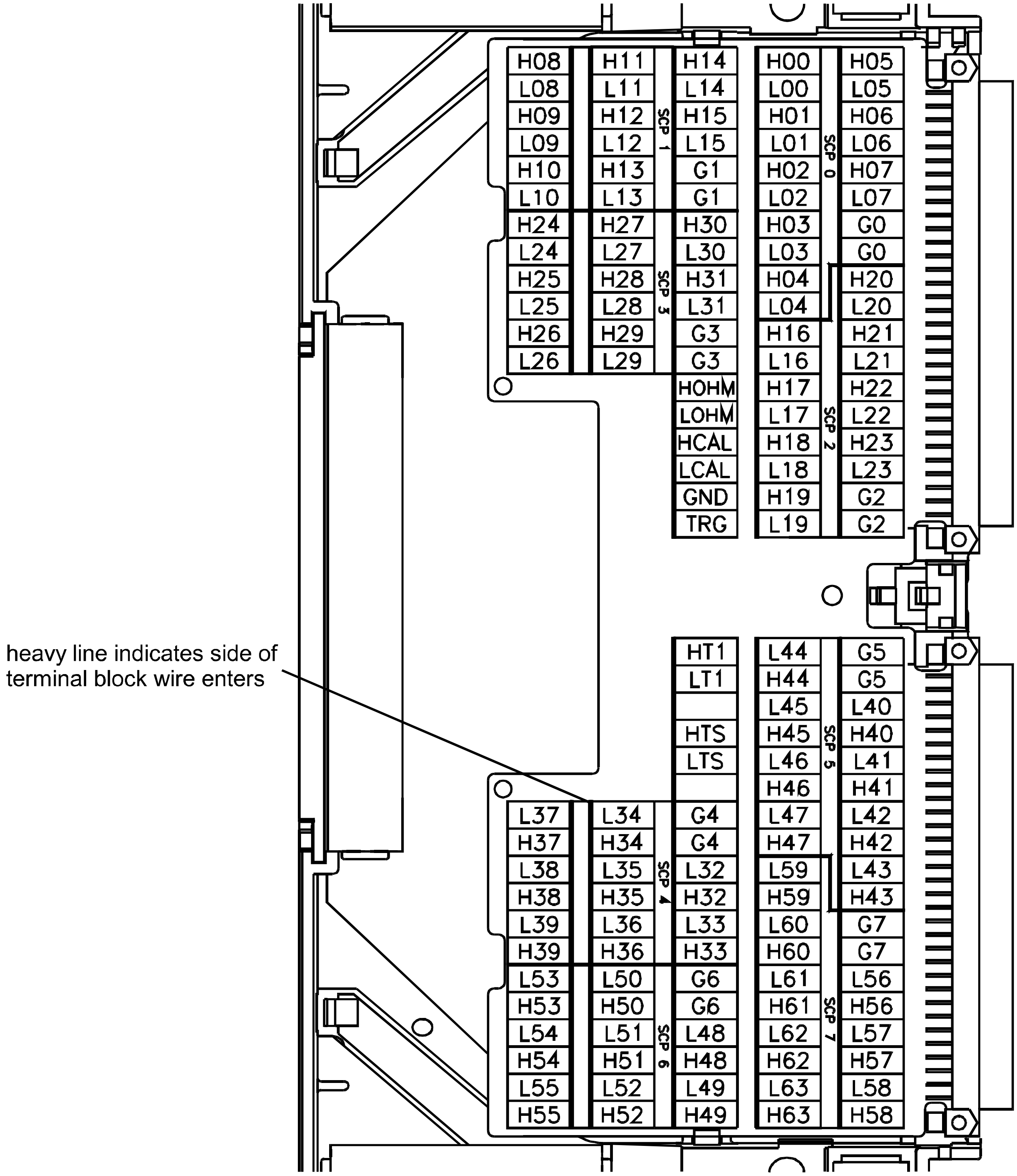


Figure 2-20. Screw Terminal Module Full-Size Wiring Map

Chapter 3

Programming the HP E1422A & HP E1529A for Remote Strain Measurement

About This Chapter

This chapter describes using the HP E1422A in combination with the HP E1529A Remote Channel SCP and HP E1529A Remote Strain Conditioning Units to make large channel count strain measurements. We show the system used in a strictly data acquisition mode where after configuration it is driven by a channel list you define (the Scan List), and sends the measurements to the unit's FIFO buffer and Current Value Table (CVT) for transfer to your computer. Of course you can also create control algorithms that execute concurrently with the Scan List driven data acquisition operation. Chapter 4 and Chapter 5 cover general data acquisition and control programming with algorithms. This chapter assumes that you are the expert when it comes to making strain measurements so we're simply going to show you how to make your strain measurements with the HP Remote Strain Measuring System (HP E1422A, HP E1539As, and HP E1529As). The chapter will cover:

Instrument Setup for Remote Strain Measurements

This section involves:

- Preparing the HP E1422A for installation into a VXIbus Mainframe
- Preparing the HP E1429A for use
- Connecting the HP E1422A to HP E1529A Remote Strain Completion units.
- Connecting Excitation power supplies to the HP E1529A
- Connecting strain bridges to the HP E1529A

Preparing the HP E1422A for Installation

The HP E1422A needs HP E1539A SCPs to control Remote Signal Conditioning Units like the HP E1529A Remote Strain Conditioning Unit. Chapter 1 “Getting Started” covers everything you need to do before you install your HP E1422A in its Mainframe. This includes switch settings and SCP installation. After performing the operations in Chapter 1, return here for Remote Strain specific operations.

Overview

Before we get into the specifics of configuring a Remote Strain Measuring System, it might help you to see what we are going to set-up. Figure 3-1 shows the components and connections of a remote strain measuring system. The circled letters identify connections that will be referred to in later sections.

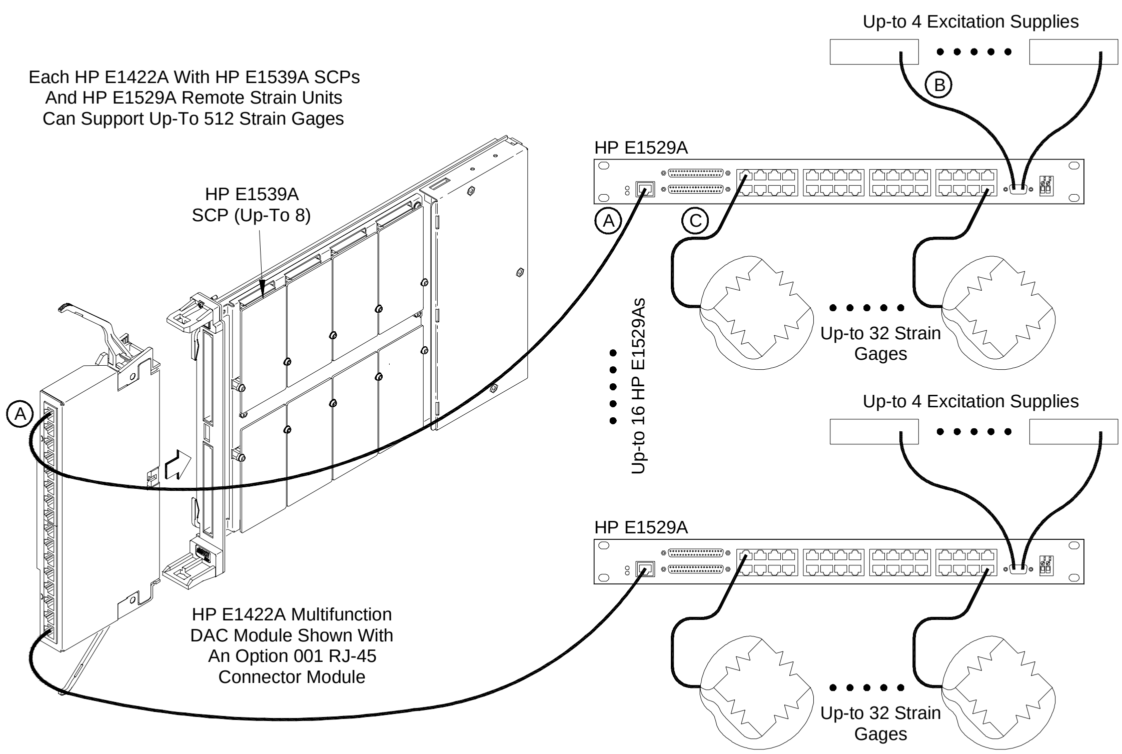


Figure 3-1. Components of the Remote Strain Measuring System

Preparing the HP E1529A for Use

For most applications, the HP E1529A is ready for installation as delivered. It is designed to be easily rack mounted in a system cabinet by its built-in front panel extensions. All user connection are accessible on the front panel with the exception of the line-cord jack which is on the rear of the unit. The only pre-installation operation you might need to consider is installation of user supplied quarter-bridge completion resistors. If your application requires these, see “Installing User Selected 1/4 Bridge Resistors (optional)” in the following section.

WARNING **Ground the equipment: The Safety earth ground for the HP E1529A is supplied through the ground conductor of the power cable. Make sure your installation’s AC line supply connectors provide a suitable earth ground.**

WARNING **The power cord is the only way to disconnect the HP E1529A from AC power. Therefore, the power cord must be accessible to the operator at all times. When the HP E1529A is mounted in a system cabinet, the power cord need not be accessible since the cabinet must have its own disconnect device.**

Installing User Selected 1/4 Bridge Resistors (optional)

Perform this operation only if you require one or more HP E1529A channels to provide 1/4 Bridge completion of other than 120Ω or 350Ω . Only those with experience soldering components on printed circuit boards should attempt this installation. The HP E1529A provides locations on its printed circuit board to install your own 1/4 bridge completion resistors. Bridge configuration commands then can switch your resistors into the bridge completion circuits where you’ve installed custom value resistors.

WARNING **Keep away from live circuits: Operating personnel must not remove equipment covers or shields. Procedures involving the removal of covers or shields are for use by service-trained personnel only. Under certain conditions, dangerous voltages may exist even with the equipment switched off. To avoid dangerous electrical shock, DO NOT perform procedures involving cover or shield removal unless you are qualified to do so.**

Removing the Top Cover

Figure 3-2 shows how to access the printed circuit board where the user specified resistors will be installed. Notice that both a surface-mount as well as a through-hole position is provided for each channel.

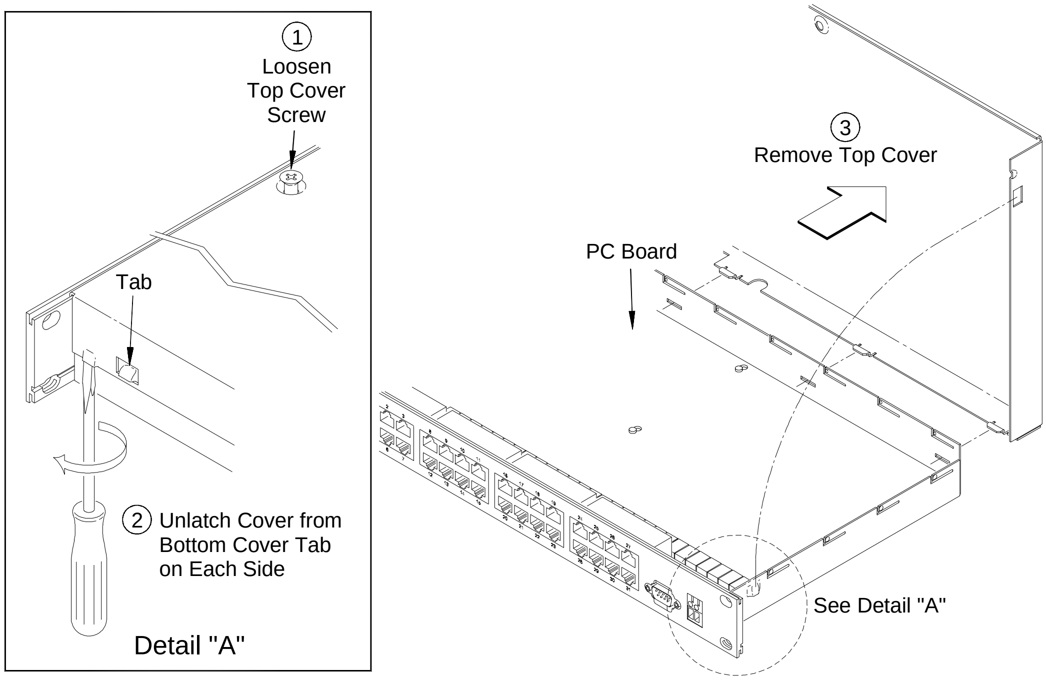


Figure 3-2. Removing the HP E1529A Top Cover

Locating Resistors

Figure 3-3 provides the relationship between P.C. board location and bridge resistor channel number. The surface mount pads nearest the through-hole locations are in parallel with them.

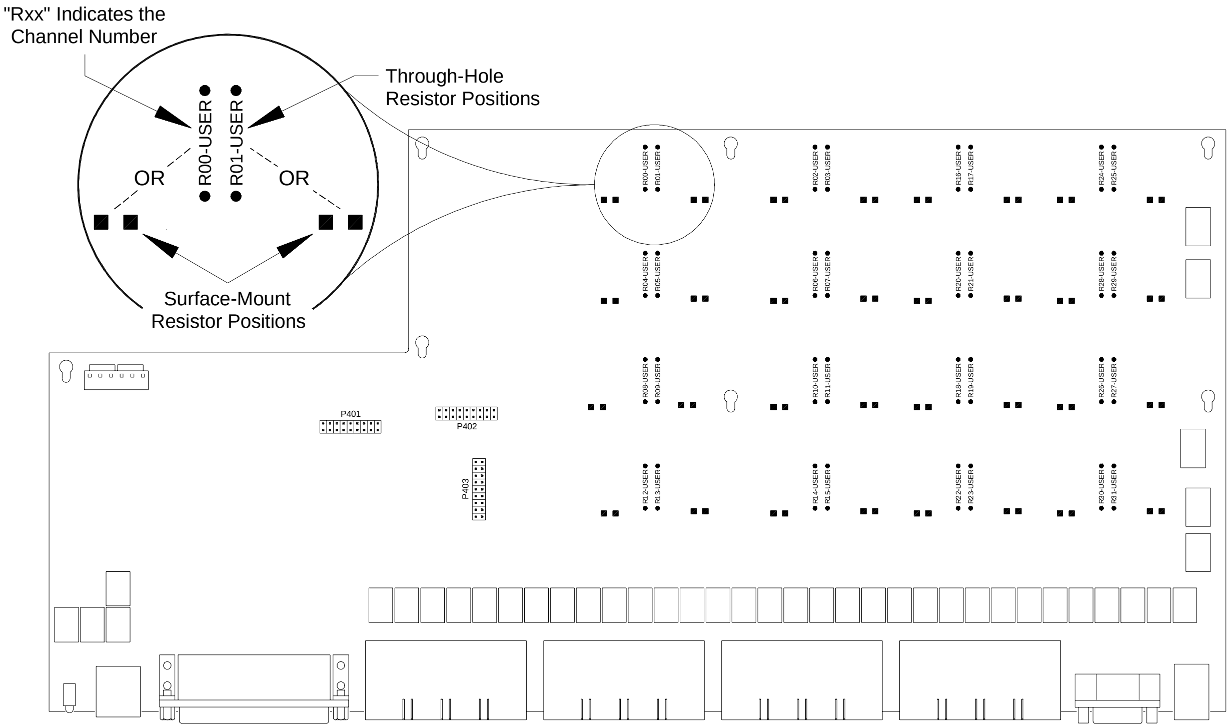


Figure 3-3. Locating User 1/4 Bridge Resistor Positions

Installing Resistors

Figure 3-4 shows a typical user selected 1/4 bridge resistor installation. Note that resistor installations can be accomplished from the top of the board without further disassembly. If you are installing through-hole resistors, you must be very careful to observe the specified maximum safe resistor lead length to avoid shorting the resistor to the chassis.

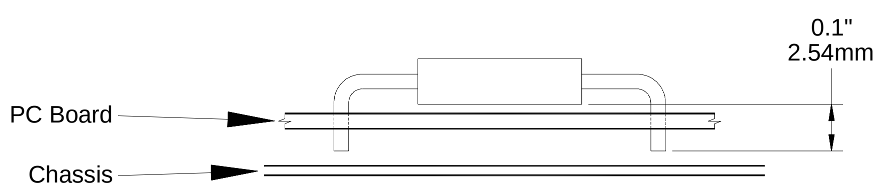


Figure 3-4. Installing User 1/4 Bridge Resistors

Connecting HP E1529As to the HP E1422A

The cable between an HP E1422A and each HP E1529A (connection "A" in Figure 3-1) is a standard type of cable used in computer Local Area Networks (LANs). The HP E1529A can be any distance up-to 1000 feet from the HP E1422A, and the interconnect cable can be easily custom made to fit the installation. In fact, if your firm has an Information Technology department, they may already be making or having made this same type of cable assembly.

The cable assembly as a whole must comply with the TIA/EIA-568 Category 5 standard for LAN interconnecting cable. This is a performance based standard and will insure that the HP E1422A will be able to make accurate measurements from an HP E1529A over the maximum cable length of 1000 feet (305 meters). Additionally, the cable and connectors must be shielded.

Cabling Supplies and Tools

Tables 3-1 and 3-2 show part numbers for supplies that will allow you to quickly custom make high quality cables for your installation. If you opt to have a third party build your cables, make certain they supply you with cables that comply with the TIA/EIA-568 Category 5 standard and are shielded. The part numbers shown here are those of major suppliers in the industry. These numbers can be cross-referenced to other supplier’s equivalent products.

Please note that safety standards for wiring (flammability etc.) may apply to your installation and you should check applicable codes and standards in your area and select the proper type of cable accordingly (plenum vs. non-plenum types etc.).

Table 3-1.

Cable Part Numbers for Belden Wire & Cable Company	
Overall-Shielded Twisted Pair (STP) TIE/EIA-568 Category 5 (4 twisted pairs)	
Plenum Type: (Flame Retardant Jacket and FEP ‡Teflon insulation)	1586A
Non-Plenum Type: (PVC Jacket and polyolefin insulation)	1584A

‡DuPont trademark

Table 3-2.

Connector Part Numbers for AMP Incorporated	
RJ-45 Plug: (for solid conductors and round shielded cable)	5-569530-4
RJ-45 Plug: (for stranded conductors and round shielded cable)	5-569550-4
Strain Relief	558527-1
Hooded Boot: (replace X with 0=Gry, 1=Blk, 2=Lt. Almond, 3=Red, 4=Grn, 5=Blu, 6=Yel, 7=Org, 8=Wht, 9=Vio)	569875-X
RJ-45: Plug Installation Tool with 8-position dies	2-231652-1

Two Interconnect Methods

Depending on which Terminal Module you ordered with your HP E1422A, there are two methods of interconnecting an HP E1529A to the HP E1422A (connection "A" in Figure 3-1).

The Option 001 RJ-45 Connector Module

The RJ-45 Connector Module is used when most or all of HP E1422A SCP positions contain an HP E1539A Remote Channel SCP. For RSCUs, you just plug one end into the HP E1422A, and the other into the HP E1529A's Data Interface connector.Figure 3-5 shows this connection and includes a schematic diagram of the RJ-45-to-RJ-45 cable. See Figure 2-4 on page 39 for on-board SCP channel connection through the RJ-45 connector module.

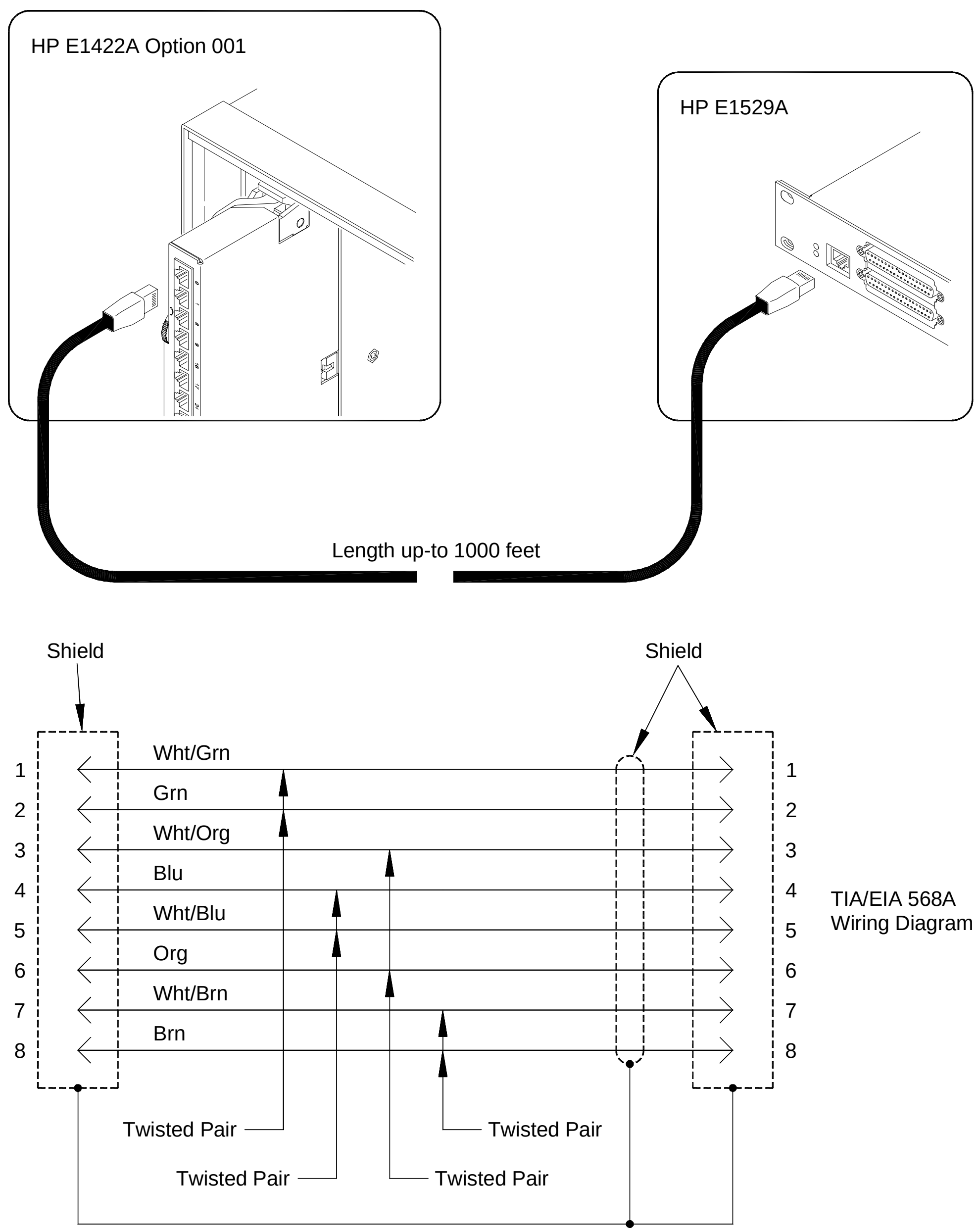


Figure 3-5. Connecting HP E1529As to the RJ-45 Connector Module

Spring, and Screw Terminal Modules

For mixed on-board SCP channels and RSCU operation, you can use the spring type, or screw type terminal modules. For standard SCP channel connections see Chapter 2 “Field Wiring” on page 33. For remote channels you connect the individual wires from each HP E1529A’s data interface cable to the appropriate terminals for remote channel operation. The HP E1539A SCP is supplied with signal locator labels for each SCP position on a Spring Terminal Module. No label is provided for the Screw terminal module. Instead, Table 3-3 provides the relationship between each HP E1439A signal name and associated terminal name as printed on the Terminal Module.

Table 3-3.

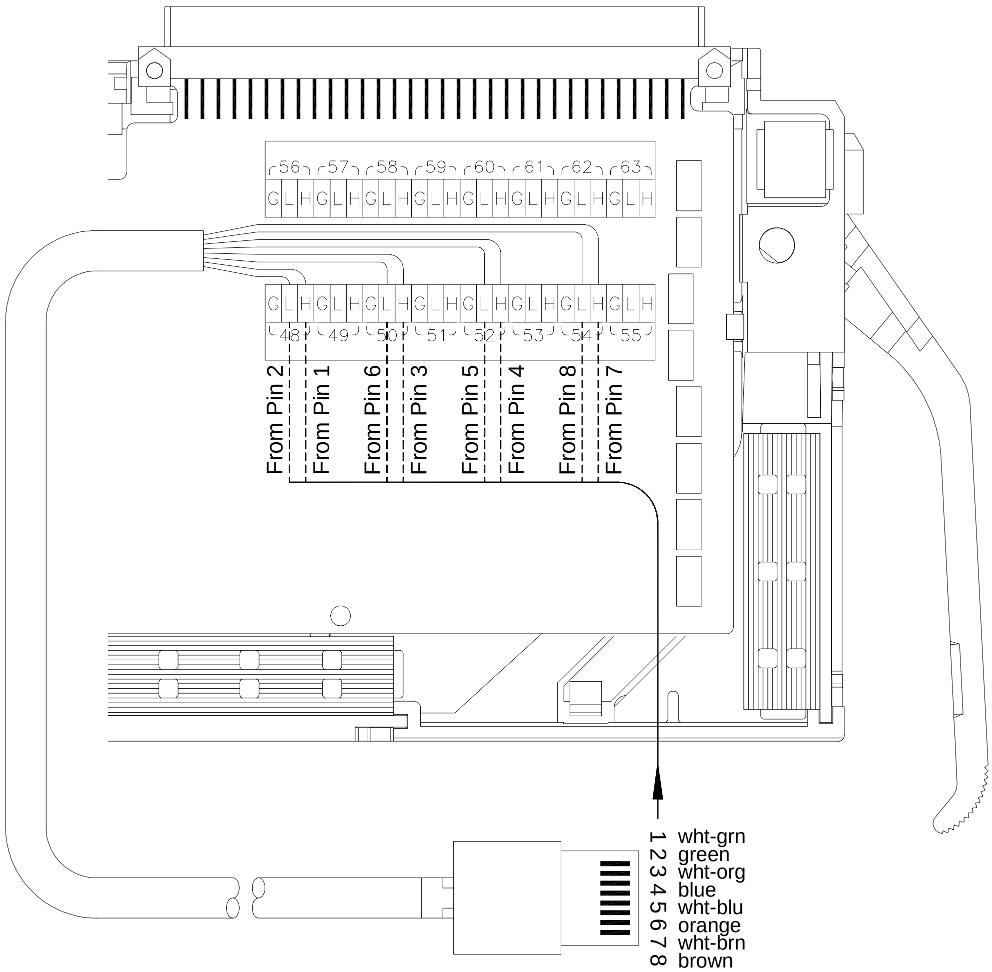
SCP Signal Names - to - Terminal Names				
SCP Position	**Plug Pin#**	**HP E1539A Signal Name (with EIA/TIA-568A wire color-code)**	**Terminal Name on Terminal Module (SCP’s low channel)**	**Terminal Name on Terminal Module (SCP’s High Channel)**
SCP Position 0 Addresses 10000 to 10131	1	Analog+ (wht-green)	HI 00	HI 01
	2	Analog- (green)	LO 00	LO 01
	3	Cal+ (wht-orange)	HI 02	HI 03
	4	RS-485+ (blue)	HI 04	HI 05
	5	RS-485- (wht-blue)	LO 04	LO 05
	6	Cal- (orange)	LO 02	LO 03
	7	Trigger+ (wht-brown)	HI 06	HI 07
	8	Trigger- (brown)	LO 06	LO 07
SCP Position 1 Addresses 10800 to 10931	1	Analog+ (wht-green)	HI 08	HI 09
	2	Analog- (green)	LO 08	LO 09
	3	Cal+ (wht-orange)	HI 10	HI 11
	4	RS-485+ (blue)	HI 12	HI 13
	5	RS-485- (wht-blue)	LO 12	LO 13
	6	Cal- (orange)	LO 10	LO 11
	7	Trigger+ (wht-brown)	HI 14	HI 15
	8	Trigger- (brown)	LO 14	LO 16
SCP Position 2 Addresses 11600 to 11731	1	Analog+ (wht-green)	HI 16	HI 17
	2	Analog- (green)	LO 16	LO 17
	3	Cal+ (wht-orange)	HI 18	HI 19
	4	RS-485+ (blue)	HI 20	HI 21
	5	RS-485- (wht-blue)	LO 20	LO 21
	6	Cal- (orange)	LO 18	LO 19
	7	Trigger+ (wht-brown)	HI 22	HI 23
	8	Trigger- (brown)	LO 22	LO 23
SCP Position 3 Addresses 12400 to 12531	1	Analog+ (wht-green)	HI 24	HI 25
	2	Analog- (green)	LO 24	LO 25
	3	Cal+ (wht-orange)	HI 26	HI 27
	4	RS-485+ (blue)	HI 28	HI 29
	5	RS-485- (wht-blue)	LO 28	LO 29
	6	Cal- (orange)	LO 26	LO 27
	7	Trigger+ (wht-brown)	HI 30	HI 31
	8	Trigger- (brown)	LO 30	LO 31

Table 3-3.

SCP Signal Names - to - Terminal Names				
SCP Position	**Plug Pin#**	**HP E1539A Signal Name (with EIA/TIA-568A wire color-code)**	**Terminal Name on Terminal Module (SCP's low channel)**	**Terminal Name on Terminal Module (SCP's High Channel)**
SCP Position 4 Addresses 13200 to 13331	1	Analog+ (wht-green)	HI 32	HI 33
	2	Analog- (green)	LO 32	LO 33
	3	Cal+ (wht-orange)	HI 34	HI 35
	4	RS-485+ (blue)	HI 36	HI 37
	5	RS-485- (wht-blue)	LO 36	LO 37
	6	Cal- (orange)	LO 34	LO 35
	7	Trigger+ (wht-brown)	HI 38	HI 39
	8	Trigger- (brown)	LO 38	LO 39
SCP Position 5 Addresses 14000 to 14131	1	Analog+ (wht-green)	HI 40	HI 41
	2	Analog- (green)	LO 40	LO 41
	3	Cal+ (wht-orange)	HI 42	HI 43
	4	RS-485+ (blue)	HI 44	HI 45
	5	RS-485- (wht-blue)	LO 44	LO 45
	6	Cal- (orange)	LO 42	LO 43
	7	Trigger+ (wht-brown)	HI 46	HI 47
	8	Trigger- (brown)	LO 46	LO 47
SCP Position 6 Addresses 14800 to 14931	1	Analog+ (wht-green)	HI 48	HI 49
	2	Analog- (green)	LO 48	LO 49
	3	Cal+ (wht-orange)	HI 50	HI 51
	4	RS-485+ (blue)	HI 52	HI 53
	5	RS-485- (wht-blue)	LO 52	LO 53
	6	Cal- (orange)	LO 50	LO 51
	7	Trigger+ (wht-brown)	HI 54	HI 55
	8	Trigger- (brown)	LO 54	LO 55
SCP Position 7 Addresses 156000 to 157131	1	Analog+ (wht-green)	HI 56	HI 57
	2	Analog- (green)	LO 56	LO 57
	3	Cal+ (wht-orange)	HI 58	HI 59
	4	RS-485+ (blue)	HI 60	HI 61
	5	RS-485- (wht-blue)	LO 60	LO 61
	6	Cal- (orange)	LO 58	LO 59
	7	Trigger+ (wht-brown)	HI 62	HI 63
	8	Trigger- (brown)	LO 62	LO 63

Example Terminal Module to HP E1529A Connection

Figure 3-6 shows a typical connection to an HP E1529A through one of the optional terminal modules. In this case, the connection is to the low channel on the HP E1539A in SCP position number 6 (channels 14800 - 14831). For connection to other SCP positions, use the "Terminal Module Connection Formula" from Figure 3-6 or the data from Table 3-3.



Terminal Module Connection Formula

SCP Low Channel			SCP High Channel		
SCP Pos. * 8 + ↓			SCP Pos. * 8 + ↓		
wht-grn	0	Hi	wht-grn	1	Hi
green	0	Lo	green	1	Lo
wht-org	2	Hi	wht-org	3	Hi
orange	2	Lo	orange	3	Lo
blue	4	Hi	blue	5	Hi
wht-blu	4	Lo	wht-blu	5	Lo
wht-brn	6	Hi	wht-brn	7	Hi
brown	6	Lo	brown	7	Lo

Figure 3-6. Connecting an HP E1529A to an Optional Terminal Module

Connecting Excitation Supplies

This connection is shown as "B" in Figure 3-1. The HP E1529A uses external excitation supplies. There are four pairs of input pins (and Gnd) at the "Bridge Excitation" connector for up-to four individual excitation supplies. Each of these four inputs powers eight channels through a programmable switch. You can of course parallel-wire multiple excitation inputs to a single power supply.

Notes

1. The maximum excitation voltage the HP E1422A can sense through the HP E1529A's excitation sense path is 16 volts (±8VDC centered about the Gnd terminal). If you supply higher excitation voltage through the HP E1529A, don't connect the excitation sense terminals.
2. Make sure that the power supply you choose can supply the current requirement of all of the bridges it can be switched to. It will be connected to all bridges you are going to measure before a measurement scan is started. The supply switches can not be programmatically re-configured while a measurement scan is under way. You must halt a measurement scan to programmatically re-configure the excitation supply switches.

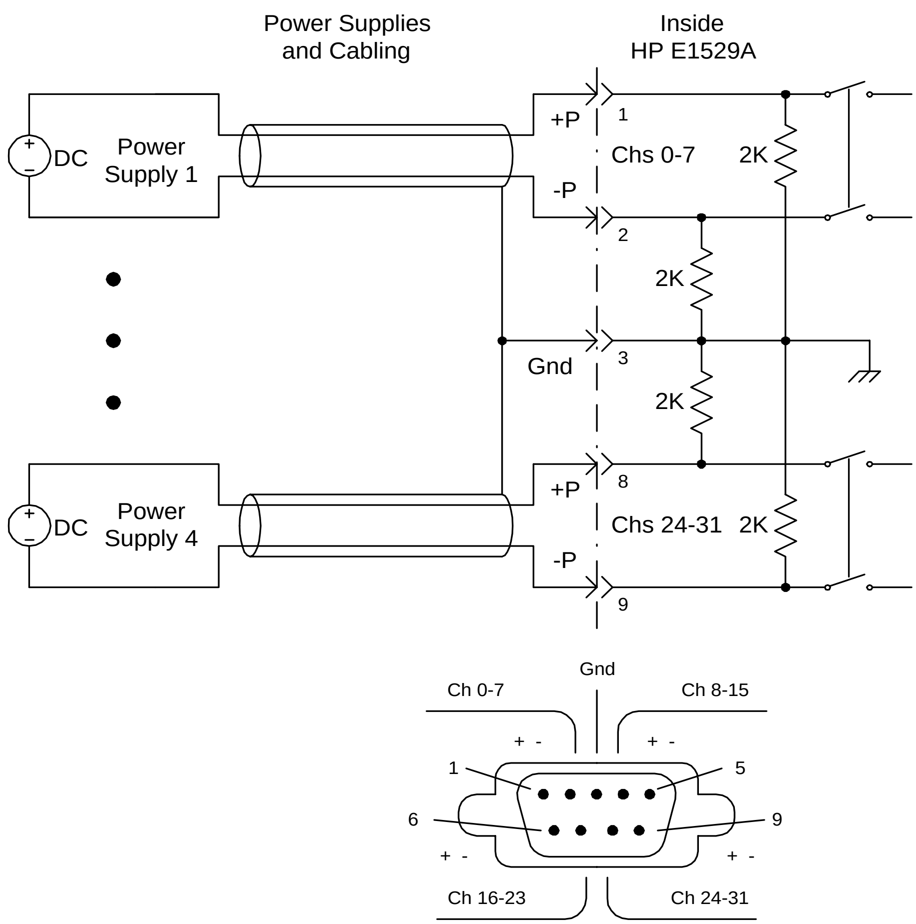


Figure 3-7. Excitation Supply Connections

Connecting the HP E1529A to Strain Gages

The following discussion relates to the connection marked "C" in Figure 3-1 on page 56. We'll show you how to connect your strain gages to the RJ-45 telecom connectors. These connections can be made with the same type of cable and crimp-on connectors used for Data Interface connection (connection "A" in Figure 3-1). See Figure 3-8 for an example gage connection.

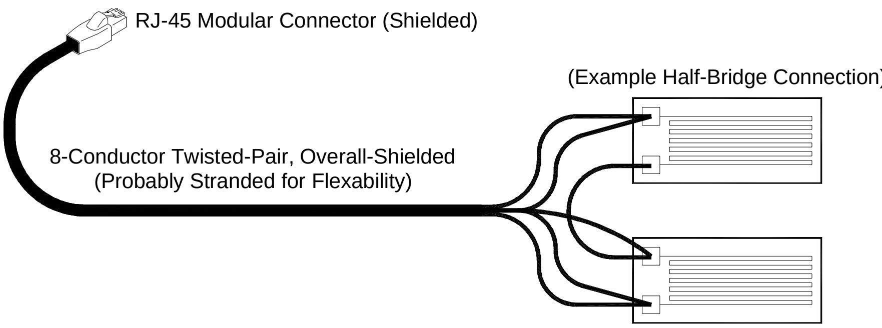


Figure 3-8. HP E1422 to Strain Gage Connection

Channel Connector Pin-to-Signal Relationship

Figure 3-9 shows the pin-to-signal relationship for each HP E1529 strain gage connector. You will find these same signal names on the following strain bridge configuration illustrations too.

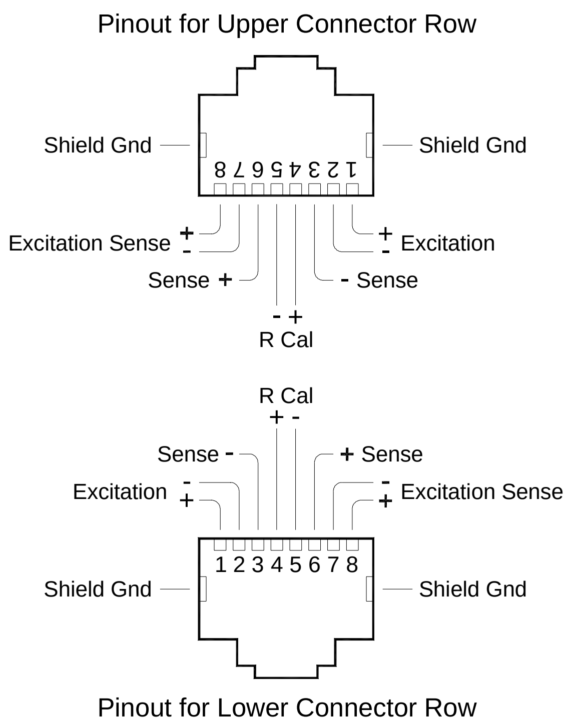


Figure 3-9. Pin-out for Strain Gage Connectors

HP E1529A Bridge Configurations

The Quarter Bridge configuration

Figure 3-10 shows the connections to the 8-pin telecom connector for a quarter bridge configuration. It also shows a simplified schematic of the bridge completion settings for a quarter bridge channel.

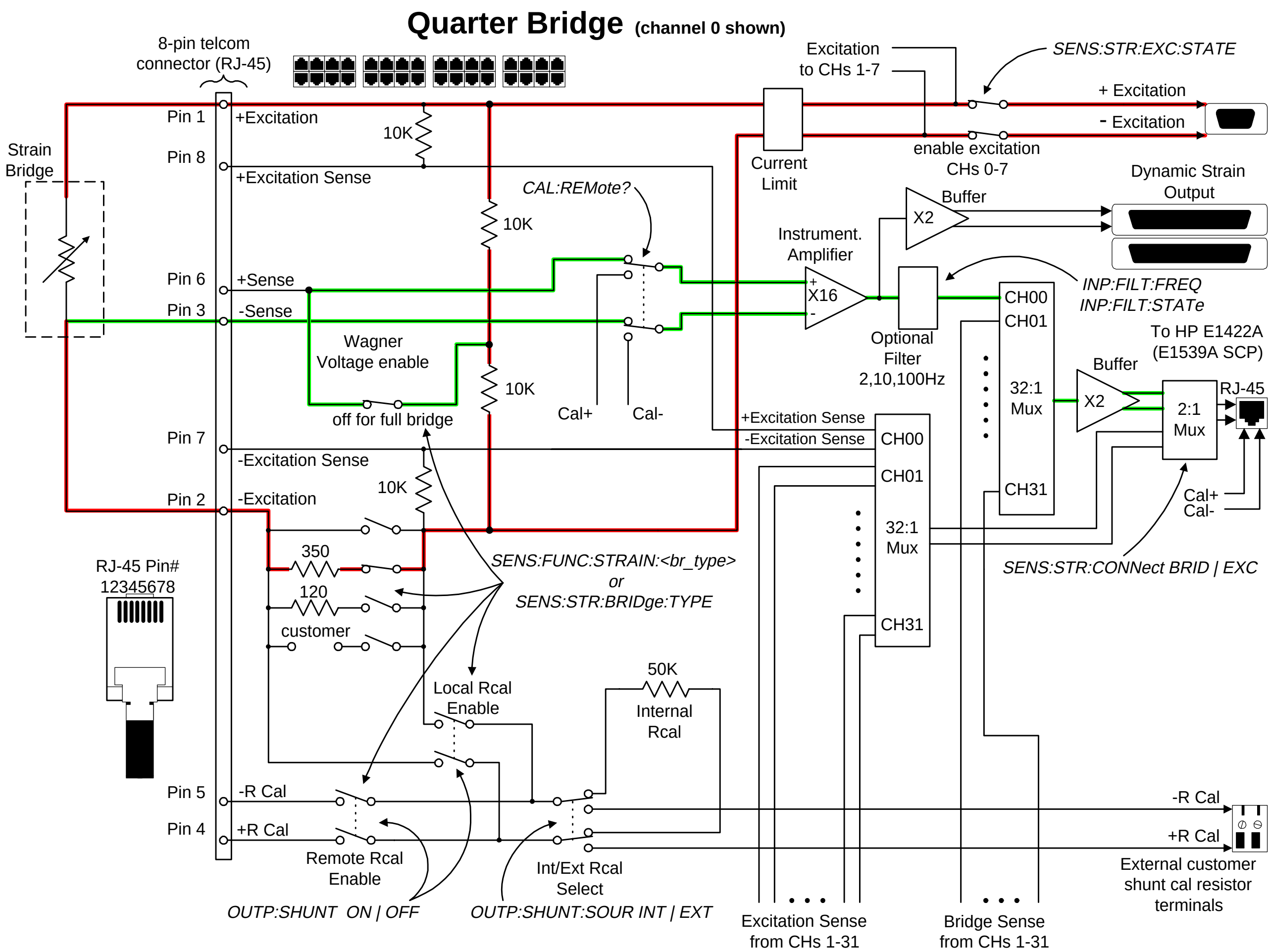


Figure 3-10. Bridge Completion for a Quarter Bridge Channel

Note While the diagram above shows amplifier gain in the measurement path, the measurement values returned by these channels are corrected by the HP E1422A's DSP chip (Digital Signal Processor) to reflect the actual value at the user input terminal. The only time you need to consider gain is when the input voltage times the gain would overload the A/D range chosen with a SENS:FUNC:... <range>,(<ch_list>) command. For example, with a gain of 32, any input voltage greater than 0.5V would cause an overload reading even on the highest A/D range (16V).

The Half Bridge configuration

Figure 3-11 shows the connections to the 8-pin telecom connector for a half bridge configuration. It also shows a simplified schematic of the bridge completion settings for a half bridge channel.

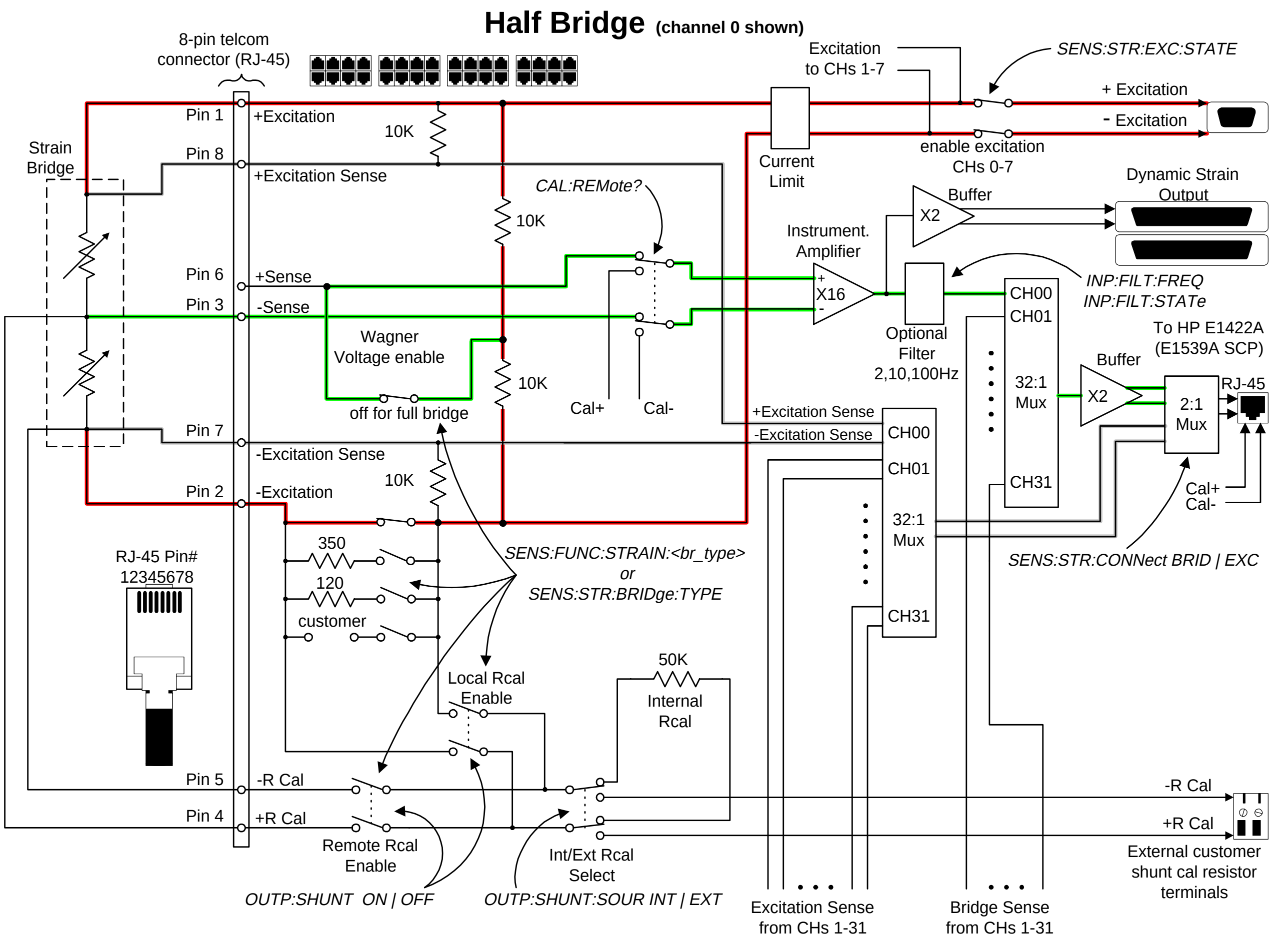


Figure 3-11. Bridge Completion for a Half Bridge Channel

Note While the diagram above shows amplifier gain in the measurement path, the measurement values returned by these channels are corrected by the HP E1422A's DSP chip (Digital Signal Processor) to reflect the actual value at the user input terminal. The only time you need to consider gain is when the input voltage times the gain would overload the A/D range chosen with a SENS:FUNC:... <range>,(<ch_list>) command. For example, with a gain of 32, any input voltage greater than 0.5V would cause an overload reading even on the highest A/D range (16V).

The Full Bridge configuration

Figure 3-12 shows the connections to the 8-pin telecom connector for a full bridge configuration. It also shows a simplified schematic of the bridge completion settings for a full bridge channel.

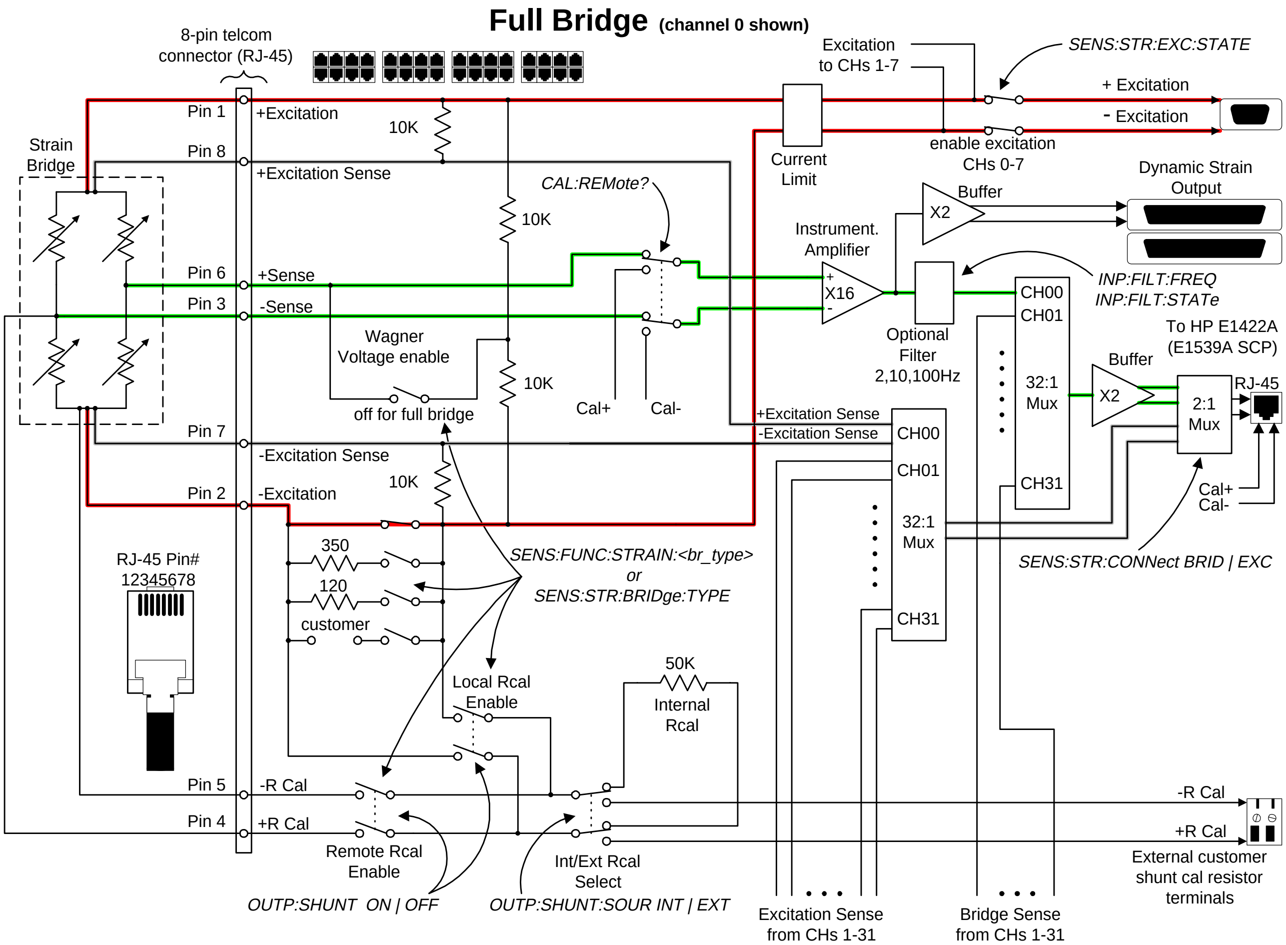


Figure 3-12. Bridge Completion for a Full Bridge Channel

Note While the diagram above shows amplifier gain in the measurement path, the measurement values returned by these channels are corrected by the HP E1422A's DSP chip (Digital Signal Processor) to reflect the actual value at the user input terminal. The only time you need to consider gain is when the input voltage times the gain would overload the A/D range chosen with a SENS:FUNC:... <range>,(<ch_list>) command. For example, with a gain of 32, any input voltage greater than 0.5V would cause an overload reading even on the highest A/D range (16V).

Connecting to the HP E1529A's Dynamic Strain Ports

The HP E1429A has two 37-pin connectors that provide wideband amplified outputs from each strain bridge signal that allow you to connect to a high-speed ADC-per-channel instrument like the HP E1432A or HP E1433A to capture dynamic strain events.

While an instruments like the HP E1432A and HP E1433A can measure signals from the HP E1529A, an HP E1422A is still required to control the HP E1529A's bridge configuration, calibration, and self-test functions. One HP E1422A can control up-to 16 HP E1529As. Figure 3-13 shows the general interconnection layout for an HP E1432A. Contact your HP representative for ordering information for this cable. A pin-out diagram for the "Buffered Output" port connectors is silk-screened onto the HP E1529A's top cover.

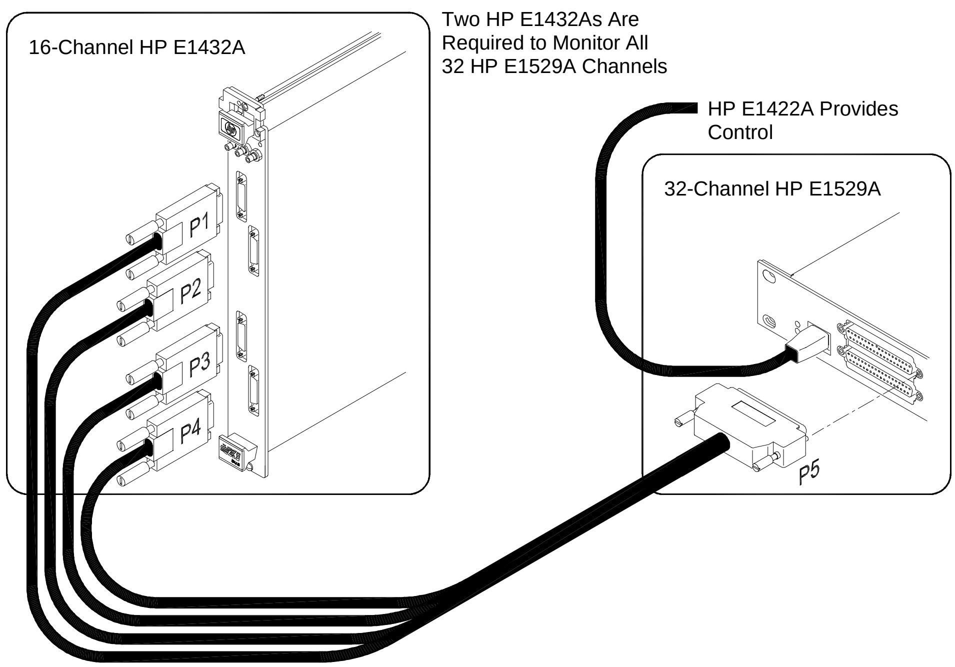


Figure 3-13. HP E1432A to HP E1529A Connection

Remote Strain Channel Addressing

Figure 3-14 shows the relationship between SCP positions and Remote Channel Addressing through the HP E1539A SCP (see Figure 2-1 on page 34 to compare with On-Board Channel Addressing). Not all SCP positions need to contain HP E1539As. You can if you need, mix HP E1539As and other analog sense, source, and digital I/O SCPs.

Channels measured through Remote Signal Conditioning Units like the HP E1529A Remote Strain Conditioning Unit are addressed with 5 digit channels specifiers rather than the traditional on-board channel's 3 digit specifier. Both 3 and 5 digit specifier start with a "1". This is the SCPI "card number" digit and is retained in the HP E1422A for SCPI compatibility. The next 2 digits complete the specification of an on-board channel. When used in a 5 digit remote multiplexed channel specifier, the first 3 digits mean the same as in the on-board specifier. Digits 2 and 3 specify the HP E1539A SCP sense channel that is connected to a particular Remote Strain Conditioning Unit. Only the first two on-board channels are ever specified with the HP E1539A Remote Channel SCP. So, digits 2 and 3 will specify channels 00, 01, 08, 09, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, or 57. This allows the HP E1422A to address up-to 16 HP E1529As. Digits 4 and 5 specify one of 32 channels on the RSCU and can range from 00 to 31.

Example channel addresses (shown in SCPI channel list syntax), see Figure 3-14 also:

(@10000)	*chan 0 on E1529A connected to on-board chan 0 (E1539A in SCP position 0).*
(@10100)	*chan 0 on E1529A connected to on-board chan 1 (E1539A in SCP position 0).*
(@14824)	*chan 24 on E1529A connected to on-board chan 48 (E1539A in SCP position 6).*

Of course, in the Scan List, the channel list syntax allows a range of channels to be specified, here are some examples:

(@10000:10131)	*channels 0 to 31 on each of the two E1529As connected to on-board channels 0 and 1 (E1539A in SCP position 0). This is 64 Chs*
(@12400:12415)	*channels 0 to 15 on the E1529A connected to on-board channel 24 (E1539A in SCP position 3).*
(@10000:10131,12400:12415)	*combined previous two examples into a single scan list to show combining ranges.*

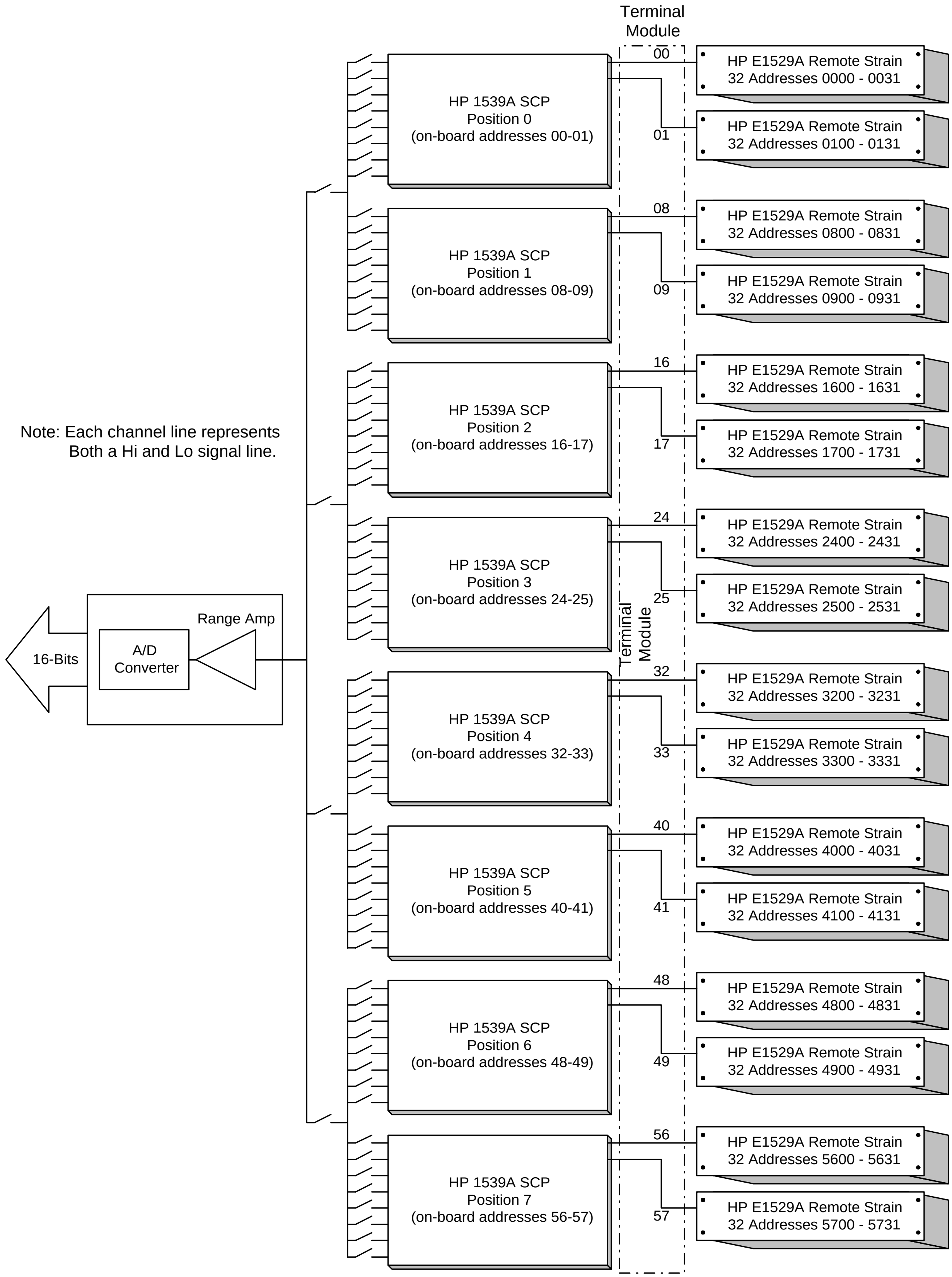


Figure 3-14. Remote Strain Channel Addressing

Programming for Remote Strain Measurement

This programming section is focused exclusively on programming the HP E1422A and HP E1529A for remote strain measurement. For more general HP E1422A programing see Chapter 4 “Programming the HP E1422A for Data Acquisition and Control”

Power-on and *RST Configuration

Some of the programming operations that follow may already be set after Power-on or after a *RST command. Where these default settings coincide with the configuration settings you require, you do not need to execute a command to set them. These are the default settings:

- No channels defined in scan list
- Programmable SCPs configured to their Power-on defaults.
- All analog input channels linked to the EU conversion for voltage
- ARM:SOURce IMMediate
- TRIGger:SOURce TIMer
- TRIGger:COUNt 1
- TRIGer:TIMer .010 (10 msec)
- FORMat ASC,7 (ASCII)
- SENSe:DATA:FIFO:MODE BLOCking

- The Defaults for the STRain Subsystem when SENS:FUNC:STRain is selected will be:
  -- Unstrained voltage for all strain channels is assumed to be zero
  -- Gage factor for all strain channels is assumed to be 2.
  -- Excitation voltage for all strain channels is assumed to be 1.0E6 (must be changed to the actual value to make reasonable measurements).

- The default for the HP E1529A strain configuration switches is:
  -- Full Bridge (FBEN) on all 32 Channels (SENS:STR:BRID FBEN
  -- Bridge output sensed.

Description of Strain Measurement

This section describes the three ways to make strain measurements with the HP E1529A. It includes references to SCPI commands as well as command sequences to perform the strain measurements described.

CALibration First

To make proper measurements, *CAL? and CAL:REMote? should have been done first. Perform *CAL? and CAL:REMote? before making important measurement runs, or if the temperature of the instrument’s environment has changed significantly. Remember, the accuracy specifications given in Appendix A on page 369 depend on recent *CAL? and CAL:REMote? operations.

Measure Strain Using Built-in Strain EU Conversion

This method lets the HP E1422A convert the strain bridge readings to units of microstrain (μ∈) before they are stored in the CVT and/or FIFO, or accessed by algorithms. There is no speed penalty and there is significant convenience in allowing the HP E1422A to make the Engineering Unit conversion to strain. In fact this is considered the "normal" HP E1422A measurement method.

When the command SENSe:FUNC:STRain:<bridge_type> is sent, the specified bridge type is configured by switches in each HP E1529A, the channel inputs are connected to the bridge outputs (see Figure 3-10 through Figure 3-12 starting on page67), and when the INIT command is sent, bridge voltage readings are automatically converted to strain before being stored into the FIFO buffer and/or CVT (current value table).

Before the E1422 can convert a channel's bridge output voltage reading to strain, the gage factor, the excitation voltage, and the unstrained reference voltage for that channel must be known.

You provide the above information to the E1422; below are the methods/commands to do so:

1. The gage factor default is 2.00 for each channel. To change any channel's gage factor value, use the SENSe:STRain:GFACtor command.

2. The unstrained reference voltage default value is 0.0 on each channel. There are two ways to change any channel's value.

   a. Use the MEAS:VOLTage:UNSTrained? command (recommended), which will take an average of 32 voltage readings on each specified channel and save the values internally for later use by the strain EU conversion process. When using this method, any loaded algorithms are not executed to avoid putting extraneous readings into the FIFO buffer. The voltage readings are also sent to the FIFO buffer in case you want to review them.

   b. Measure the voltage directly using the following series of commands:

      ROUTe:SEQ:DEFine (input the list of channels to measure)

      SENSe:FUNC:VOLT (set measurement to voltage)

      INIT (take the measurement)

      SENS:DATA:FIFO? (read the data)

   Next, the unstrained voltage values read in above must be sent back to the E1422A's EU conversion routine by using the command: SENS:STRain:UNST <voltage value>,channel list

Note If an algorithm is loaded while method "b" is used, the fifo may contain more than just the unstrained voltage readings. It is up to the user to obtain the correct data and input it into the E1422.

3. The power-on and *RST excitation voltage value is 1.0E6; this value was chosen purposely so that obviously bad readings would result if this value was not changed to the true excitation voltage. You MUST change this value to get reasonable reading values. There are two ways to change any channel’s value.

   a. Use the MEAS:VOLTage:EXCitation? command (recommended), which will take an average of 32 voltage readings on each specified channel(s) and save the value(s) internally for later use by the strain EU conversion process. When using this method, any loaded algorithm(s) are not executed to avoid putting extraneous values into the FIFO buffer. The voltage readings are also sent to the FIFO buffer in case you want to review them.

   b. Measure the voltage directly using the following series of commands:

      ROUTe:SEQ:DEFine (input the list of channels to measure)

      SENSe:FUNC:VOLT (sets measurement to voltage)

      INIT (assuming trigger system defaults, starts single scan)

      SENS:DATA:FIFO? (reads the data)

   Next, the excitation voltage values read in above must be sent back to the E1422A’s EU conversion routine by using the command: SENS:STRain:EXC <voltage value>,(@<channel>)

Note If an algorithm is loaded while method "b" is used, it will execute and may place values in the FIFO in addition to the unstrained voltage readings. It is up to the user to obtain the correct data and input it into the E1422.

Figure 3-15 shows the sequence of commands to measure remote strain channels using the built-in strain Engineering Unit Conversion routines.

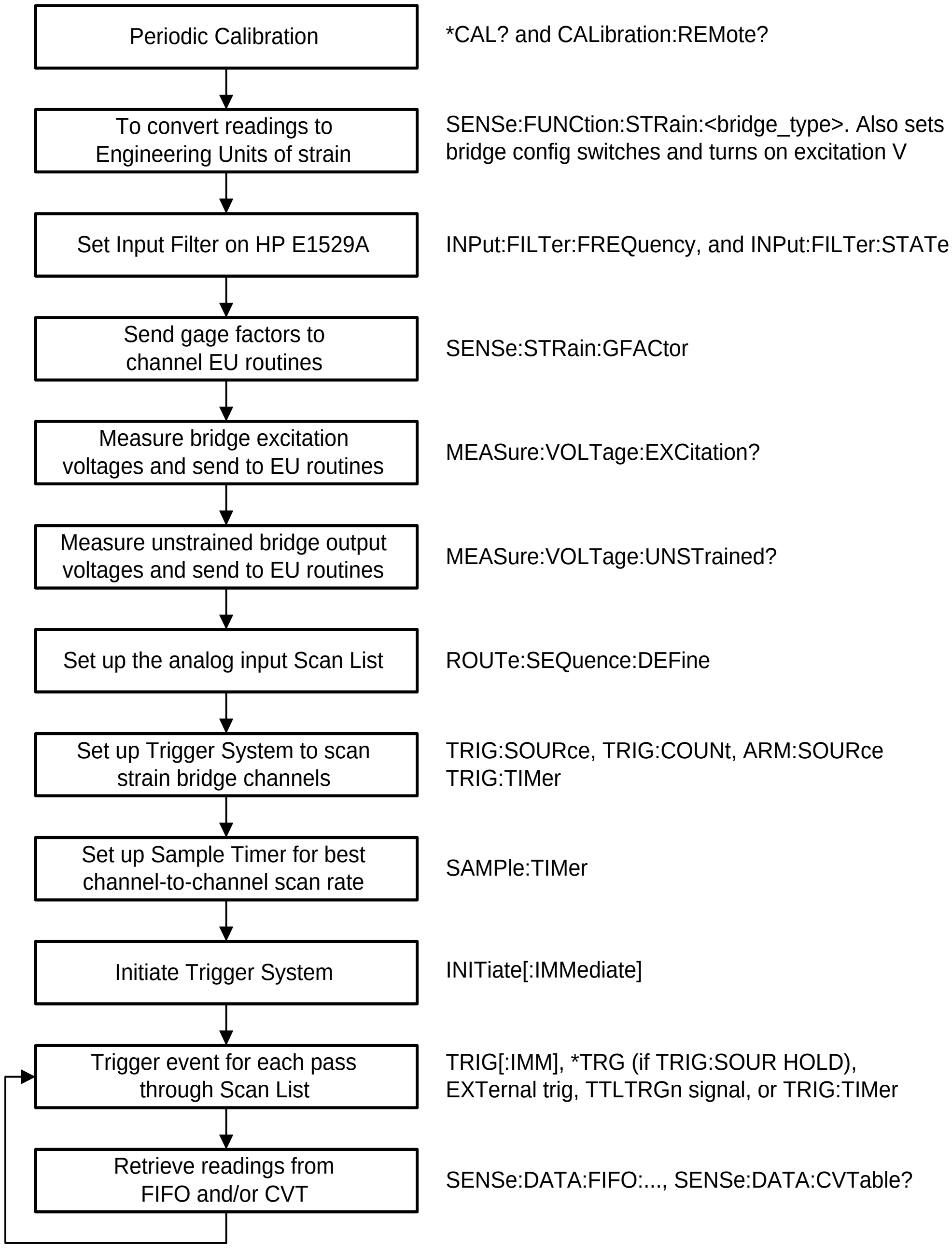


Figure 3-15. Sequence for Built-in Strain EU Conversion

Built-in EU Conversion Command Sequence

Here is an example VXIplug&play command sequence. Note that this in not executable, it's been simplified for easier reading. The C++ example source file (euseq.cpp) is on the CD supplied with your instrument. View the readme.txt file provided with the VXIplug&play driver for example program file location.

```
 /* set Engineering Units (function) to strain */
 errStatus = hpe1422_cmd(sessn,"sens:func:str:hben auto,(@10000:10003)");
 errStatus = hpe1422_cmd(sessn,"sens:func:str:fben auto,(@10004:10007)");


 /* optionally set HP E1529A input filters */
 errStatus = hpe1422_cmd(sessn,"input:filter:frequency 10,(@10000:10007)");
 errStatus = hpe1422_cmd(sessn,"input:filter:state ON,(@10000:10007)");


 /* send gage factors to channel EU converaion routines */
 errStatus = hpe1422_cmd(sessn,"sense:strain:gfactor 2,(@10000:10003)");
 errStatus = hpe1422_cmd(sessn,"sense:strain:gfactor 2.5,(@10004:10007)");


 /* measure the excitation voltage at each bridge. The values go to the
    channel EU conversion as well as the FIFO. We'll clear the FIFO */
 errStatus = hpe1422_cmdInt16_Q(sessn,"meas:volt:excitation? (@10000:10007)",
&result16);
 errStatus = hpe1422_cmd(sessn,"sense:data:fifo:reset"); /* throw away exc
readings */


 /* measure the unstrained bridge voltage at each bridge. The values go to the
    channel EU conversion as well as the FIFO. We'll clear the FIFO */
 errStatus = hpe1422_cmdInt16_Q(sessn,"meas:volt:unstrained? (@10000:10007)",
&result16);
 errStatus = hpe1422_cmd(sessn,"sense:data:fifo:reset"); /* throw away exc
readings */


 /* set up the scan list to include the strain channels to measure */
 errStatus = hpe1422_cmd(sessn,"route:sequence:define (@10000:10007)");


 /* set up the trigger system to make one scan for each trigger.
    Note that the default is one scan per trigger and trigger source
    is TIMer, so we only have to INITiate the trigger system to
    take readings. */
 errStatus = hpe1422_cmd(sessn,"trigger:count 1"); /* *RST default */
 errStatus = hpe1422_cmd(sessn,"trigger:source TIMer"); /* *RST default */
 errStatus = hpe1422_cmd(sessn,"arm:source IMMediate"); /* *RST default */


 /* set up the sample timer. This controls the channel to channel scan
    rate and can be important when channels need more than the default
    40 microsecond sample time. */
 errStatus = hpe1422_cmd(sessn,"sample:timer 40E-6"); /* *RST default */


 /* set the data FIFO format from a command module to 64-bit */
 errStatus = hpe1422_cmd(sessn,"FORM PACK,64");
```

```
/* INITiate the trigger system to execute a measurement scan */
errStatus = hpe1422_cmd(sessn,"INIT:IMMediate");


/* retrieve readings from FIFO. Notice that for each scan, we read the
   number of values in the FIFO (sens:data:fifo:count?), then apply
   that value to control the number of readings we read with the
   hpe1422_readFifo_Q() function. For continuous data aquisition, see
   Chapter 4 of the manual under "Reading Fifo Data". */
errStatus = hpe1422_cmd(sessn,"INIT:IMMediate");


/* find the number of readings present in the FIFO */
errStatus = hpe1422_cmdInt32_Q(sessn,"sense:data:fifo:count?",&result32);

/* read the values from the FIFO. count returns number actually read */
errStatus = hpe1422_readFifo_Q(sessn, result32, 65024, f64_array, &count);
```

Measure Strain Using User Specified EU Conversion

The HP E1422 measures voltage, and then applies a conversion routine (linear) supplied by the user. The user must supply the M (slope) and B (offset) of a linear M*volt + B conversion.

The DIAGnostic:CUSTom:MXB <slope>,<offset>,(@<ch_list>) command is used to supply the slope and offset for the strain conversion. To select the custom linear conversion to be used, the command SENSe:FUNCtion:CUSTom [<range>,](@<ch_list>) must be sent before starting measurements with the INIT command.

Before taking a measurement the following must be done:

1. The type of bridge connection must be specified using the [SENSe:]STRain:BRIDge[:TYPE] <select>,(@<ch_list>) command. The allowable values for <select> are: FBEN, HBEN, Q120 (quarter bridge, 120 ohms), Q350 (quarter bridge, 350 Ohms) or USER (quarter bridge, with the user supplied resistor). The power-on and *RST default setting is FBEN.

2. Configure channels to measure their strain bridge outputs rather than their excitation supply. This is done by sending the command: [SENSe:]STRain:CONNect BRIDge,(@<ch_list>)
   The power on and reset setting is BRIDge.

3. Turn on excitation voltage to the strain bridges with the SENSe:STRain:EXCitation:STATe ON,(@<ch_list>) command.

4. The linear conversion slope and offset (M and B) must be input via the DIAG:CUST:MXB command as mentioned above. The user must supply M and B, which both are functions of the excitation voltage, the unstrained reference and the gage factor.

5. The E1422 must be told to use the custom conversion when taking measurements. This is done by sending the command: SENSe:FUNC:CUSTom [<range>,](@<ch_list>)

Figure 3-16 shows the sequence of commands to convert remote measurements according to the user's own down-loaded EU conversion method..

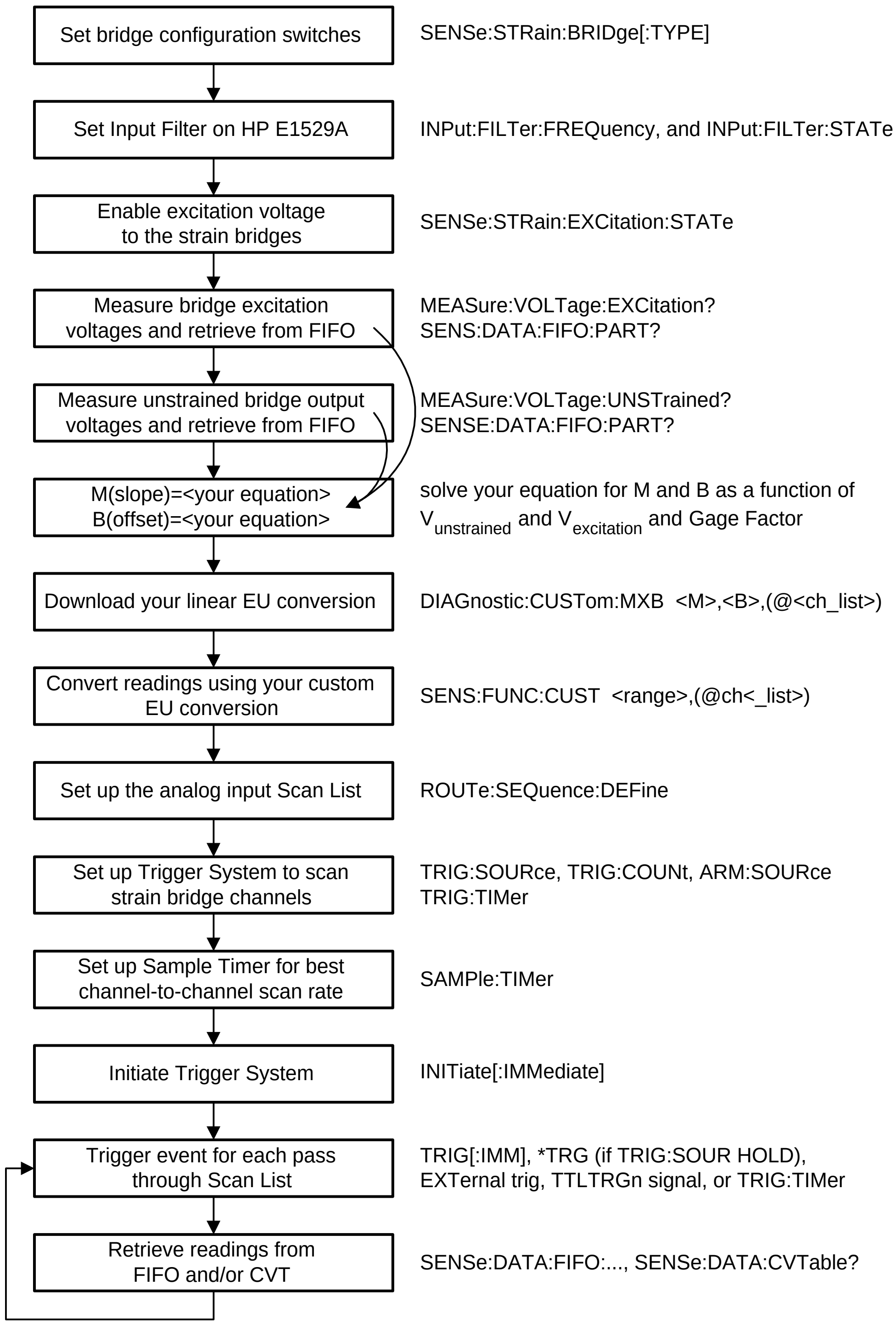


Figure 3-16. Sequence for User's Custom EU Conversion

Custom EU Conversion Command Sequence

Here is an example VXIplug&play command sequence. Note that this in not executable, it's been simplified for easier reading. The C++ example source file (mxbseq.cpp) is on the CD supplied with your instrument. View the readme.txt file provided with the VXIplug&play driver for example program file location.

```
/* set bridge configuration switches */
errStatus = hpe1422_cmd(sessn,"sens:str:bridge fben,(@10000:10007)");


/* optionally set HP E1529A input filters */
errStatus = hpe1422_cmd(sessn,"input:filter:frequency 10,(@10000:10007)");
errStatus = hpe1422_cmd(sessn,"input:filter:state ON,(@10000:10007)");


/* enable excitation voltage to strain bridges. Note that excitation is
   switched in banks of channels. So "E1529A relative" channels to switch
   are 0, 8, 16, and 24. The channel-range shown works too and is easier. */
errStatus = hpe1422_cmd(sessn,"sense:strain:excitation:state
ON,(@10000:10007)");


/* set the data FIFO format for the command module to 64-bit */
errStatus = hpe1422_cmd(sessn,"FORM PACK,64");


/* measure the excitation voltage at each bridge. The values go to the
   FIFO. We'll put them in their own array */
errStatus = hpe1422_cmdInt16_Q(sessn,"meas:volt:excitation? (@10000:10007)",
&result16);

/* read the values from the FIFO. count returns number actually read */
errStatus = hpe1422_readFifo_Q(sessn, 0, 512, exc_array, &count);


/* measure the unstrained bridge voltage at each bridge. The values go to the
   channel EU conversion as well as the FIFO. We'll clear the FIFO */
errStatus = hpe1422_cmdInt16_Q(sessn,"meas:volt:unstrained? (@10000:10007)",
&result16);

/* read the values from the FIFO. count returns number actually read */
errStatus = hpe1422_readFifo_Q(sessn, 0, 512, uns_array, &count);


/*********************** Custom EU Pre-processing **************************
 *                                                                         *
 *  Solve your custom equation for M (slope) and B (offset) as a function    *
 *  of channel Vexcitation (exc_array), Vunstrained (uns_array) and          *
 *  gage factor.                                                            *
 *  For this example, we'll just fix M and B at 2 and 0 respectively.       *
 *                                                                         *
 *  ************************************************************************
*/
M = 2;
B = 0;

/* download your derived Ms and Bs. We show downloading the same M and B to all 8
   channels. For highest accuracy, you would generate M and B for each channel to
   account for the channel-to-channel variability of the unstrained and
```

```
excitation
   values measured. */

 /* create scpi command string with M, B, and channel list */
    sprintf( cmd_str, "diag:cust:mxb %f, %f,(@%s)", M, B, "10000:10007");
 errStatus = hpe1422_cmd(sessn,cmd_str);


 /* link your derived linear EU conversion(s) to the required channels */
 errStatus = hpe1422_cmd(sessn,"sens:func:custom (@10000:10007)");


 /* set up the scan list to include the strain channels to measure bridge outputs
*/
 errStatus = hpe1422_cmd(sessn,"route:sequence:define (@10000:10007)");


 /* set up the trigger system to make one scan for each trigger.
    Note that the default is one scan per trigger and trigger source
    is TIMer, so we only have to INITiate the trigger system to
    take readings. */
 errStatus = hpe1422_cmd(sessn,"trigger:count 1"); /* *RST default */
 errStatus = hpe1422_cmd(sessn,"trigger:source TIMer"); /* *RST default */
 errStatus = hpe1422_cmd(sessn,"arm:source IMMediate"); /* *RST default */


 /* set up the sample timer. This controls the channel to channel scan
    rate and can be important when channels need more than the default
    40 microsecond sample time. */
 errStatus = hpe1422_cmd(sessn,"sample:timer 40E-6"); /* *RST default */


 /* INITiate the trigger system to execute a measurement scan */
 errStatus = hpe1422_cmd(sessn,"INIT:IMMediate");


 /* retrieve readings from FIFO. Notice that for each scan, we read the
    number of values in the FIFO (sens:data:fifo:count?), then apply
    that value to control the number of readings we read with the
    hpe1422_readFifo_Q() function. For continuous data aquisition, see
    Chapter 4 of the manual under "Reading Fifo Data". */
 errStatus = hpe1422_cmd(sessn,"INIT:IMMediate");


 /* find the number of readings present in the FIFO */
 errStatus = hpe1422_cmdInt32_Q(sessn,"sense:data:fifo:count?",&result32);

 /* read the values from the FIFO. count returns number actually read */
 errStatus = hpe1422_readFifo_Q(sessn, result32, 512, brdg_array, &count);
```

Measure Bridge Voltages and Convert to Strain

If you want to use this method, you will make voltage measurements at the strain bridges while unstrained, then again while under strain. You will also measure the excitation voltage at each bridge. Using this data as well as the gage factor, you strain conversion equations in your computer.

1. Set the measurement function to voltage with the [SENSe:]FUNC:VOLT <range>,(@<ch_list>)

2. The type of bridge connection must be specified using the [SENSe:]STRain:BRIDge[:TYPE] <select>,(@<ch_list>) command. The allowable values for <select> are: FBEN, HBEN, Q120 (quarter bridge, 120 ohms), Q350 (quarter bridge, 350 Ohms) or USER (quarter bridge, with the user supplied resistor). The power-on and *RST default setting is FBEN.

3. Configure channels to measure their strain bridge outputs rather than their excitation supply. This is done by sending the command: [SENSe:]STRain:CONNect BRIDge,(@<ch_list>)
   The power on and reset setting is BRIDge.

4. Use the MEAS:STR:UNSTrained? (@<ch_list>) command to read the voltage on each specified channels while the bridges are unstrained. This command which will take an average of 32 voltage readings on each channel and save the values to the FIFO buffer. The command returns the number of readings in the FIFO. When using this method, any loaded algorithms are not executed to avoid putting extraneous readings into the FIFO buffer.

5. Use the MEAS:STR:EXCitation? (@<ch_list>) command to sense the excitation at each of the specified bridges. This command will take an average of 32 voltage readings on each channel and save the values to the FIFO buffer. The command returns the number of values in the FIFO. When using this method, any loaded algorithm(s) are not executed to avoid putting extraneous values into the FIFO buffer.

6. Turn on excitation voltage to the strain bridges with the SENSe:STRain:EXCitation:STATe ON,(@<ch_list>) command.

7. Use the ROUTe:SEQuence:DEFine (@<ch_list>) command to define the scan list to measure the output voltage at each strain bridge. The <ch_list> specified here must match the <ch_list> specified in the two previous steps (measuring unstrained and excitation voltages).

8. Start the measurement scan with the INIT command. The default trigger system settings will execute a single measurement scan. During the scan, each channel reading is sent to the FIFO and CVT. Now you retrieve the readings and calculate the strain for each channel using the excitation, unstrained, and strained voltage values.

Figure 3-17 shows the sequence of commands to convert bridge voltage measurements to strain by post-processing.

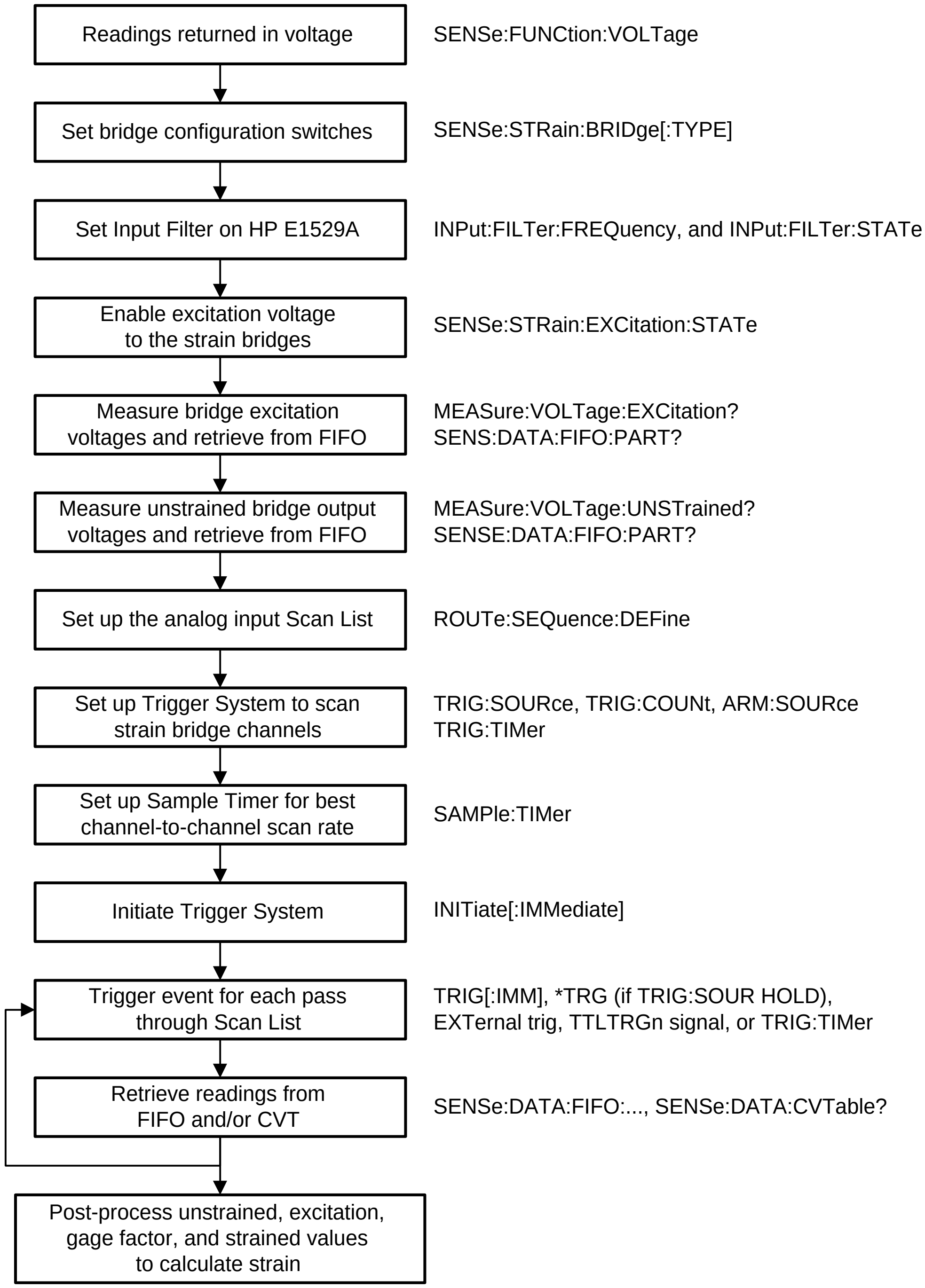


Figure 3-17. Converting Bridge Voltage Measurements to Strain

Voltage Conversion Command Sequence

Here is an example VXIplug&play command sequence. Note that this in not executable, it's been simplified for easier reading. The C++ example source file (voltseq.cpp) is on the CD supplied with your instrument. View the readme.txt file provided with the VXIplug&play driver for example program file location.

```
/* set channel function for voltage readings (autorange) */
errStatus = hpe1422_cmd(sessn,"sens:func:voltage auto,(@10000:10007)");


/* set bridge configuration switches */
errStatus = hpe1422_cmd(sessn,"sens:str:bridge fben,(@10000:10007)");


/* optionally set HP E1529A input filters */
errStatus = hpe1422_cmd(sessn,"input:filter:frequency 10,(@10000:10007)");

errStatus = hpe1422_cmd(sessn,"input:filter:state ON,(@10000:10007)");


/* enable excitation voltage to strain bridges. Note that excitation is
   switched in banks of channels. So "E1529A relative" channels to switch
   are 0, 8, 16, and 24. The channel-range shown works too and is easier. */
errStatus = hpe1422_cmd(sessn,"sense:strain:excitation:state
ON,(@10000:10007)");


/* set the data FIFO format for the command module to 64-bit */
errStatus = hpe1422_cmd(sessn,"FORM PACK,64");


/* measure the excitation voltage at each bridge. The values go to the
   FIFO. We'll put them in their own array */
errStatus = hpe1422_cmdInt16_Q(sessn,"meas:volt:excitation? (@10000:10007)",
&result16);

/* read the values from the FIFO. count returns number actually read */
errStatus = hpe1422_readFifo_Q(sessn, 0, 512, exc_array, &count);


/* measure the unstrained bridge voltage at each bridge. The values go to the
   FIFO. We'll put them in their own array */
errStatus = hpe1422_cmdInt16_Q(sessn,"meas:volt:unstrained? (@10000:10007)",
&result16);

/* read the values from the FIFO. count returns number actually read */
errStatus = hpe1422_readFifo_Q(sessn, 0, 512, uns_array, &count);


/* set up the scan list to include the strain channels to measure bridge outputs
*/
errStatus = hpe1422_cmd(sessn,"route:sequence:define (@10000:10007)");


/* set up the trigger system to make one scan for each trigger.
   Note that the default is one scan per trigger and trigger source
   is TIMer, so we only have to INITiate the trigger system to
   take readings. */
errStatus = hpe1422_cmd(sessn,"trigger:count 1"); /* *RST default */
```

```
errStatus = hpe1422_cmd(sessn,"trigger:source TIMer"); /* *RST default */
errStatus = hpe1422_cmd(sessn,"arm:source IMMediate"); /* *RST default */


/* set up the sample timer. This controls the channel to channel scan
   rate and can be important when channels need more than the default
   40 microsecond sample time. */
errStatus = hpe1422_cmd(sessn,"sample:timer 40E-6"); /* *RST default */


/* INITiate the trigger system to execute a measurement scan */
errStatus = hpe1422_cmd(sessn,"INIT:IMMediate");


/* retrieve readings from FIFO. Notice that for each scan, we read the
   number of values in the FIFO (sens:data:fifo:count?), then apply
   that value to control the number of readings we read with the
   hpe1422_readFifo_Q() function. For continuous data aquisition, see
   Chapter 4 of the manual under "Reading Fifo Data". */
errStatus = hpe1422_cmd(sessn,"INIT:IMMediate");


/* find the number of readings present in the FIFO */
errStatus = hpe1422_cmdInt32_Q(sessn,"sense:data:fifo:count?",&result32);

/* read the values from the FIFO. count returns number actually read */
errStatus = hpe1422_readFifo_Q(sessn, result32, 512, brdg_array, &count);


/********************** Strain post-processing **********************
 *                                                                  *
 *  here you take the values for excitation (exc_array), unstrained  *
 *  (uns_array), bridge output values (brdg_array), and gage foactor *
 *  and calculate individual strain values for each channel using    *
 *  your own equations.                                             *
 *                                                                  *
 ********************************************************************
 */
```

Built-in Strain Conversion Equations

When you use the HP E1422A's built-in strain conversion (SENSe:FUNCtion:STRain:<bridge_type> <range>,(@<ch_list>)), the following equations are used to convert voltage to strain.

Full Bridge Equation (bridge_type=FBEN)

This equation is used by the HP E1422A to convert bridge measurements to Engineering Units of Strain for channels specified in the command SENSe:FUNCtion:STRain:FBEN <range>,(@<ch_list>).

$$\mu Strain = (V_{measured} - V_{unstrained}) / (gFactor \times V_{excitation})$$

where:

$V_{measured}$ = measured voltage value
$V_{excitation}$ = excitation voltage
$V_{unstrained}$ = unstrained voltage
gFactor = gage factor

Half Bridge Equation (bridge_type=HBEN)

This equation is used by the HP E1422A to convert bridge measurements to Engineering Units of Strain for channels specified in the command SENSe:FUNCtion:STRain:HBEN <range>,(@<ch_list>).

$$\mu Strain = 2 \times (V_{measured} - V_{unstrained}) / (gFactor \times V_{excitation})$$

where:

$V_{measured}$ = measured voltage value
$V_{excitation}$ = excitation voltage
$V_{unstrained}$ = unstrained voltage
gFactor = gage factor

Quarter Bridge Equation (bridge_type=Q120, Q350, or USER)

This second-order equation is the used by the HP E1422A to convert bridge measurements to Engineering Units of Strain for on-board strain SCP channels only. Because HP E1529As can expand the HP E1422As strain channel count to 512, use of this non-linear strain conversion equation would require too much HP E1422A memory. Instead, a linear approximation of this equation is used. See below. For the following equations, V_i = bridge output while strained, V_u = bridge output unstrained, and V_e = excitation voltage at the bridge.

Quarter Bridge Equation for Strain SCPs only

$$\mu Strain = \frac{-4V_r}{GF(1 + 2V_r)} \quad Where V_r = \frac{V_i - V_u}{V_e}$$

Quarter Bridge Equation for HP E1529A only

$$\mu Strain = a_2 V_i^2 + a_1 V_i + a_0$$

$$Where\ a_2 = \frac{8}{GF \times V_e^2},\quad a_1 = \frac{-4(4V_u + V_e)}{GF \times V_e^2},\quad a_0 = \frac{4(V_u^2 + V_e V_u)}{GF \times V_e^2}$$

Error Analysis

Figure 3-18 compares the non-linear quarter bridge equation used for strain SCPs with the linear approximation used with the HP E1529A. Notice that while the error is independent of excitation voltage and unstrained voltage, error is quite sensitive to gage factor.

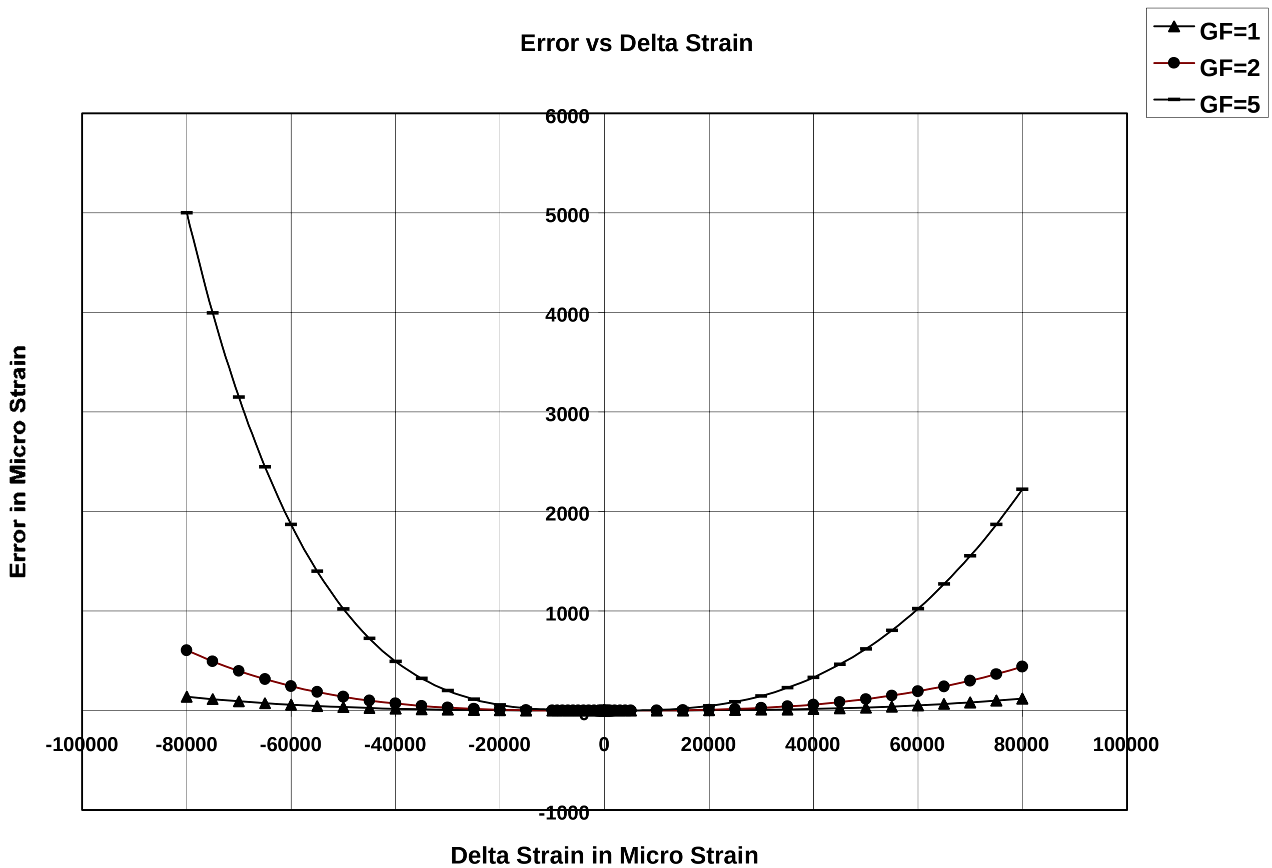


Figure 3-18. Error of Quarter Bridge Linear Approximation

Chapter 4

Programming the HP E1422A for Data Acquisition and Control

About This Chapter

The focus in this chapter is to show the HP E1422's general programming model. The programming model is basically the sequence of SCPI commands your application program will send to the HP E1422 to configure it to execute the defined Scan List and/or algorithms. This chapter contains:

Overview of the HP E1422A Multifunction DAC Module

This section describes how the HP E1422 gathers input data, executes its 'C' algorithms, and sends its output data. Figure 4-1 shows a simplified functional block diagram.

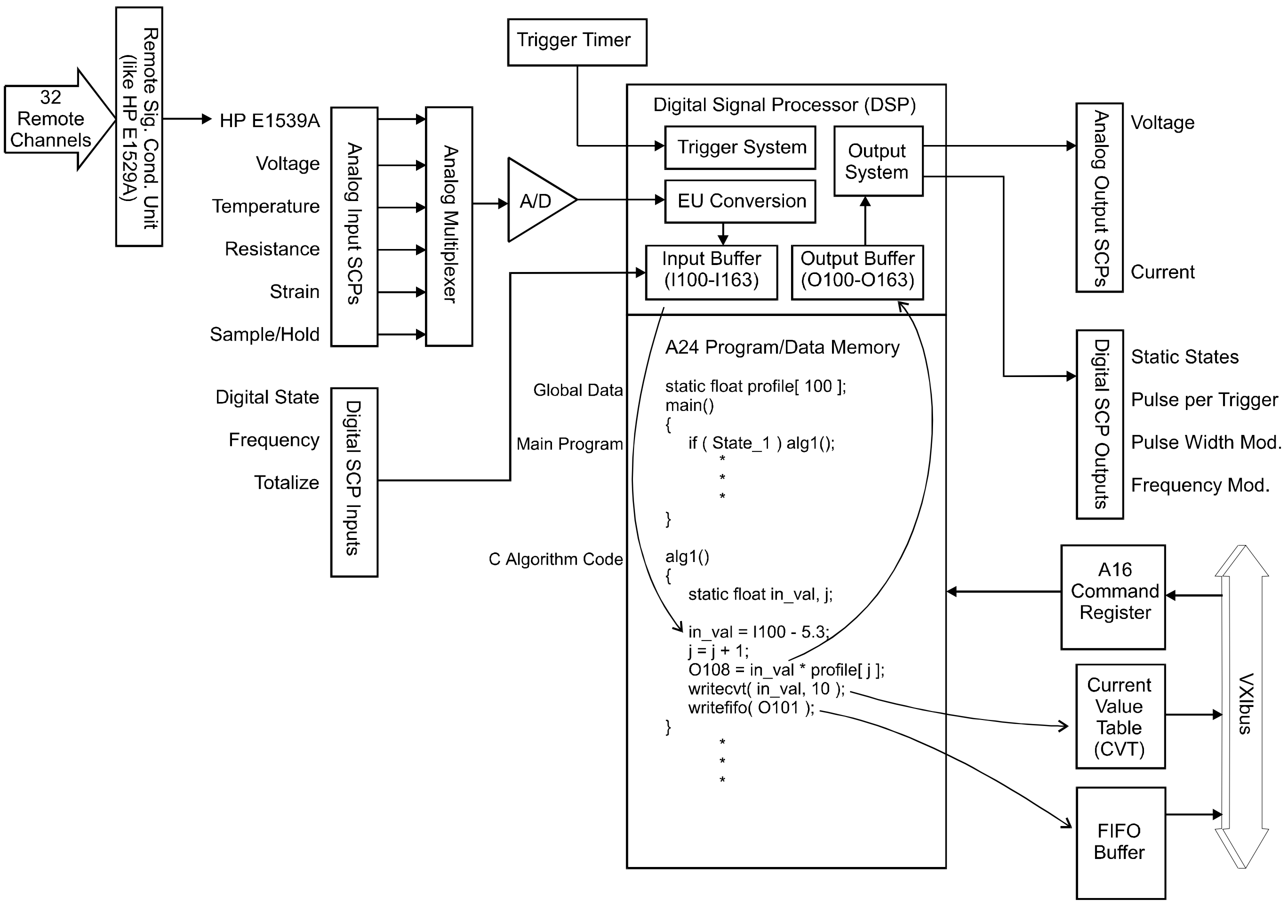


Figure 4-1. Simplified Functional Block Diagram

Multifunction DAC?

The HP E1422 is a complete data acqusition and control system on a single VXI card. It is multifunction because it uses the Signal Conditioning Plug-on(SCP) concept whereby you can mix and match your analog input/output and digital input/output channels to meet various application needs. It can be self-contained because it has local intelligence to permit the card to run stand-alone with very little interaction required from the supervisory computer.

Flexible Signal Conditioning for Input and Output

The HP E1422 has eight SCP positions with each SCP position capable of addressing up to 8 channels of input or 8 channels of output for a total of 64 channels. These 64 channels are know as the on-board channels. The 8 SCP slots can be used for any of the twenty-plus analog/digital SCP's available for the E1422 which cover most data acquisition and control needs.

Remote Multiplexing and Signal Conditioning

In addition, each SCP slot that contains an HP E1539A SCP can operate two Remote Signal Conditioning Units (RSCUs) that each externally multiplexes up-to 32 channels. These channels are known as remote channels because they are multiplexed remotely to the HP E1422. So, with 32 channels per RSCU and 16 RSCUs, the HP E1422 can make analog measurements on 512 remote channels. The upper left corner of Figure 4-1shows how Remote Signal Conditioning Units fit in.

Programmable Signal Conditioning and Engineering Unit Conversion

You configure the input and output SCP's with the SCPI and/or VXIplug&play programming. Analog SCP's are measured with the E1422's A/D. Configuring the analog SCP's includes specifying what type of Engineering Unit (EU) conversion you want for each analog input channel. For example, one channel may require a type T thermocouple conversion and another may be a resistance measurement. The on-board Digital Signal Processor(DSP) converts the voltage read across the analog input channel and applies a high-speed conversion which results in temperature, resistance, etc. Digital input SCP's perform their own conversions as configured by the SCPI language.

On-Card Scan List and/or 'C' Language Control Programming

The HP E1422 can be used as either a conventional Scan List controlled data acquisition unit with analog measurements automatically buffered and available to the supervisory computer, or the HP E1422 can execute its own internal 'C' language algorithms which can perform data acquisition and control and pass values to the supervisory computor when required. Of course both modes can be used for example when many analog data acquisition channels need to be measured using the Scan List, and one or more algorithms are needed to perhaps monitor some of the data points and make control decisions.

Operational Overview

When the Trigger System is configured and either generates its own trigger or accepts a trigger from an external source, an instrument operation cycle begins. A simplified description of cycle follows.

Acquire Input Values

All digital input SCP's latch their current input state and the A/D starts scanning the analog channels specified in the Scan List with the ROUTe:SEQuence:DEFine command or analog channels referenced by any 'C' algorithms. All measurement data as seen by the 'C' algorithms is represented as 32-bit real numbers even if the input channel is inherently integer (digital byte/word). The EU-converted numbers such as temperature, strain, resistance, volts, state, frequency, etc. when from Scan List channels (ROUT:SEQ:DEF) is stored by default in the Current Value Table (CVT), and the FIFO reading buffer. Values from analog channels referenced by ’C’ algorithms are stored in an Input Buffer and later accessed by those 'C' algorithms executing on the E1422 card.

Analog input values from channels in the Scan List, stored in the FIFO and/or CVT can be read from the HP E1422 without creating or running any ’C’ algorithms. This makes for easy traditional analog data acqusition where no control aspect is required.

Start Algorithms

Approximately 2000 lines of user-written 'C' code can be downloaded into the E1422's memory and can be split among up to 32 algorithms. HP refers to these as algorithms because an algorithm is a step-by-step procedure for solving some problem or accomplishing some end. Though the documentation continues to refer to the 'C' code as algorithms, you may think in traditional terms as each algorithm representing a 'C' function with a main() program which calls them.

The user-written 'C' algorithms execute after all analog/digital inputs have been stored in the Input Buffer. The 'C' code accesses the measurement data like constants with the names of I100-I163 (for on-board channels) and I10000-I15731 (for remote channels) representing the 32-bit real EU-converted numbers. As seen in Figure 3-1, the algorithms have access to both local and global variables and arrays. The I-variables are inherently global and accessible by any algorithm. Local variables are only visible to the particular algorithm (just like in 'C' functions). Declared global variables can be shared by any algorithm.

Communicating with Algorithms

Your application program can read or write any local or global variable in any algorithm by using SCPI syntax that actually identifies the variable by name, but a more efficient means of reading data is available through the E1422's FIFO and Current Value Table(CVT). As seen in Figure 3-1, any algorithm can write any expression or constant to the FIFO/CVT. Your application can then read the FIFO/CVT to characterize what's happening inside the E1422 and to provide an operator view of any input/output channel, variable, or constant.

Algorithms Control Output Values

Output SCP's derive their channel values from O-variables that are written by the algorithms. O100-O163 are read/write global variables that are read after all algorithms have finished executing. The 32-bit real values are converted to the appropriate units as defined by the SCPI configuration commands and written to the various output SCP's by channel number.

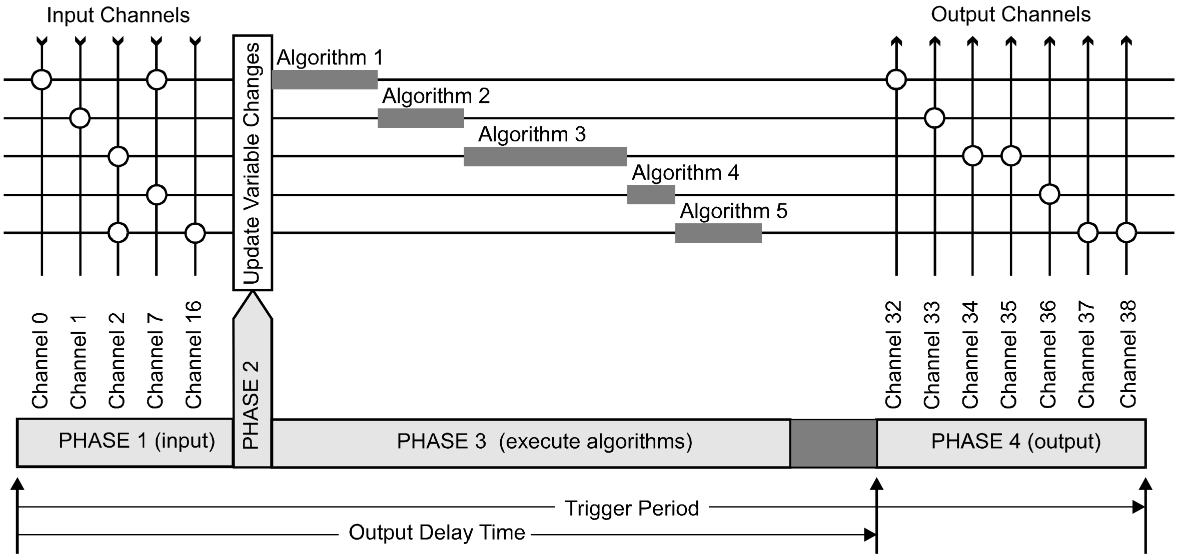


Figure 4-2. Instrument Operation Cycle Phases

Detailed Instrument Operation Cycle

Figure 4-2 illustrates the timing of all these operations and describes the E1422's input-update-execute algorithms-output phases. This cycle-based design is desirable because it results in deterministic operation of the E1422. That is, the input channels are always scanned, and the output channels are always written at pre-defined intervals. Note too that any number of input channels or output channels are accessible by any of up to 32 user-written algorithms. The algorithms are named ALG1-ALG32 and execute in numerical order.

Notice the Update Window (phase 2) illustrated in Figure 4-2. This window has a user-specified length and is used to accept and make changes to local and global variables from the supervisory computer. Up to 512 scalar or array changes can be made while executing algorithms. Special care was taken to make sure all changes take place at the same time so that any particular algorithm or group of algorithms all operate on the new changes at a user-specified time. This does not mean that all scalar and array changes have to be received during one cycle to become effective at the next cycle. On the contrary, it may take a number of cycles to download new values, especially when trying to re-write 1024 element arrays and especially when the trigger cycle time is very short.

There are multiple times between the base triggers where scalar and array changes can be accepted from the supervisory computer, and these changes are held in a holding buffer until the supervisory computer instructs the changes to take effect. These changes then take place during the Update window and take effect BEFORE algorithms start executing. The "do-update-now" signal can be sent by command(ALG:UPD) or by a

change in a digital input state(ALG:UPD:CHAN). In either case, the programmer has control over when the new changes take effect.

The E1422's ability to execute programs directly on the card and its fast execution speed give the programmer real-time response to changing conditions. And, programming the card has been made very easy to understand. HP chose C as the language used to write user programs since that language is already considered the industry standard. Choosing C allows you to write algorithms on PC's or UNIX workstations that have C compilers, so you can debug algorithms before execution on the card. The E1422 also provides good debugging tools that permit you to determine worst-case execution speed, monitor variables while running, and selectively enable/disable any of the E1422's 32 algorithms.

HP created a limited and simplified version of C since most applications need only basic operations: add, subtract, multiply, divide, scalar variables, arrays, and programming constructs. The programming constructs are limited to if-then-else to allow conditional evaluation and response to input changes. Since all algorithms have an opportunity to execute after each time-base trigger, the if-then-else constructs permit conditional skipping of cycle intervals so that some code segments or algorithms can execute at multiples of the cycle time instead of every cycle.

Looping constructs such as for or while are purposely left out of the language so that user programs are deterministic. Note that looping is not really needed for most applications since the cycle interval execution (via the trigger system) of every algorithm has inherent repeat looping. With no language looping constructs, the HP E1422's C compiler can perform a worst-case branch analysis of user programs and return the execution time for determining the minimum time-base interval. Making this timing query available allows the programmer to know exactly how much time may be required to execute any/all phases before attempting to set up physical test conditions.

Note the darker shaded portion at the end of the Execute Algorithms Phase in Figure 4-2. The conditional execution of code can cause the length of this phase to move back and forth like an accordion. This can cause undesirable output jitter when the beginning of the output phase starts immediately after the last user algorithm executes. The HP E1422's design allows the user to specify when output signals begin relative to the start of the trigger cycle. Outputs then always occur at the same time, every time.

The programming task is further made easy with this design because all the difficult structure of handling input and output channels is done automatically. This is not true of many other products that may have several ways to acquire measurement data or write results to its I/O channels. When the E1422's user-written C algorithms are compiled, input channels and output channels are detected in the algorithms and are automatically grouped and configured for the Input and Output phases as seen in Figure 4-2. Each algorithm simply accesses input channels as variables and writes to output channels as variables. The rest is handled and optimized by the Input and Output phases. You're left to think of solving your application in terms of input and output values variables rather than worrying about how to deal with each SCP's differences.

Programming Model

You configure, start, stop, and communicate with the HP E1422 using its SCPI commands and/or VXIplug&play driver functions. The module can be in one of two states; either the "idle" state, or the "running" state. The INITiate[:IMMediate] command moves the module from the "idle" state to the "running" state. We will call these two states "before INIT", and "after INIT". See Figure 4-3 for the following discussion.

Before INIT the module is in the Trigger Idle State and its DSP chip (the on-board control processor) is ready to accept virtually any of its SCPI or Common commands. At this point, you will send it commands that configure SCPs, link input channels to EU conversions, configure Remote Signal Conditioning Units, configure digital input and output channels, define a Scan List, configure the trigger system, and define control algorithms.

After INIT (and with trigger events occurring), the DSP is busy measuring input channels, executing algorithm code, sending internal algorithm values to the CVT, and updating control outputs. To insulate the DSP from commands that would interrupt its measurement scanning and/or algorithm execution, the HP E1422's driver disallows execution of most SCPI commands and VXIplug&play functions after INIT. The driver does allow certain commands that make sense while the module is scanning and running algorithms. These are the commands that read and update algorithm variables, retrieve data aquisition values from the CVT and FIFO, and return Status System values. The Command Reference Section (Chapter 6) specifies whether a command is accepted before or after INIT.

The next section in this chapter ("Executing the Programming Model") shows the programming sequence that should be followed when setting up the HP E1422 to make measurement scans and/or run algorithms.

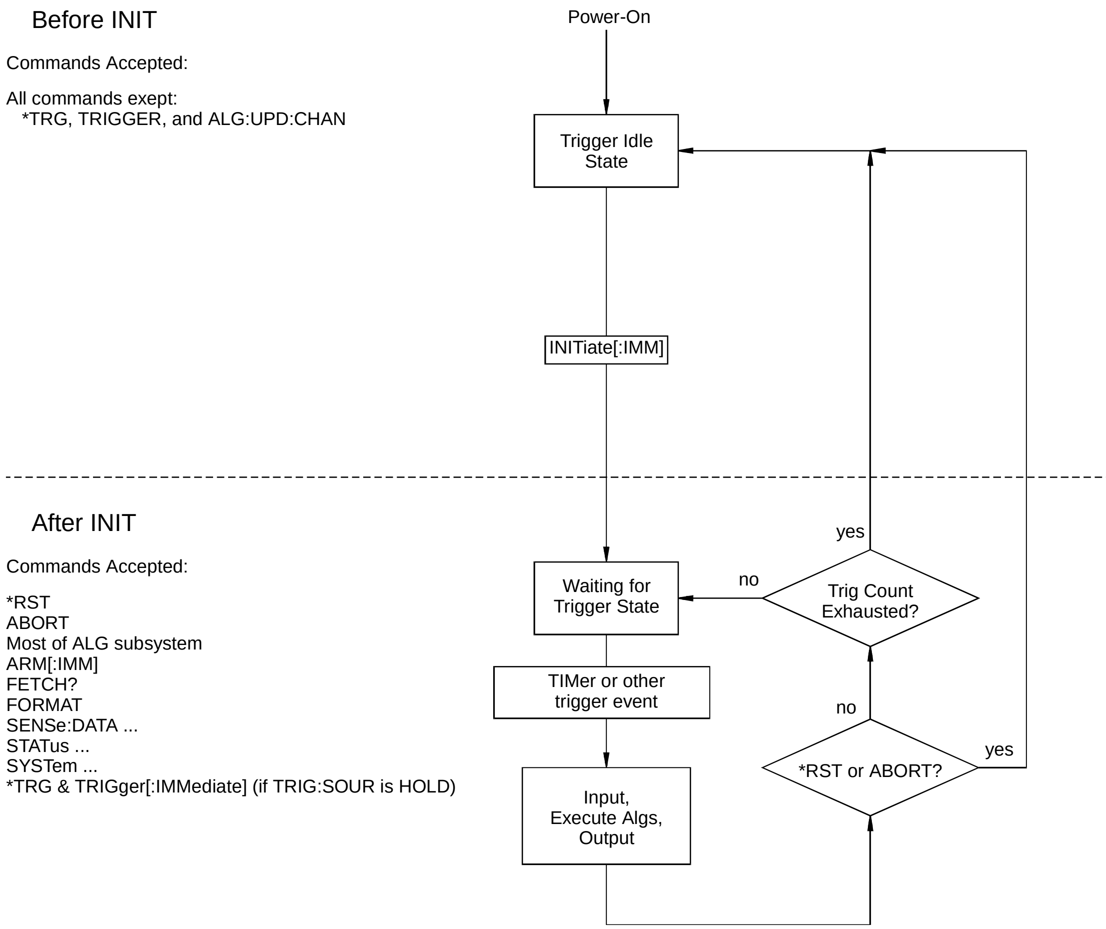


Figure 4-3. Module States

Executing the Programming Model

This section shows the sequence of programming steps that should be used for the HP E1422. Within each step, most of the available choices are shown using command sequence examples, with further details available in the Command Reference Chapter 6.

IMPORTANT! Most programming difficulties can be resolved by you if you know what's wrong. It is very important while developing your application that you execute the SYSTem:ERRor? command after each programming command. This is the only way you will know if there is a programming error. SYST:ERR? returns an error number and description (or +0, "No Error").

Power-on and *RST Default Settings

Some of the programming operations that follow may already be set after Power-on or after a *RST command. Where these default settings coincide with the configuration settings you require, you do not need to execute a command to set them. These are the default settings:

- No algorithms defined
- No channels defined in channel lists
- Programmable SCPs configured to their Power-on defaults (see individual SCP User's Manuals)
- All analog input channels linked to EU conversion for voltage
- All analog output channels ready to take values from an algorithm
- All digital I/O channels set to input static digital state
- ARM:SOURce IMMediate
- SAMPle:TIMer 40E-6 (40μsec)
- TRIGger:SOURce TIMer
- TRIGger:COUNt 1 (note that this default was chosen to make testing data aquisition scan list easier. For algorithm operation, you will probably want to change the count to INFinite.)
- TRIGer:TIMer .010 (10 msec)
- FORMat ASC,7 (ASCII)
- SENSe:DATA:FIFO:MODE BLOCking

Figure 4-4 provides a quick reference to the Programming model. Refer to this, together with the “Programming Overview Diagram” to keep an overview of the HP E1422 SCPI programming sequence. Again, where default settings are what you want, you can skip that configuration step

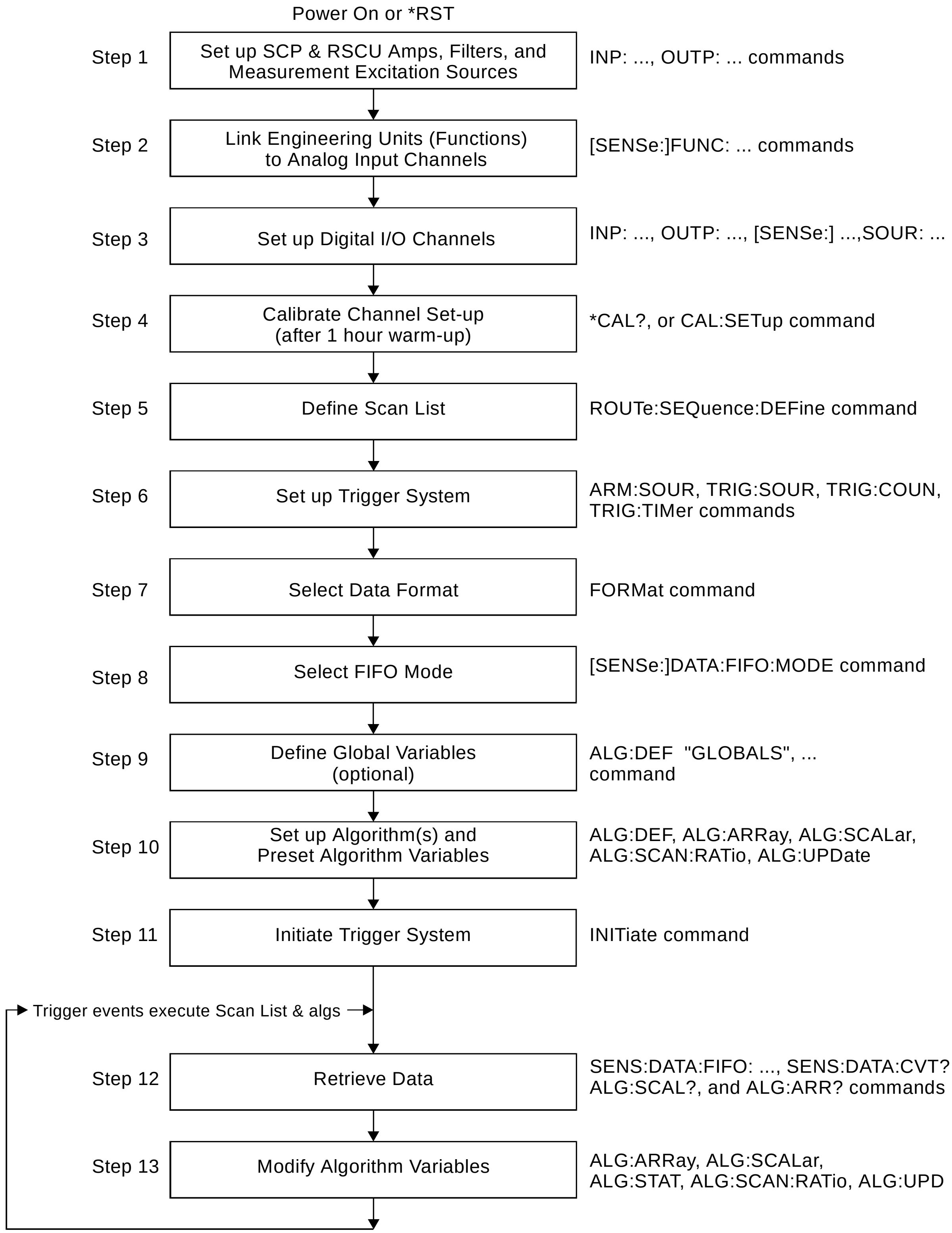
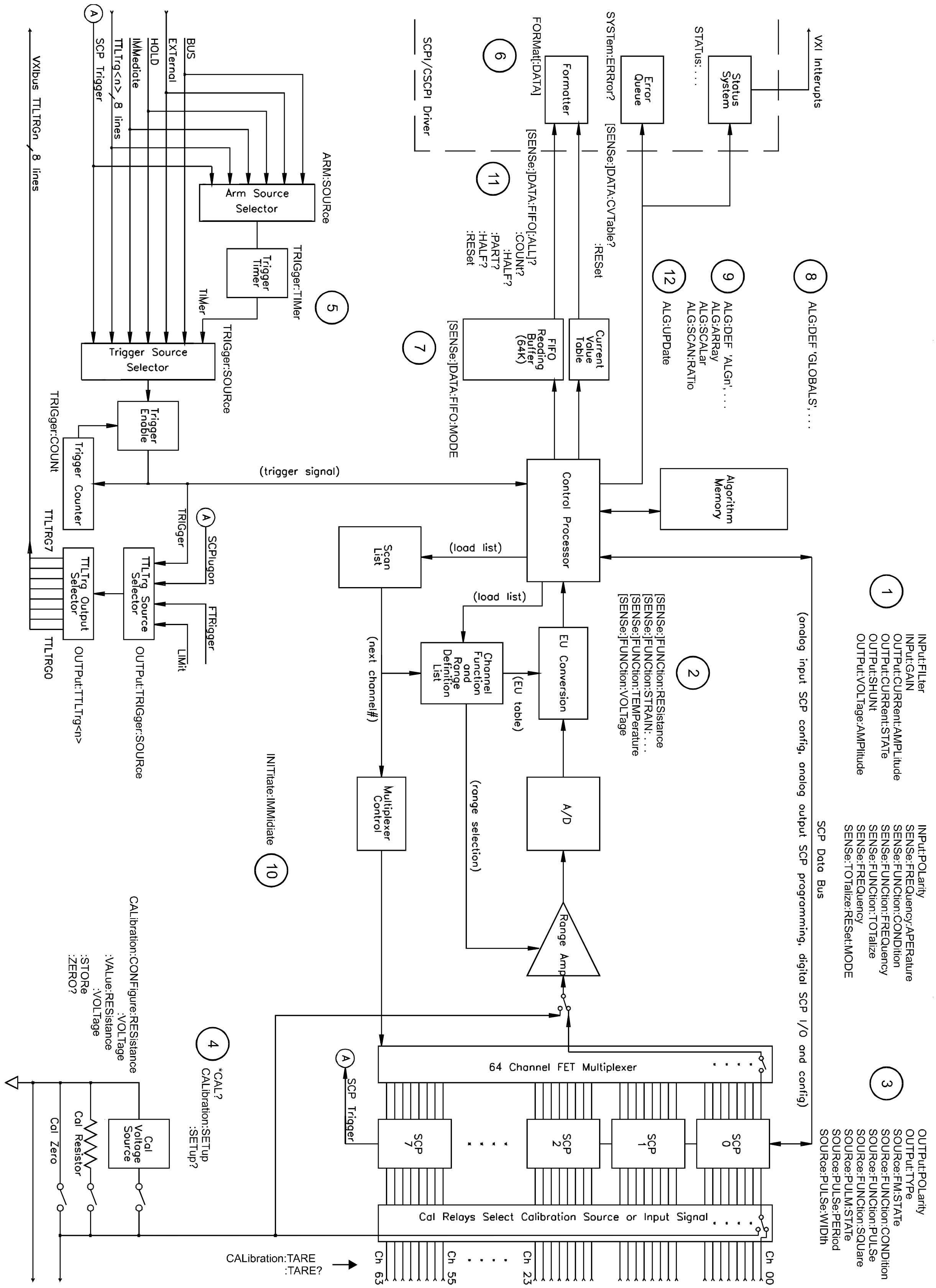


Figure 4-4. Programming Sequence

Programming Overview Diagram



Setting up Analog Input and Output Channels

This section covers configuring input and output channels to provide the measurement values and output characteristics that your algorithms need to operate.

Configuring Programmable Analog SCP Parameters

This step applies only to programmable Signal Conditioning Plug-ons such as the HP E1503 Programmable Amplifier/Filter SCP, the HP E1505 Current Source SCP, the HP E1510 Sample and Hold SCP, the HP E1511 Transient Strain SCP, and Remote Signal Conditioning Units (RSCUs, like the HP E1529A Remote Strain Conditioning Unit). See the particular SCP's User's manual to determine the gain, filter cutoff frequency, or excitation amplitude selections that it may provide. See "Programming the HP E1422A & HP E1529A for Remote Strain Measurement" on page 55. for information on the HP E1529A's programmable settings.

Setting SCP Gains

An important thing to understand about input amplifier SCPs and RSCUs is that given a fixed input value at a channel, changes in channel gain do not change the value returned from that channel. The DSP chip (Digital Signal Processor) keeps track of SCP gain and Range Amplifier settings, and "calculates" a value that reflects the signal level at the input terminal. The only time this in not true is when the SCP gain chosen would cause the output of the SCP amplifier to be too great for the selected A/D range. As an example; with SCP gain set to 64, an input signal greater than ±0.25 volts would cause an over-range reading even with the A/D set to its 16 volt range.

The gain command for SCPs with programmable amplifiers is:

INPut:GAIN <*gain*>,(@<*ch_list*>) to select SCP channel gain.

The gain selections provided by the SCP can be assigned to any channel individually or in groups. Send a separate command for each gain selection. An example for the HP E1503 programmable Amp&Filter SCP:

To set the SCP gain to 8 for channels 0, 4, 6, and 10 through 19 send:

```
INP:GAIN 8,(@100,104,106,110:119)
```

To set the SCP gain to 16 for channels 0 through 15, and to 64 for channels 16 through 23 send:

```
INP:GAIN 16,(@100:115)
INP:GAIN 64,(@116:123)
```

or to combine into a single command message:

```
INP:GAIN 16,(@100:115);GAIN 64,(@116:123)
```

Setting Filter Cutoff Frequency

The commands for programmable filters are:

INPut:FILTer[:LPASs]:FREQuency <*cutoff_freq*>,(@<*ch_list*>) to select cutoff frequency

INPut:FILTer[:LPASs][:STATe] ON | OFF,(@<*ch_list*>) to enable or disable input filtering

The cutoff frequency selections provided by the SCP can be assigned to any channel individually or in groups. Send a separate command for each frequency selection. For example:

To set 10 Hz cutoff for channels 0, 4, 6, and 10 through 19 send:

```
INP:FILT:FREQ  10,(@100,104,106,110:119)
```

To set 10 Hz cutoff for channels 0 through 15, and 100 Hz cutoff for channels 16 through 23 send:

```
INP:FILT:FREQ  10,(@100:115)
INP:FILT:FREQ  100,(@116:123)
```

or to combine into a single command message

```
INP:FILT:FREQ  10,(@100:115);FREQ  100,(@116:123)
```

By default (after *RST or at power-on) the filters are enabled. To disable or re-enable individual (or all) channels, use the INP:FILT ON | OFF, (@<ch_list>) command. For example, to program all but a few filters on, send:

INP:FILT:STAT ON,(@100:163)	*all channel's filters on (same as at *RST)*
INP:FILT:STAT OFF,(@100, 123,146,163)	*only channels 0, 23, 46, and 63 OFF*

Setting the HP E1505 Current Source SCP and HP E1518 Resistance Measurement SCP

The Current Source and Resistance Measurement SCPs supplie excitation current for resistance type measurements. These include resistance, and temperature measurements using resistance temperature sensors. The commands to control Current Source SCPs are: **OUTPut:CURRent:AMPLitude <*amplitude*>,(@<*ch_list*>)** and **OUTPut:CURRent[:STATe] <*enable*>**.

- The *amplitude* parameter sets the current output level. It is specified in units of Amps DC and for the HP E1505/E1518 SCP can take on the values 30e-6 (or MIN), and 488e-6 (or MAX). Select 488µA for measuring resistances of less than 8,000 Ohms. Select 30µA for resistances of 8,000 Ohms and above.

- The *ch_list* parameter specifies the Current Source SCP channels that will be set.

To set channels 0 through 9 to output 30 µA and channels 10 through 19 to output 488 µA:

OUTP:CURR 30e-6,(@100:109)	
OUTP:CURR 488e-6,(@110:119)	*separate command per output level*

or to combine into a single command message:

```
OUTP:CURR  30e-6,(@100:109);CURR  488e-6,(@110:119)
```

NOTE The OUTPut:CURRent:AMPLitude command is only for programming excitation current used in resistance measurement configurations. It is does not program output DAC SCPs like the HP E1532.

Setting the HP E1511 Strain Bridge SCP Excitation Voltage

The HP E1511 Strain Bridge Completion SCP has a programmable bridge excitation voltage source. The command to control the excitation supply is **OUTPut:VOLTage:AMPLitude <*amplitude*>,(@<*ch_list*>)**

- The <*amplitude*> parameter can specify 0, 1, 2, 5, or 10 volts for the HP E1511's excitation voltage.
- The <*ch_list*> parameter specifies the SCP and bridge channel excitation supply that will be programmed. There are four excitation supplies in each HP E1511.

To set the excitation supplies for channels 0 through 3 to output 2 volts:

```
OUTP:VOLT:AMPL  2,(@100:103)
```

NOTE The OUTPut:VOLTage:AMPLitude command is only for programming excitation voltage used measurement configurations. It is does not program output DAC SCPs like the HP E1531.

Linking Input Channels to EU Conversion

This step links each of the module's channels to a specific measurement type. For analog input channels this "tells" the on-board control processor which EU conversion to apply to the value read on any channel. The processor is creating a list of conversion types vs. channel numbers.
The commands for linking EU conversion to channels are:

[SENSe:]FUNCtion:RESistance <*excite_current*>,[<*range*>,](@<*ch_list*>) for resistance measurements

[SENSe:]FUNCtion:STRain:… <*excite_current*>,[<*range*>,](@<*ch_list*>) for strain bridge measurements

[SENSe:]FUNCtion:TEMPerature <type>,<sub_type>,[<range>,](@<ch_list>) for temperature measurements with thermocouples, thermistors, or RTDs

[SENSe:]FUNCtion:VOLTage <*range*>,(@<*ch_list*>) for voltage measurements

[SENSe:]FUNCtion:CUSTom <*range*>,(@<*ch_list*>) for custom EU conversions.

NOTE At Power-on and after *RST, the default EU Conversion is autorange voltage for all 64 channels.

Linking Voltage Measurements

To link channels to the voltage conversion send the [SENSe:]FUNCtion:VOLTage [<*range*>,] (@<*ch_list*>) command.

- The *ch_list* parameter specifies which channels to link to the voltage EU conversion.
- The optional *range* parameter can be used to choose a fixed A/D range. Valid values are: .0625, .25, 1, 4, 16, or AUTO. When not specified, the module uses auto-range (AUTO).

To set channels 0 through 15 to measure voltage using auto-range:

```
SENS:FUNC:VOLT  AUTO,(@100:115)
SENS:FUNC:VOLT  AUTO,(@10000:10131)  first 64 RSCU channels
```

To set channels 16 and 24 to the 16 volt range, and 32 through 47 to the .0625 volt range:

```
SENS:FUNC:VOLT  16,(@116,124)
SENS:FUNC:VOLT  .625,(@132:147)          must send a command per range
```

or to send both commands in a single command message:

```
SENS:FUNC:VOLT  16,(@116,124);VOLT  .0625,(@123:147)
```

NOTE When using manual range in combination with amplifier SCPs, the EU conversion will try to return readings which reflect the value of the input signal. However, it is up to you to choose range values that will provide good measurement performance (avoiding over-ranges and selecting ranges that provide good resolution based on the input signal). In general, measurements can be made at full speed using auto-range. Auto-range will choose the optimum A/D range for the amplified signal level.

Linking Resistance Measurements

To link channels to the resistance EU conversion send the [SENSe:]FUNCtion:RESistance <*excite_current*>,[<*range*>,](@<*ch_list*>) command.

Resistance measurements assume that there is at least one Current Source SCP installed (eight current sources per SCP). See Figure 4-5

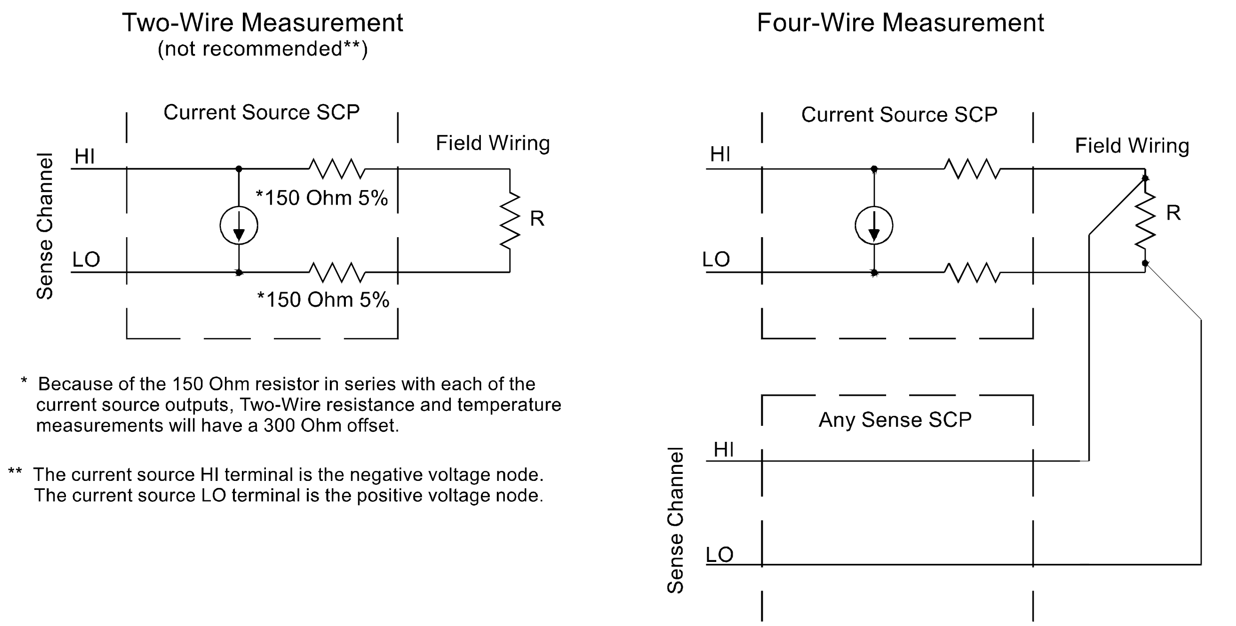


Figure 4-5. Resistance Measurement Sensing

- The *excite_current* parameter is used only to tell the EU conversion what the Current Source SCP channel is now set to. *Excite_current* is specified in Amps DC and the choices for the HP E1505 SCP are 30e-6 (or MIN) and 488e-6 (or MAX). Select 488µA for measuring resistances of less than 8,000 Ohms. Select 30µA for resistances of 8,000 Ohms and above.

- The optional *range* parameter can be used to choose a fixed A/D range. When not specified (defaulted), the module uses auto-range.

- The *ch_list* parameter specifies which channel(s) to link to the resistance EU conversion. These channels will sense the voltage across the unknown resistance. Each can be a Current Source SCP channel (a two-wire resistance measurement) or a sense channel separate from the Current Source SCP channel (a four-wire resistance measurement). See Figure 4-5 for diagrams of these measurement connections.

To set channels 0 through 15 to measure resistances greater than 8,000 Ohms and set channels 16, 20, and 24 through 31 to measure resistances less than 8K (in this case paired to current source SCP channels 32 through 57):

OUTP:CURR:AMPL 30e-6, (@132:147)
set 16 channels to output 30µA for 8KΩ or greater resistances
SENS:FUNC:RES 30e-6, (@100:115)
link channels 0 through 15 to resistance EU conversion (8KΩ or greater)
OUTP:CURR:AMPL 488e-6, (@148,149,150:157)
set 10 channels to output 488µA for less than 8KΩ resistances

SENS:FUNC:RES 488e-6, (@116,120,124:132)

link channels 16, 20 and 24 through 32 to resistance EU conversion (less than 8KΩ)

Linking Temperature Measurements

To link channels to temperature EU conversion send the [SENSe:]FUNCtion:TEMPerature <*type*>, <*sub_type*>, [<*range*>,](@<*ch_list*>) command.

- The *ch_list* parameter specifies which channel(s) to link to the temperature EU conversion.
- The *type* parameter specifies RTD, THERmistor, or TC (for ThermoCouple)
- The optional *range* parameter can be used to choose a fixed A/D range. When not specified (defaulted), the module uses auto-range.

RTD and Thermistor Measurements

Temperature measurements using resistance type sensors involve all the same considerations as resistance measurements discussed in the previous section. See the discussion of Figure 4-5 in "Linking Resistance Measurements".

For resistance temperature measurements the *sub_type* parameter specifies:

- For RTDs; "85" or "92" (for 100 Ohm RTDs with 0.003**85** or 0.003**92** Ohms/Ohm/Degree C temperature coefficients respectively)
- For Thermistors; 2250, 5000, or 10000 (the nominal value of these devices at 25 degrees C)

NOTES

1. Resistance temperature measurements (RTDs and THERmistors) require the use of Current Source Signal Conditioning Plug-Ons. The following table shows the Current Source setting that must be used for the following RTDs and Thermistors:

Required Current Amplitude	Temperature Sensor Types and Subtypes
MAX (488µA)	RTD,85 | 92 and THER,2250
MIN (30µA)	THER,5000 | 10000

2. *sub_type* values of 2250, 5000, and 10000 refer to thermistors that match the Omega 44000 series temperature response curve. These 44000 series thermistors have been selected to match the curve within 0.1 or 0.2°C.

To set channels 0 through 15 to measure temperature using 2,250 Ohm thermistors (in this case paired to current source SCP channels 16 through 31):

OUTP:CURR:AMPL 488e-6,(@116:131)
set excite current to 488µA on current SCP channels 16 through 31
SENS:FUNC:TEMP THER, 2250, (@100:115)
link channels 0 through 15 to temperature EU conversion for 2,250Ω thermistor

To set channels 32 through 47 to measure temperature using 10,000 Ohm thermistors (in this case paired to current source SCP channels 48 through 63):

OUTP:CURR:AMPL 30e-6,(@148:163)
set excite current to 30µA on current SCP channels 48 through 63
SENS:FUNC:TEMP THER, 10000, (@132:147)
link channels 32 through 47 to temperature EU conversion for 10,000Ω thermistor

To set channels 48 through 63 to measure temperature using 100 Ohm RTDs with a TC of .00385 Ohm/Ohm/°C (in this case paired to current source SCP channels 32 through 47):

OUTP:CURR:AMPL 488e-6,(@132:147)
set excite current to 488µA on current SCP channels 32 through 47
SENS:FUNC:TEMP RTD, 85, (@148:163)
link channels 48 through 63 to temperature EU conversion for 100Ω RTDs with .00385 TC.

Thermocouple Measurements

Thermocouple measurements are voltage measurements that the EU conversion changes into temperature values based on the *sub_type* parameter and latest reference temperature value.

- For Thermocouples the *sub_type* parameter can specify CUSTom, E, EEXT, J, K, N, R, S, T (CUSTom is pre-defined as Type K, no reference junction compensation. EEXT is the type E for extended temperatures of 800°F or above).

To set channels 32 through 40 to measure temperature using type E thermocouples:

SENS:FUNC:TEMP TC, E, (@132:140)
(see following section to configure a TC reference measurement)

Thermocouple Reference Temperature Compensation

The isothermal reference temperature is required for thermocouple temperature EU conversions. The Reference Temperature Register must be loaded with the current reference temperature before thermocouple channels are scanned. The Reference Temperature Register can be loaded two ways:

1. By measuring the temperature of an isothermal reference junction

during an input scan.

2. By supplying a constant temperature value (that of a controlled temperature reference junction) before a scan is started.

Setting up a Reference Temperature Measurement

This operation requires two commands, the [SENSe:]REFerence command and the [SENSe:]REFerence:CHANnels command.

The [SENSe:]REFerence <*type*>, <*sub_type*>,[<*range*>,](@<*ch_list*>) command links channels to the reference temperature EU conversion.

- The *ch_list* parameter specifies the sense channel that you have connected to the reference temperature sensor.

- The *type* parameter can specify THERmistor, RTD, or CUSTom. THER and RTD, are resistance temperature measurements and use the on-board 122 µA current source for excitation. CUSTom is pre-defined as a Type E thermocouple which has a thermally controlled ice point reference junction.

- The *sub_type* parameter must specify:

  -- For RTDs; "85" or "92" (for 100 Ohm RTDs with 0.003**85** or 0.003**92** Ohms/Ohm/Degree C temperature coefficients respectively)

  -- For Thermistors; only "5000" (See previous note on page 105)

  -- For CUSTom; only "1"

- The optional *range* parameter can be used to choose a fixed A/D range. When not specified (defaulted), or set to AUTO, the module uses auto-range.

Reference Measurement Before Thermocouple Measurements

At this point we are going to introduce you to the concept of the HP E1422's Scan List. As you define each algorithm, the HP E1422 places any reference to an analog input channel into the Scan List (this is in addition to those channels specified by the ROUT:SEQ:DEF command, see "Defining an Analog Input Scan List" on page 116). When you run algorithms, the scan list tells the HP E1422 which analog channels to scan during the Input Phase.

The [SENSe:]REFerence:CHANnels (@<*ref_chan*>),(@<*meas_ch_list*>) is used to place the <*ref_chan*> channel in the scan list before the related thermocouple measuring channels in <*meas_chan*>. Now when analog channels are scanned, the HP E1422 will include the reference channel in the scan list and will scan it before the specified thermocouples are scanned. The reference measurement will be stored in the Reference Temperature

Register. The reference temperature value is applied to the thermocouple EU conversions for thermcouple channel measurements that follow.

A Complete Thermocouple Measurement Command Sequence

The command sequence performs these functions:

- Configures reference temperature measurement on channel 15.
- Configures thermocouple measurements on channels 16 through 23.
- Instructs the HP E1422 to add channel 15 to the Scan List and order channels so channel 15 will be scanned before channels 16 through 23.

SENS:REF THER, 5000, (@115)	*5K thermistor temperature for channel 15*
SENS:FUNC:TEMP TC,J,(@116:123)	*Type J thermocouple temperature for channels 16 through 23*
SENS:REF:CHAN (@115),(@116:123)	*reference channel scanned before channels 16 - 23*

Supplying a Fixed Reference Temperature

The [SENse:]REFerence:TEMPerature <*degrees_c*> command immediately stores the temperature of a controlled temperature reference junction panel in the Reference Temperature Register. The value is applied to all subsequent thermocouple channel measurements until another reference temperature value is specified or measured. There is no need to use SENS:REF:CHANNELS.

To specify the temperature of a controlled temperature reference panel:

SENS:REF:TEMP 50	*reference temp = 50 °C*
Now begin scan to measure thermocouples	

Linking Strain Measurements

Strain measurements usually employ a Strain Completion and Excitation SCP (HP E1506,E1507,E1511) or HP E1529 Remote Strain Conditioning Unit. To link channels to strain EU conversions send the [SENSe:]FUNCtion:STRain:<*bridge_type*> [<*range*>,](@<*ch_list*>)

- <*bridge_type*> is not a parameter but is part of the command syntax. The following table relates the command syntax to bridge type. See the HP E1506 and HP E1507, and HP E1511 SCPs' user's manual for bridge schematics and field wiring information.

Command	Bridge Type
:FBENding	Full Bending Bridge
:FBPoisson	Full Bending Poisson Bridge
:FPOisson	Full Poisson Bridge
:HBENding	Half Bending Bridge
:HPOisson	Half Poisson Bridge
[:QUARter]	Quarter Bridge (default)
:Q120 *	Quarter using HP E1529A's internal 120Ω resistor
:Q350 *	Quarter using HP E1529A's internal 350Ω resistor
:User **	Quarter using HP E1529A's user supplied resistor value

* These choices are only available with the HP E1529A

** This choice is only available with HP E1529A channels that have had a user supplied resistor installed.

- The *ch_list* parameter specifies which sense SCP channel(s) to link to the strain EU conversion, not the strain bridge completion SCP channels. *ch_list* does not specify channels on the HP E1506, and 07 Strain Bridge Completion SCPs. *ch_list* can specify any of the lower four channels of an HP E1511 SCP since these channels are the sense channels used to measure the SCPs four bridge completion channels.

- HP E1529A channels provide both strain bridge completion and bridge output sense so *ch_list* links strain EU conversion directly to those channels.

Note When the SENS:FUNC:STR:<bridge_type> command is used with HP E1529A channels, the bridge configuration switches for those channels are set to actually configure the bridge type specified. There is no need to send the configuration only SENSe:STRain:BRIDge:TYPE command for channels that use the SENSe:FUNCtion:STRain:<bridge_type> command.

- The optional *range* parameter can be used to choose a fixed A/D range. When not specified (defaulted), the module uses auto-range.

The following command sequence is for conventional strain completion SCPs. For HP E1529A based command sequences, see “Programming the HP E1422A & HP E1529A for Remote Strain Measurement” on page 55

To link channels 23 through 30 to the quarter bridge strain EU conversion:

SENS:FUNC:STR:QUAR (@123:130) *uses autorange*

Other commands used to set up strain measurements are:
[SENSe:]STRain:POISson
[SENSe:]STRain:EXCitation
[SENSe:]STRain:GFACtor
[SENSe:]STRain:UNSTrained

NOTE Because of the number of possible strain gage configurations, the driver must generate any Strain EU conversion tables and download them to the instrument when INITiate is executed. This can cause the time to complete the INIT command to exceed 1 minute.

See the Command Reference Chapter 6 and the HP E1506/E1507, and HP E1511 User's Manuals for more information on strain measurements.

Custom EU Conversions

"Creating and Loading Custom EU Conversion Tables" on page 142.

Linking Output Channels to Functions

Analog outputs are implemented either by an HP E1531 Voltage Output SCP or an HP E1532 Current Output SCP. Channels where these SCPs are installed are automatically considered outputs. No SOURce:FUNCtion command is required since the HP E1531 can only output voltage, while the HP E1532 can only output current. The only way to control the output amplitude of these SCPs is through the HP E1422's Algorithm Language.

Setting up Digital Input and Output Channels

Setting up Digital Inputs

Digital inputs can be configured for polarity and depending on the SCP model, a selection of input functions as well. The following discussion will explain which functions are available with a particular Digital I/O SCP model. Setting a digital channel's input function is what defines it as an input channel.

Setting Input Polarity

To specify the input polarity (logical sense) for digital channels use the command INPut:POLarity <*mode*>,(@<*ch_list*>). This capability is available on all digital SCP models. This setting is valid even while the specified channel in not an input channel. If and when the channel is configured for input (an input FUNCtion command), the setting will be in effect.

- The <*mode*> paramter can be either NORMal or INVerted. When set to NORM, an input channel with 3v applied will return a logical 1. When set to INV, a channel with 3v applied will return a logic 0.
- The <*ch_list*> parameter specifies the channels to configure. The HP E1533 has 2 channels of 8 bits each. All 8 bits in a channel take on

the configuration specified for the channel. The HP E1534 has 8 I/O bits that are individually configured as channels.

To configure the lower 8 bit channel of an HP E1533 for inverted polarity:

INP:POLARITY INV,(@108) *SCP in SCP position 1*

To configure the lower 4 bits of an HP E1534 for inverted polarity:

INP:POL INV,(@132:135) *SCP in SCP position 4*

Setting Input Function

The HP E1533 Digital I/O SCP and the HP E1534 Frequency/Totalizer SCP can both input static digital states. The HP E1534 Frequency/Totalizer SCP can also input Frequency measurements and Totalize the occurrence of positive or negative edges.

Static State (CONDition) Function

To configure digital channels to input static states, use the [SENSe:]FUNCtion:CONDition (@<*ch_list*>) command. Examples:

To set the lower 8 bit channel of an HP E1533 in SCP position 4 to input
SENS:FUNC:COND (@132)
To set the upper 4 channels (bits) of an HP E1534 in SCP pos 2 to input states
SENS:FUNC:COND (@120:123)

Frequency Function

The frequency function uses two commands. For more on this HP E1534 and HP E1538 capability see the appropriate SCP's User's Manual.

To set the frequency counting gate time execute:
[SENSe:]FREQuency:APERature <*gate_time*>,(@<*ch_list*>)

Sets the digital channel function to frequency
[SENSe:]FUNCtion:FREQuency (@<*ch_list*>)

Totalizer Function

The totalizer function uses two commands also. One sets the channel function, and the other sets the condition that will reset the totalizer count to zero. For more on this HP E1534 and HP E1538 capability see the appropriate SCP's User's Manual.

To set the HP E1534's totalize reset mode

[SENSe:]TOTalize:RESet:MODE INIT | TRIG,(@<*ch_list*>)

To configure HP E1534 channels to the totalizer function
[SENSe:]FUNCtion:TOTalize (@<*ch_list*>)

Setting up Digital Outputs

Digital outputs can be configured for polarity, output drive type, and depending on the SCP model, a selection of output functions as well. The following discussion will explain which functions are available with a particular Digital I/O SCP model. Setting a digital channel's output function is what defines it as an output channel.

Setting Output Polarity

To specify the output polarity (logical sense) for digital channels use the command OUTPut:POLarity <*mode*>,(@<*ch_list*>). This capability is available on all digital SCP models. This setting is valid even while the specified channel in not an output channel. If and when the channel is configured for output (an output FUNCtion command), the setting will be in effect.

- The <*mode*> paramter can be either NORMal or INVerted. When set to NORM, an output channel set to logic 0 will output a TTL compatible low. When set to INV, an output channel set to logic 0 will output a TTL compatible high.

- The <*ch_list*> parameter specifies the channels to configure. The HP E1533 has 2 channels of 8 bits each. All 8 bits in a channel take on the configuration specified for the channel. The HP E1534 and HP E1538 have 8 I/O bits that are individually configured as channels.

To configure the higher 8 bit channel of an HP E1533 for inverted polarity:

OUTP:POLARITY INV,(@109) *SCP in SCP position 1*

To configure the upper 4 bits of an HP E1534 for inverted polarity:

OUTP:POL INV,(@132:135) *SCP in SCP position 4*

Setting Output Drive Type

The HP E1533 and HP E1534 use output drivers that can be configured as either active or passive pull-up. To configure this, use the command OUTPut:TYPE <*mode*>,(@<*ch_list*>). This setting is valid even while the specified channel in not an output channel. If and when the channel is configured for output (an output FUNCtion command), the setting will be in effect.

- The <*mode*> parameter can be either ACTive or PASSive. When set to ACT (the default), the output provides active pull-up. When set to PASS, the output is pulled up by a resistor.

- The <*ch_list*> parameter specifies the channels to configure. The HP E1533 has 2 channels of 8 bits each. All 8 bits in a channel take on the configuration specified for the channel. The HP E1534 has 8 I/O bits that are individually configured as channels.

To configure the higher 8 bit channel of an HP E1533 for passive pull-up:

OUTP:TYPE PASS,(@109) *SCP in SCP position 1*

To configure the upper 4 bits of an HP E1534 for active pull-up:

OUTP:TYPE ACT,(@132:135) *SCP in SCP position 4*

Setting Output Functions

Both the HP E1533 Digital I/O SCP, and HP E1534 and HP E1538 Frequency/Totalizer SCPs can output static digital states. The Frequency/Totalizer SCPs can also output single pulses per trigger, continuous pluses that are width modulated (PWM, and continuous pulses that are frequency modulated (FM).

Static State (CONDition) Function

To configure digital channels to output static states, use the SOURce:FUNCtion:CONDition (@<*ch_list*>) command. Examples:

To set the upper 8 bit channel of an HP E1533 in SCP position 4 to output

```
SOUR:FUNC:COND (@133)
```

To set the lower 4 channels (bits) of an HP E1534 in SCP pos 2 to output states

```
SOUR:FUNC:COND (@116:119)
```

To configure digital channels to output static states:

Variable Width Pulse Per Trigger

This function sets up one or more HP E1534 or HP E1538 channels to output a single pulse per trigger (per algorithm execution). The width of the pulse from these channels is controlled by Algorithm Language statements. Use the command SOURce:FUNCtion[:SHAPe]:PULSe (@<*ch_list*>). Example command sequence:

To set HP E1534/38 channel 2 at SCP position 4 to output a pulse per trigger

```
SOUR:FUNC:PULSE (@134)
```

Example algorithm statement to control pulse width to 1 msec

```
O134 = 0.001;
```

Variable Width Pulses at Fixed Frequency (PWM)

This function sets up one or more HP E1534/38 channels to output a train of pulses. A companion command sets the period for the complete pulse (↑ edge to ↑ edge). This of course fixes the frequency of the pulse train. The width of the pulses from these channels is controlled by Algorithm Language statements.

Use the command SOURce:FUNCtion[:SHAPe]:PULSe (@<*ch_list*>). Example command sequence:

Enable pulse width modulation for HP E1534's first channel at SCP position 4

```
SOUR:PULM:STATE ON,(@132)
```

To set pulse period to 0.5 msec (which sets the signal frequency 2 KHz)

```
SOUR:PULSE:PERIOD 0.5e-3,(@132)
```

To set function of HP E1534's first channel in SCP position 4 to PULSE

```
SOUR:FUNCTION:PULSE (@132)
```

Example algorithm statement to control pulse width to .1 msec (20% duty-cycle)

```
O132 = 0.1e-3;
```

Fixed Width Pulses at Variable Frequency (FM)

This function sets up one or more HP E1534/38 channels to output a train of pulses. A companion command sets the width (↑ edge to ↓ edge) of the pulses. The frequency of the pulse train from these channels is controlled by Algorithm Language statements.

Use the command SOURce:FUNCtion[:SHAPe]:PULSe (@<*ch_list*>). Example command sequence:

To enable frequency modulation for HP E1534's second channel at SCP position 4

```
SOUR:FM:STATE  ON,(@133)
```
To set pulse width to 0.3333 msec
```
SOUR:PULSE:WIDTH  0.3333e-3,(@133)
```
To set function of HP E1534's second channel in SCP position 4 to PULSE
```
SOUR:FUNCTION:PULSE  (@133)
```
Example algorithm statement to control frequency to 1000 Hz
```
O133 = 1000;
```

Variable Frequency Square-Wave Output (FM)

To set function of HP E1534/38's third channel in SCP position 4 to output a variable frequency square-wave.
```
SOUR:FUNCTION:SQUare  (@134)
```
Example Algorithm Language statement to set output to 20KHz
```
O134 = 20e3;
```

For complete HP E1534/38 capabilities, see the SCP's User's Manual.

Performing Channel Calibration (Important!)

Calibrationg the HP E1422A

The *CAL? (also performed using CAL:SETup then CAL:SETup?) is a very important step. *CAL? generates calibration correction constants for all analog input and output channels on-board the HP E1422A. *CAL? must be performed in order for the HP E1422 to deliver its specified accuracy.

Operation and Restrictions

*CAL? generates calibration correction constants for each analog input channel for offset and gain at all 5 A/D range settings. For programmable input SCPs, these calibration constants are only valid for the current configuration (gain, and filter cut-off frequency). This means that *CAL? calibration is no longer valid if you change channel gain or filter settings (INP:FILT or INP:GAIN), but is still valid for changes of channel function or range (using SENS:FUNC ...). The calibration becomes invalid if you move these SCPs to different SCP locations.

For analog output channels (both measurement excitation SCPs as well as control output SCPs) *CAL? also generates calibration correction constants. These calibration constants are valid only for the specific SCPs in the positions they are currently in. The calibration becomes invalid if you move these SCPs to different SCP locations.

How to Use *CAL?

When you turn power on to the HP E1422 after you have first installed your SCPs (or after you have moved SCPs), the module will use approximate values for calibration constants. This means that input and output channels will function although the values will not be as accurate as the HP E1422's specified capability. At this point, make sure the module is firmly anchored to the mainframe (front panel screws are tight), and let it warm up for a full hour. After it has warmed up, execute *CAL?.

What *CAL? Does

The *CAL? command causes the module to calibrate A/D offset and gain, and all channel offsets. This may take many minutes to complete. The actual time it will take your HP E1422 to complete *CAL? depends on the mix of SCPs installed. *CAL? performs literally hundreds of measurements of the internal calibration sources for each channel and must allow 17 time constants of settling wait each time a filtered channel's calibration source changes value. The *CAL? procedure is internally very sophisticated and results in an extremely well calibrated module.

When *CAL? finishes, it returns a +0 value to indicate success. The generated calibration constants are now in volatile memory as they always are when ready to use. If the configuration just calibrated is to be fairly long-term, you should now execute the CAL:STORE ADC command to store these constants in non-volatile memory. That way the module can restore calibration constants for this configuration in case of a power failure. After power returns, and after the module warms up, these constants will be relatively accurate.

Re-Execute *CAL? When:

- When you change the channel gain and/or filter cut-off frequency on programmable SCPs (using INPut:GAIN, or INPut:FILTer ...)
- When you re-configure SCPs to different locations. This is true even if you replace an SCP with an identical model SCP because the calibration constants are specific to each SCP channel's individual performance.
- When the ambient temperature within the mainframe changes significantly. Temperature changes affect accuracy much more than long-term component drift. See temperature coefficients in Appendix A page 369 "Specifications".

NOTE To save time when performing channel calibration on multiple HP E1422s in the same mainframe, use the CAL:SETup and CAL:SETup? commands (See "CALibration:SETup" on page 232. for details).

Calibrating Remote Signal Conditioning Units

RSCUs have a local calibration source that the HP E1422A can measure directly. This source voltage along with a local short can be fed to each channel on the RSCU. The HP E1422A reads the output value of each remote channel when connected to the short, and then the calibration voltage source. Using this method, the HP E1422A can determine the offset and gain values for each remote channel. Further, these values can be stored in non-volatile memory in the RSCU. The commands used to perform the remote calibration are:CALibration:REMote (@<*ch_list*>) where *ch_list* need only contain the first channel on each RSCU to calibrate all channels on that RSCU. The command to store the calibration constants into non-volatile memory is: CALibration:REMote:STORe (@<*ch_list*>) where *ch_list* need only contain the first channel on each RSCU to store the calibration constants into non-volatile flash memory.

Defining an Analog Input Scan List (ROUT:SEQ:DEF)

In this programming step you will define the contents of the analog input Scan List using the ROUTe:SEQuence:DEFine command. This allows you to make measurements that will be stored to the Current Value Table (CVT) and/or the FIFO buffer without programming or executing any algorithms. While you can use the HP E1422 exlusively in this way, you can also combine both modes of operation (scanned analog input and algorithmic acquisition-and-control). In fact there is only a single analog input scan list and it is defined as the sum of channels specified by ROUT:SEQ:DEF and referenced in any algorithms downloaded with the ALG:DEF SCPI command or the hpe1422_downloadAlg(...) plug&play function. Duplicate channel references are discarded. No matter how many times a channel is referenced, it is only measured once per trigger and the same value is seen in storage and by algorithms.

ROUTe:SEQuence:DEFine accepts both on-board channels from conventional SCPs as well as remote channels from Remote Signal Conditioning Units (RSCUs). For details about syntax see “Channel List (Standard Form)” starting on page 199, and “ROUTe:SEQuence:DEFine” on page 287.

Note Certain analog input SCPs display higher than normal offset and noise figures if their channels are scanned just before channels on a Remote Signal Conditioning Unit. To avoid any such interraction, you should order your scan list so all remote channels (5-digit channel numbers) appear before any on-board channels (3-digit channel numbers)

Example Scan List

To set-up a scan list to take measurements on all on-board channels of a conventional SCP in position 0 and all remote channels of 4 HP E1529As connected to 2 HP E1539A SCPs in SCP positions 1 and 2:

```
ROUT:SEQ:DEF  (@100:107,10800:11731)
```

Controlling Scan List Data Destination

Readings taken on channels specified by ROUT:SEQ:DEF by default go to both the FIFO buffer and the CVT. By using another form of the <ch_list> parameter this data destination can be controlled to be the CVT, theFIFO or even neither (no reading stored). For more on controlling data destination See “ROUTe:SEQuence:DEFine” on page 287.

Example Scan List with controlled data destination

To set-up a scan list as above but send the remote channel readings only to the FIFO buffer:

```
ROUT:SEQ:DEF  (@100:107,2(10800:11731))
```

Defining C Language Algorithms

This section is an overview of how to write and download C algorithms into the E1422's memory. The assumption is that you have some programming experience in C, but since the E1422's version of C is limited, just about any experience with a programming language will suffice. "Creating and Running Algorithms" on page 153 for a complete descussion of the E1422's C language algorithm functionality.

Arithmetic Operators: add `+`, subtract `-`, multiply `*`, divide `/`
Assignment Operator: =
Comparison Functions: less than `<`, less than or equal `<=`, greater than `>`, greater than or equal `>=`, equal to `==`, not equal to `!=`
Boolean Functions: and `&&`, or `||`, not `!`
Variables: scalars of type `static float`, and single dimensioned arrays of type `static float` limited to 1024 elements.
Constants:
32-bit decimal integer; `Dddd...` where `D` and `d` are decimal digits but `D` is not zero. No decimal point or exponent specified.
32-bit octal integer; `0oo...` where `0` is a leading zero and `o` is an octal digit. No decimal point or exponent specified.
32-bit hexadecimal integer; `0Xhhh...` or `0xhhh...` where `h` is a hex digit.
32-bit floating point; `ddd.`, `ddd.ddd`, `ddde±dd`, `dddE±d`, `ddd.ddde±dd`, or `ddd.dddE±dd` where d is a decimal digit.
Flow Control: conditional construct `if(){ } else { }`
Intrinsic Functions:
Return the absolute value; `abs(`<expr>`)`
Return minimum; `min(`<expr1>`,`<expr2>`)`
Return maximum; `max(`<expr1>`,`<expr2>`)`
User defined function; <user_name>`(`<expr>`)`
Write value to CVT element; `writecvt(`<expr>`,`<expr>`)`
Write value to FIFO buffer; `writefifo(`<expr>`)`
Write value to both CVT and FIFO; `writeboth(`<expr>`,`<expr>`)`

Note for VXIplug&play users

While the following discussion of algorithm definition is useful for plug&play users as regards the coding of the algorithm or global variable definition, the method of generating the algorithm code and actually down-loading it to the HP E1422 becomes much easier because of plug&play e1422.exe Soft Front Panel program, and hpe1422_downloadAlg(...) plug&play driver function.

Using the SFP "Algorithm Panel", you can create and test your algorithm, and then store it to a file. The hpe1422_downloadAlg(...) plug&play driver function was created specifically to download algorithms from files into your HP E1422A as part of your application program.

Global variable definition

Global variables are necessary when you need to communicate information from one algorithm to another. Globals are initialized to 0 unless specifically assigned a value at define time. The initial value is only valid at the time of definition. That is, globals remain around and may be altered by other SCPI commands or algorithms. Globals are removed only by power-ON or *RST. The following string output is valid for strings of 256 characters or less.

```
ALG:DEF ’globals’,’static float output_max = 1, coefficients[ 10 ];’
```

If the global definition exceeds 256 characters, you need to download an indefinite block header, the definitions, and terminated by a LF/EOI sequence:
ALG:DEF ’globals’,#0static float output_max = 1, ... LF/EOI

The LF/EOI sequence is part of the I/O and Instrument Manager in HP VEE. You must edit the E1422 I/O device for direct I/O and purposely select EOI to be sent with the EOL terminator.

Algorithm definition

Algorithms are similar in nature to global definitions. Both scalars and arrays can be defined for local use by the algorithm. If less than 256 characters, you need only place the algorithm code within string quotes:

```
ALG:DEF ’alg1’,’static float a = 1; if ( I100 > a ) writecvt( I100,10);’
```

If the algorithm exceeds 256 characters, you need to download an indefinite block header, the algorithm code, and terminated by a LF/EOI sequence:

```
ALG:DEF ’alg2’,#0static float a = 1; ... ;LF/EOI
```

Algorithms remain around and cannot be altered once defined unless you specify a fixed size for the algorithm(see Chapter 4). Algorithms are removed from memory only by issueing a *RST or power-ON condition.

Pre-setting Algorithm Variables

As you may have noticed in the examples above, you can initialize a variable to a particular value. However, that value is a one-time initialization. Later program execution may alter the variable, and re-issueing an INIT command to re-start program execution will NOT re-initialize that variable. Instead, you can alter any scalar or array using SCPI commands prior to issueing the INIT command, or you can rely upon the intrinsic variable First_loop to conditionally preset variables after receiving the INIT command. First_loop is a variable that is preset to non-zero due to the execution of the INIT command. With the occurrence of the first scan trigger and when algorithms execute for the first time, First_loop’s value will be non-zero. Subsequent triggers will find this variable cleared. Here’s an example of how First_loop can be used:

```
ALG:DEF ’alg1’,#0static float a,b,c, start, some_array[ 4 ]; if ( First_loop ){ a
= 1; b = 2; c = 3; } * * LF/EOI
```

To pre-set variables under program control before issueing the INIT command, the ALG:SCALAR and ALG:ARRAY commands can be used. Assume the example algorithm above has already been defined. To preset the scalar start and the array some_array, you can use the following commands:

```
ALG:SCAL 'alg1','start',1.2345
ALG:ARR 'alg1','some_array',#232..........LF/EOI
ALG:UPD
```

The ALG:SCAL command designates the name of the algorithm of where to find the local variable start and assigns that variable the value of 1.2345. Likewise, the ALG:ARRAY command designates the name of the algorithm, the name of the local array, and a definite length block for assigning the four real number values. As you can see, the scalar assignment uses ASCII and the array assignment uses binary. The later makes for a much faster transfer especially for large arrays. The format used is IEEE-754 8-byte binary real numbers. The header is #232 which states "the next 2 bytes are to be used to specify how many bytes are comming". In this case, 32 bytes represent the four 8-byte elements of the array. A 100 element array would have a header of #3800. If you wanted to pre-initialize a global scalar or array, the word 'globals' must be used instead of the algorithm name. The name simply specifies the memory space of where to find those elements.

As stated earlier in the chapter, all updates (changes) are held in a holding buffer until the computer issues the update command. The ALG:UPD is that command. Executing ALG:UPD before INIT does not make much difference since there is no concern as to how long it takes or how it is implemented. After INIT forces the buffered changes to all take place during the next Update Phase in the trigger cycle after reception of the ALG:UPD command..

For VXIplug&play users use the functions hpe1422_algArray, hpe1422_algScal to send new values to algorithm variables, and hpe1422_cmd to send the ALG:UPD... SCPI command. See your HP E1422 plug&play driver Help file

Defining Data Storage

Specifying the Data Format

The format of the values stored in the FIFO buffer and CVT never changes. They are always stored as IEEE 32-bit Floating point numbers. The FORMat <*format*>[,<*length*>] command merely specifies whether and how the values will be converted as they are transferred from the CVT and FIFO to the host computer.

- The <*format*>[,<*length*>] parameters can specify:

PACKED	Same as REAL,64 except for the values of IEEE -INF, IEEE +INF, and Not-a-Number (NaN). See FORMat command in Chapter 5 for details.
REAL,32	means real 32-bit (no conversion, fastest)
REAL	same as above
REAL,64	means real 64-bit (values converted)
ASCii,7	means 7-bit ASCII (values converted)
ASCii	same as above (the *RST condition)

To specify that values are to remain in IEEE 32-bit Floating Point format for fastest transfer rate:

```
FORMAT REAL,32
```

To specify that values are to be converted to 7-bit ASCII and returned as a 15 character per value comma separated list:

FORMAT ASC,7	*The *RST, *TST? and power-on default format*
or	
FORM ASC	*same operation as above*

Selecting the FIFO Mode

The HP E1422's FIFO can operate in two modes. One mode is for reading FIFO values while the HP E1422 is scanning and/or running algorithms, the other mode is for reading FIFO values after operation have been halted (ABORT sent).

- BLOCking; The BLOCking mode is the default and is used to read the FIFO while algorithms are executing. Your application program must read FIFO values often enough to keep it from overflowing (See "Continuously Reading the FIFO (FIFO mode BLOCK)" on page 127.). The FIFO stops accepting values when it becomes full (65,024 values). Values sent after the FIFO is full are discarded. The first value to exceed 65,024 sets the STAT:QUES:COND? bit 10 (FIFO Overflowed), and an error message is put in Error Queue (read with SYS:ERR? command).

- Overwrite; When the HP E1422 is running and the FIFO fills, the oldest values in the FIFO are overwritten by the newest values. Only the latest 65,024 values are available. In OVERwrite mode the module must be halted (ABORT sent) before reading the FIFO (See "Reading the Latest FIFO Values (FIFO mode OVER)" on page 128.). This mode is very useful when you want to view an algorithm's response to a disturbance.

To set the FIFO mode (blocking is the *RST/Power-on condition):

[SENSe:]DATA:FIFO:MODE OVERWRITE	*select overwrite mode*
[SENSe:]DATA:FIFO:MODE BLOCK	*select blocking mode*

Setting up the Trigger System

Arm and Trigger Sources

Figure 4-6 shows the trigger and arm model for the HP E1422. Note that when the Trigger Source selected is TIMer(the default), the remaining sources become Arm Sources. Using ARM:SOUR allows you to specify an event that must occur in order to start the Trigger Timer. The default Arm source is IMMediate (always armed).

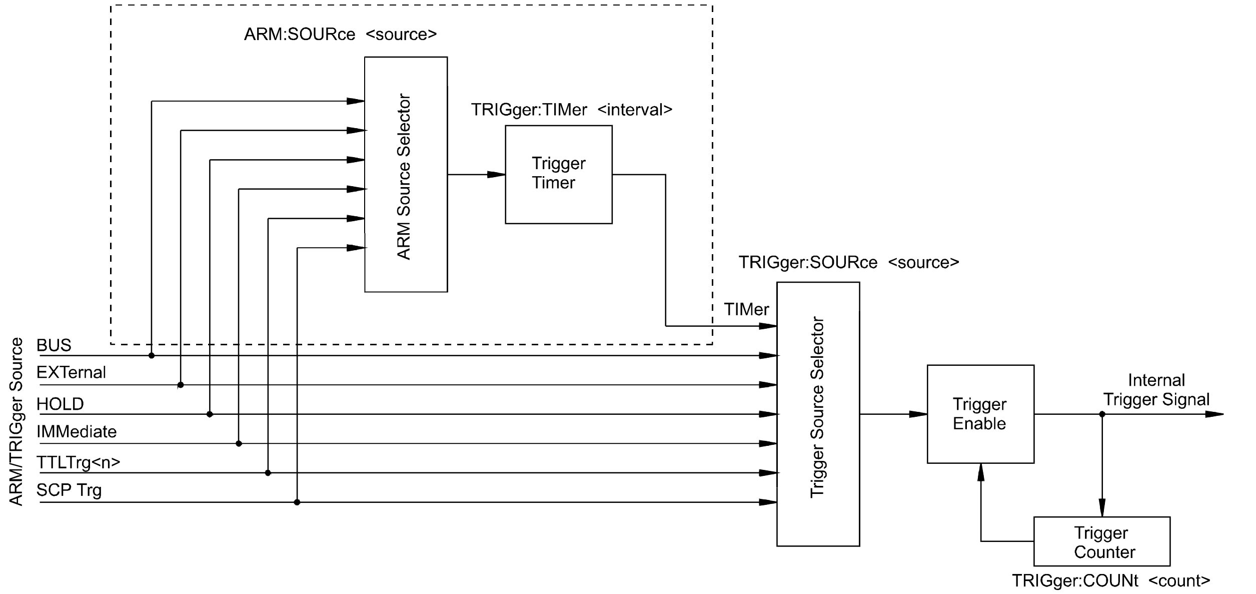


Figure 4-6. Logi

Selecting the Trigger Source

In order to start an instrument operation cycle, a trigger event must occur. The source of this event is selected with the TRIGger:SOURce <*source*> command. The following table explains the possible choices for <*source*>.

Parameter Value	Source of Trigger (after INITiate:… command)
BUS	TRIGger[:IMMediate], *TRG, GET (for HP-IB)
EXTernal	"TRG" signal input on terminal module
HOLD	TRIGger[:IMMediate]
IMMediate	The trigger signal is always true (scan starts when an INITiate:… command is received).
SCP	SCP Trigger Bus (future HP or SCP Breadboard)
TIMer	The internal trigger interval timer (must set Arm source)
TTLTrg<n>	The VXIbus TTLTRG lines (n=0 through 7)

NOTES

1. When TRIGger:SOURce is not TIMer, ARM:SOURce must be set to IMMediate (the *RST condition). If not, the INIT command will generate an error -221,"Settings conflict".
2. When TRIGger:SOURce is TIMer, the trigger timer interval (TRIG:TIM <*interval*>) must allow enough time to scan all channels, execute all algorithms and update all outputs or a +3012, "Trigger Too Fast" error will be generated during the trigger cycle. See the

TRIG:TIM command on page 349 for details.

To set the trigger source to the internal Trigger Timer (the default):

TRIG:SOUR TIMER *now select ARM:SOUR*

To set the trigger source to the External Trigger input connection:

TRIG:SOUR EXT *an external trigger signal*

To set the trigger source to a VXIbus TTLTRG line:

TRIG:SOUR TTLTRG1 *the TTLTRG1 trigger line*

Selecting Trigger Timer Arm Source

Figure 4-6 shows that when the TRIG:SOUR is TIMer, the other trigger sources become Arm sources that control when the timer will start. The command to select the arm source is ARM:SOURce <*source*>.

- The <*source*> parameter choices are explained in the following table

Parameter Value	Source of Arm (after INITiate:... command)
BUS	ARM[:IMMediate]
EXTernal	"TRG" signal input on terminal module
HOLD	ARM[:IMMediate]
IMMediate	The arm signal is always true (scan starts when an INITiate:... command is received).
SCP	SCP Trigger Bus (future HP or SCP Breadboard)
TTLTrg<n>	The VXIbus TTLTRG lines (n=0 through 7)

NOTE When TRIGger:SOURce is not TIMer, ARM:SOURce must be set to IMMediate (the *RST condition). If not, the INIT command will generate an error -221,"Settings conflict".

To set the external trigger signal as the arm source:

ARM:SOUR EXT *trigger input on connector module*

Programming the Trigger Timer

When the HP E1422 is triggered, it begins its instrument operation cycle. The time it takes to complete a cycle is the minimum interval setting for the Trigger Timer. If programmed to a shorter time, the module will generate a "Trigger too fast" error. So, how can you determine this minimum time? After you have defined all of your algorithms, you send the ALG:TIME? command with its <*alg_name*> parameter set to ’MAIN’. This causes the

HP E1422's driver to analyze the time required for all four phases of the operation cycle; Input, Update, Calculate, and Output. The value returned from ALG:TIME? 'MAIN' is the minimum allowable Trigger Timer interval required to avoid the "Trigger too fast" error. With this information you now execute the command TRIGger:TIMer <*interval*> and set <*interval*> to the desired time that is equal to or greater than the minimum.

Setting the Trigger Counter

The Trigger Counter controls how many trigger events will be allowed to start an input-calculate-output cycle. When the number of trigger events set with the TRIGger:COUNt command is reached, the module returns to the Trigger Idle State (needs to be INITiated again). The default Trigger Count is 1. Note that this default was chosen to make testing data aquisition scan list easier (only one scan list of data in the FIFO). For algorithm operation, you will probably want to change the count to INFinite (can be triggered an unlimited number of times). This setting will be used most often for un-interrupted execution of control algorithms.

To set the trigger count to 50 (perhaps to help debug an algorithm):

TRIG:COUNT 50	*execute algorithms 50 times then return to Trig Idle State.*

Sending Trigger Signals to Other Instruments

The HP E1422 can output trigger signals on any of the VXIbus TTLTRG lines. Use the OUTPut:TTLTrg<n>[:STATe] ON | OFF command to select one of the TTLTRG lines and then choose the source that will drive the TTLTRG line with the command OUTPut:TTLTrg:SOURce command. For details see OUTP:TTLTRG commands starting on page 282

To output a signal on the TTLTRG1 line each time the Trigger Timer cycles execute the commands:

TRIG:SOUR TIMER	*select trig timer as trig source*
OUTP:TTLTRG1 ON	*select and enable TTLTRG1 line*
OUTP:TTLTRG:SOUR TRIG	*each trigger output on TTLTRG1*

INITiating the Module/Starting Scanning and Algorithms

When the INITiate[:IMMediate] command is sent, the HP E1422 builds the input Scan List from the input channels you referenced when you defined the algorithm with the ALG:DEF command above and from the channels you referenced with the ROUTe:SEQuence:DEFine command. The module also enters the Waiting For Trigger State. In this state, all that is required to start a scan and/or run an algorithm is a trigger event for each pass through the input-calculate-output instrument operation cycle. To initiate the module, send the command:

INIT	*module to Waiting for Trigger State*

When an INIT command is executed, the driver checks several interrelated settings programmed in the previous steps. If there are conflicts in these settings an error message is placed in the Error Queue (read with the

SYST:ERR? command). Some examples:

- If TRIG:SOUR is not TIMer then ARM:SOUR must be IMMediate.
- The time it would take to execute all algorithms is longer than the TRIG:TIMER interval currently set.

Starting Scanning and/or Algorithms

Once the module is INITiated it can accept triggers from any source specified in TRIG:SOUR.

TRIG:SOUR TIMER	*(*RST default)*
ARM:SOUR IMM	*(*RST default)*
INIT	*INIT starts Timer triggers*

or

TRIG:SOUR TIMER	
ARM:SOUR HOLD	
INIT	*INIT readies module*
ARM	*ARM starts Timer triggers.*

... and the algorithms start to execute.

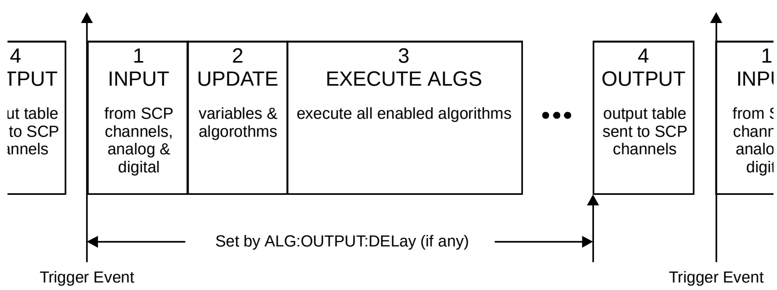


Figure 4-7. Sequence of Loop Operations

The Operating Sequence

The HP E1422 has four major operating phases. Figure 4-7 shows these phases. A trigger event starts the sequence:

1. (INPUT); the state of all digital inputs are captured and each analog input channel that is in the scan list and/or referenced by an algorithm variable is scanned. Reading values from channels placed in the Scan List with ROUT:SEQ:DEF are sent to the CVT and/or FIFO.

2. (UPDATE); The update phase is a window of time made large enough to process all variables and algorithm changes made after INIT. Its width is specified by ALG:UPDATE:WINDOW. This window is the only time variables and algorithms can be changed. Variable and

algorithm changes can actually be accepted during other phases, but the changes don't take place until an ALG:UPDATE command is received and the update phase begins. If no ALG:UPDATE command is pending, the update phase is simply used to accept variable and algorithm changes from the application program (using ALG:SCAL, ALG:ARR, ALG:DEF). Data acquired by external specialized measurement instruments can be sent to your algorithms at this time.

Note Changing algorithm variables requires HP E1422 hardware resources that can only be provided during the INPUT and UPDATE phases of the operating cycle. The HP E1422A does not update variables during the time between the CALCULATE and OUTPUT phases. Therefore, applications that are intensive in the update area should consciously extend the INPUT and UPDATE periods through use of the ALG:UPD:WINDOW and SAMP:TIME commands or, by reducing the time between the CALCULATE and OUTPUT phases through shorter algorithm loop time. See TRIG:TIMer.

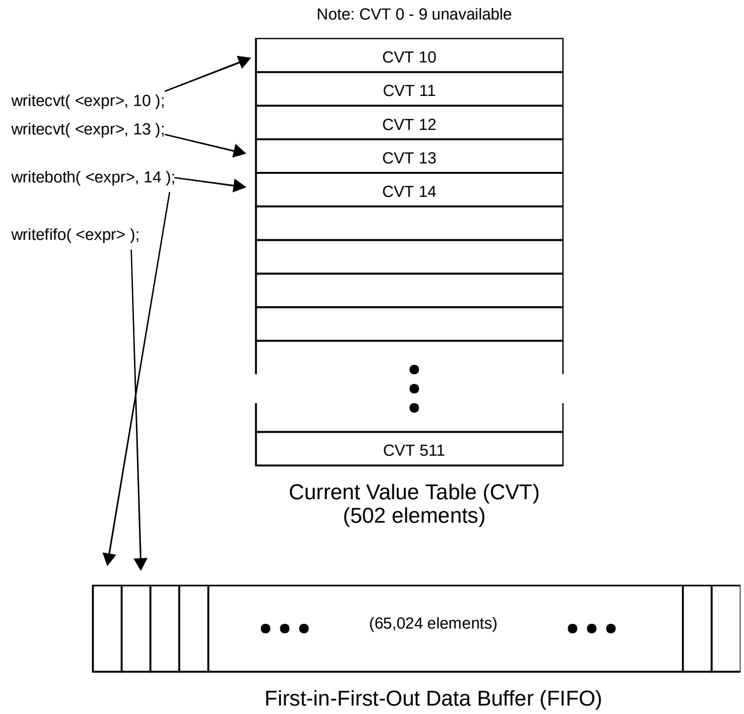
3. (CALCULATE); all INPUT and UPDATE values have been made available to the algorithm variables and each enabled algorithm is executed. The results to be output from algorithms are stored in the Output Channel Buffer.

4. (OUTPUT); each Output Channel Buffer value stored during (CALCULATE) is sent to its assigned SCP channel. The start of the OUTPUT phase relative to the Scan Trigger can be set with the SCPI command ALG:OUTP:DELay.

Reading Running Algorithm Values

The most efficient means of acquiring algorithm derived data from the E1422 is to have its algorithms store real-number results in the FIFO or CVT. The algorithms use the writefifo(), writecvt(), and writeboth()

intrinsic functions to perform this operation as seen in Figure 3-9.



Reading CVT Data

Note that the first 10 elements of the CVT are unavailable. These are used by the driver for internal data retrieval. However, all algorithms have access to the remaining 502 elements. Data is retrieved from the CVT with the SCPI command DATA:CVT? (@10,12,14:67)

For VXIplug&play users use the function hpe1422_readCVT_Q for reading contiguous elements or hpe1422_cmdReal64Arr_Q(ViSession vi, 'DATA:CVT? (@<element_list>)', ViInt32 size, ViReal64 _VI_FAR result[ ], ViPInt32 count) for non-contiguous elements (as in the example above). See your HP E1422 plug&play driver Help file.

The format of data comming from the CVT is determined by the FORMat command.

Important! There is a fixed relationship between channel number and CVT element for values from channels placed in the Scan List with ROUT:SEQ:DEF. When you are mixing Scan List data acquisition with algorithm data storage, be careful not to overwrite Scan List generated values with algorithm generated values. See “ROUTe:SEQuence:DEFine” on page 287. for controlling CVT entries from the analog scan list.

Note After *RST/Power-on, each element in the CVT contains the IEEE-754 value "Not-a Number" (NaN). Channel values which are a positive overvoltage return IEEE +INF and negative overvoltage return IEEE -INF. Refer to the FORMat command in on page 257 for the NaN, +INF, and -INF values for each data format.

Reading FIFO Data

The FIFO can store up to 65024 real numbers. Each writefifo() or writeboth() cause that expression to be placed into the FIFO. With a FIFO this large, you can store many seconds worth of data, dependent upon the volume of writes and the trigger cycle time. The FIFO's most valuable service is to keep your computer from having to spend too much time acquiring data from the E1422. Data is retrieved from the FIFO with the SCPI command DATA:FIFO:PART?<count>

<count> can be a number larger than the FIFO(up to 2.1 billion) if you want to read data continuously. And, you can query how much data is in the FIFO with the DATA:FIFO:COUNT? command.

For VXIplug&play users see the functions hpe1422_readFifo32_Q andhpe1422_readFifo_Q in your HP E1422 plug&play driver Help file.

Which FIFO Mode?

The way you will read the FIFO depends on how the FIFO mode was set in the programming step 7 of the "Programming Sequence" on page 98.

Continuously Reading the FIFO (FIFO mode BLOCK)

If you are going to read the FIFO while algorithms are running you must set the FIFO mode to SENS:DATA:FIFO:MODE BLOCK. In this mode if the FIFO fills up, it stops accepting values from algorithms. The algorithms continue to execute, but the latest data is lost. To avoid losing any FIFO data, your application needs to read the FIFO often enough to keep it from overflowing. Here's a flow diagram to show you where and when to use the FIFO commands.

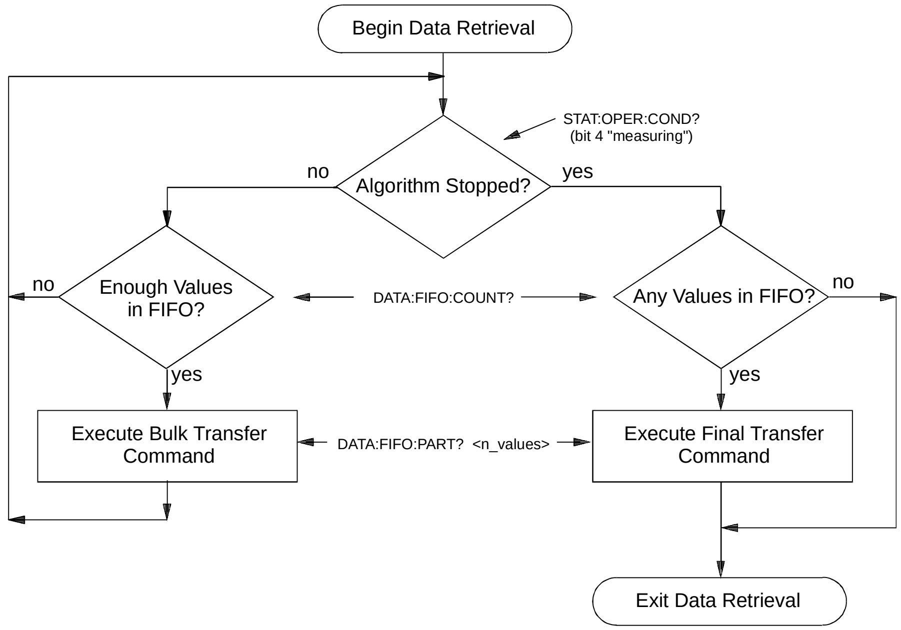


Figure 4-8. Controlling Reading Count

Here's an example command sequence for Figure 4-8. It assumes that the FIFO mode was set to BLOCK and that at least one algorithm is sending values to the FIFO.

```
        following loop reads number of values in FIFO while algorithms executing
loop while "measuring" bit is true             see STAT:OPER:COND bit 4
    SENS:DATA:FIFO:COUNT?                      query for count of values in
                                               FIFO
    input n_values here
    if n_values >= 16384                       Set minimum block size you want
                                               to transfer
        SENS:DATA:FIFO:PART?  n_values         ask for n_values
        input read_data here                   Format depends on FORMat
                                               cmd
    end if
end while loop
        following checks for values remaining in FIFO after "measuring" false
SENS:DATA:FIFO:COUNT?                          query for values still in FIFO
input n_values here
if n_values                                    if any values...
    SENS:DATA:FIFO:PART?  n_values
    input read_data here                       get remaining values from FIFO
end if
```

Reading the Latest FIFO Values (FIFO mode OVER)

In this mode the FIFO always contains the latest values (up to the FIFO's capacity of 65,024 values) from running algorithms. In order to read these values the algorithms must be stopped (use ABORT).This forms a record of the algorithm's latest performance. In the OVERwrite mode, the FIFO can not be read while it is accepting readings from algorithms. Algorithm execution must be stopped before your application program reads the FIFO.

Here is an example command sequence you can use to read values from the FIFO after algorithms are stopped (ABORT sent).

SENS:DATA:FIFO:COUNT?	*query count of values in FIFO*
input *n_values* here	
if *n_values*	*if any values...*
SENS:DATA:FIFO:PART? *n_values*	*Format of values set by FORMat*
input *read_data* here	*get remaining values from FIFO*
end of if	

For VXIplug&play users see the functions hpe1422_sensDataFifoCoun_Q, hpe1422_readFifo32_Q in your HP E1422 plug&play driver Help file.

Reading Algorithm Variables Directly

To directly read algorithm variables that are not stored in the FIFO or CVT, you only need specify the memory space(algorithm name or globals) and the name of the variable. To read the values of scalar variables or single array elements, you use the command ALG:SCALar?. To read an entire array, you use ALG:ARRay? The former returns data in ASCII, and the later returns data in REAL,64(8-byte IEEE-754 format). This coincides with the ALG:SCAL and ALG:ARR commands form writing data to these variables. Here are some examples:

```
ALG:SCAL? 'globals','my_var'
ALG:SCAL? 'alg1','my_array[6]'
ALG:ARR? 'alg2','my_other_array'
```

The ALG:ARR? response data will consist of a block header and real-64 data bytes. For example, if my_other_array was 10 elements, the block header would be #280 which says there are two bytes of count that specify 80 bytes of data to follow. Data from the E1422 is terminated with the HP-IB EOI signal.

For VXIplug&play users see the functions hpe1422_algScal_Q (for scalar variables) and hpe1422_algArray_Q (for array variables) in your HP E1422 plug&play driver Help file.

Modifying Running Algorithm Variables

Updating the Algorithm Variables and Coefficients

The values sent with the ALG:SCALAR and ALG:ARRAY command are kept in the Update Queue until an ALGorithm:UPDate command is received.

ALG:UPD	*cause changes to take place*

Updates are performed during phase 2 of the instrument operation cycle (see Figure 4-7 on page 124). The UPDate:WINDow <*num_updates*> command can be used to specify how many updates you need to perform during phase

2 (UPDATE phase) and assigns a constant window of time to accomplish all of the updates you will make. The default value for <*num_updates*> is 20. Fewer updates (shorter window) means slightly faster loop execution times. Each update takes approximately 1.4 µseconds.

To set the Update Window to allow 10 updates in phase 2:

ALG:UPD:WIND 10	*allows slightly faster execution than default of 20 updates*

A way to synchronize variable updates with an external event is to send the ALGorithm:UPDate:CHANnel '<*dig_chan/bit*>' command.

- The <*dig_chan/bit*> parameter specifies the digital channel/bit that controls execution of the update operation.

When the ALG:UPD:CHAN command is received, the module checks the current state of the digital bit. When the bit next changes state, pending updates are made in the next UPDATE Phase.

ALG:UPD:CHAN 'I133.B0'	*perform updates when bit zero of HP E1533 at channel 133 changes state*

For VXIplug&play users use the functions hpe1422_algArray, hpe1422_algScal to send new values to algorithm variables, and hpe1422_cmd to send the ALG:UPD... SCPI command. See your HP E1422 plug&play driver Help file

Enabling and Disabling Algorithms

An algorithm is enabled by default when it is defined. However, the ALG:STATe <*alg_name*>, ON | OFF command is provided to allows you to enable or disable algorithms. When an individual algorithm is enabled, it will execute when the module is triggered. When disabled, the algorithm will not execute.

NOTE The command ALG:STATE <*alg_name*>, ON | OFF does <u>not</u> take effect until an ALG:UPDATE command is received. This allows you to send multiple ALG:STATE commands and then synchronize their effect.

To enable ALG1 and ALG2, and disable ALG3 and ALG4:

ALG:STATE 'ALG1',ON	*enable algorithm ALG1*
ALG:STATE 'ALG2',ON	*enable algorithm ALG2*
ALG:STATE 'ALG3',OFF	*disable algorithm ALG3*
ALG:STATE 'ALG4',OFF	*disable algorithm ALG4*
ALG:UPDATE	*changes take effect at next update phase*

VXIplug&play users see the function hpe1422_cmd to send ALG:STATE

Setting Algorithm Execution Frequency

The ALGorithm:SCAN:RATio '<*alg_name*>',<*num_trigs*> command sets the number of trigger events that must occur before the next execution of algorithm <*alg_name*>. If you wanted 'ALG3' to execute only every 20 triggers, you would send ALG:SCAN:RATIO 'ALG3',20, followed by an ALG:UPDATE command. 'ALG3' would then execute on the first trigger after INIT, then the 21st, then the 41st, etc. This can be useful to adjust the response time of a control algorithm relative to others. The *RST default for all algorithms is to execute on every trigger event.

Example SCPI Command Sequence

This example SCPI command sequence puts together all of the steps discussed so far in this chapter.

```
*RST                                          Reset the module
    Setting up Signal Conditioning (only for programmable SCPs & RSCUs)
INPUT:FILTER:FREQUENCY  2,(@116:119)          On-board SCP channels
INPUT:FILTER:FREQUENCY  10,(@14000:14931)     128 Remote RSCU channels
INPUT:GAIN  64,(@116:119)
INPUT:GAIN  8,(@120:123)
    set up digital channel characteristics
INPUT:POLARITY  NORM,(@125)                   (*RST default)
OUTPUT:POLARITY  NORM,(@124)                  (*RST default)
OUTPUT:TYPE  ACTIVE,(@124)
    link channels to EU conversions (measurement functions)
SENSE:FUNCTION:VOLTAGE  AUTO,(@100:107)  (*RST default)
SENSE:REFERENCE  THER,5000,AUTO,(@108)
SENSE:FUNCTION:TEMPERATURE  TC,T,AUTO,(@109:123)
SENSE:REFERENCE:CHANNELS  (@108),(@109:123)
    configure digital output channel for "alarm channel"
SOURCE:FUNCTION:CONDITION  (@132)
    execute On-board channel calibration (can take several minutes)
*CAL?
enter statement here for cal return
    execute Remote channel calibration on RSCUs
CAL:REMOTE?  (@14000:14931)
enter statement here for cal return
    Direct data acquisition channels placed in Scan List. On-board channels
    00-07, and 128 remote channels covered by HP E1539 SCPs in positions 5 & 6
ROUTE:SEQUENCE:DEFINE  (@100:107,14000:14931)
    Configure the Trigger System
ARM:SOURCE IMMEDIATE                          (*RST default)
TRIGGER:COUNT  1                              (*RST default)
TRIGGER:TIMER  .010                           (*RST default)
TRIGGER:SOURCE  TIMER                         (*RST default)
    Set the channel-to-channel measurement pacing (channel settling time)
SAMPle:TIMer  4E-5                            (*RST default)
    specify data format
FORMAT  ASC,7                                 (*RST default)
    select FIFO mode
SENSE:DATA:FIFO:MODE  BLOCK                   may read FIFO while running

    Define algorithm
ALG:DEFINE 'ALG1','static float a,b,c, div, mult, sub;
                  if ( First_loop )
```

```
    {
     a = 1; b = 2; c = 3;
    writecvt( a, 10 ); writefifo( b, 11 ); writefifo( c, 12 );
    }
writecvt( a / div, 13 );
writecvt( b * mult, 14 );
writecvt( c - sub, 15 ); /* end of algorithm */'
```

Pre-set the algorithm coefficients

```
ALG:SCAL 'ALG1','div',5
ALG:SCAL 'ALG1','mult',5
ALG:SCAL 'ALG1','sub',0
ALG:UPDATE
```
all alg vars updated at this time

initiate trigger system (start algorithm)

```
INITIATE
```

retrieve algorithm data from elements 10 through 15

```
SENSE:DATA:CVT?  (@10:15,330:457)
```
enter statement here for CVT values from 6 on-board and 128 remote chans

Example VXIplug&play Driver Function Sequence

This example plug&play command sequence puts together all of the steps discussed so far in this chapter.

```
hpe1422_init(INSTR_ADDRESS, 0, 0, &vi)
hpe1422_reset(vi)                                                    Reset the module
      Setting up Signal Conditioning (only for programmable SCPs & RSCUs)
hpe1422_cmd(vi, 'INPUT:FILTER:FREQUENCY  2,(@116:119)')              On-board SCP channels
hpe1422_cmd(vi, 'INPUT:FILTER:FREQUENCY  10,(@14000:14931)')         128    Remote channels
hpe1422_cmd(vi, 'INPUT:GAIN  64,(@116:119)')
hpe1422_cmd(vi, 'INPUT:GAIN  8,(@120:123)')
   set up digital channel characteristics
hpe1422_cmd(vi, 'INPUT:POLARITY  NORM,(@125)')                       (*RST default)
hpe1422_cmd(vi, 'OUTPUT:POLARITY  NORM,(@124)')                      (*RST default)
hpe1422_cmd(vi, 'OUTPUT:TYPE  ACTIVE,(@124)')
   link channels to EU conversions (measurement functions)
hpe1422_cmd(vi, 'SENSE:FUNCTION:VOLTAGE  AUTO,(@100:107)')           (*RST default)
hpe1422_cmd(vi, 'SENSE:REFERENCE  THER,5000,AUTO,(@108)')
hpe1422_cmd(vi, 'SENSE:FUNCTION:TEMPERATURE  TC,T,AUTO,(@109:123)')
hpe1422_cmd(vi, 'SENSE:REFERENCE:CHANNELS  (@108),(@109:123)')
   configure digital output channel for "alarm channel"
hpe1422_cmd(vi, 'SOURCE:FUNCTION:CONDITION  (@132)')
   execute On-board channel calibration (can take several minutes)
hpe1422_cmdInt16_Q(vi, '*CAL?', ViPInt16 result)
test "result" for success
   execute Remote channel calibration on RSCUs
hpe1422_cmdInt16_Q(vi, 'CAL:REMOTE?  (@14000:14931)', ViPInt16 &result)
test "result" for success
   Direct data acquisition channels placed in Scan List. On-board channels 00-07, and 128 remote channels
   covered by HP E1539 SCPs in positions 5 & 6
hpe1422_cmd(vi, 'ROUTE:SEQUENCE:DEFINE  (@100:107,14000:14931)')
   Configure the Trigger System
```

```
hpe1422_cmd(vi, ’ARM:SOURCE IMMEDIATE’)                    (*RST default)
hpe1422_cmd(vi, ’TRIGGER:COUNT  1’)                        (*RST default)
hpe1422_cmd(vi, ’TRIGGER:TIMER  .010’)                     (*RST default)
hpe1422_cmd(vi, ’TRIGGER:SOURCE  TIMER’)                   (*RST default)
```

Set the channel-to-channel measurement pacing (channel settling time)

```
hpe1422_cmd(vi, ’SAMPle:TIMer  4E-5’)                      (*RST default)
```

specify data format

```
hpe1422_cmd(vi, ’FORMAT  ASC,7’)                           (*RST default)
```

select FIFO mode

```
hpe1422_cmd(vi, ’SENSE:DATA:FIFO:MODE  BLOCK’)             may read FIFO while running
```

Define algorithm. Algorithm from SCPI sequence on previoud page can be put in a text file and saved as "seqalg.c".

```
hpe1422_downloadAlg(vi, ’ALG1’, 0, ’seqalg.c’)
```

Pre-set the algorithm coefficients

```
hpe1422_algScal(vi, 'ALG1', ’div’, 5)
hpe1422_algScal(vi, 'ALG1', 'mult', 5)
hpe1422_algScal(vi, 'ALG1', 'sub', 0)
hpe1422_cmd(vi, ’ALG:UPDATE’)                              all alg vars updated at this time
```

initiate trigger system (start algorithm)

```
hpe1422_initImm(vi)
```

retrieve algorithm data from elements 10 through 15

```
hpe1422_cmdReal64Arr_Q(vi, ’SENSE:DATA:CVT?  (@10:15,330:457)’, 502, myfloat64array[ ],&count)
may test the int32 value "count" for number of cvt values reurned
```

Using the Status System

The HP E1422's Status System allows you to quickly poll a single register (the Status Byte) to see if any internal condition needs attention. Figure 4-9 shows that the three Status Groups (Operation Status, Questionable Data, and the Standard Event Groups) and the Output Queue all send summary information to the Status Byte. By this method the Status Byte can report many more events than its eight bits would otherwise allow. Figure 4-10 shows the Status System in detail.

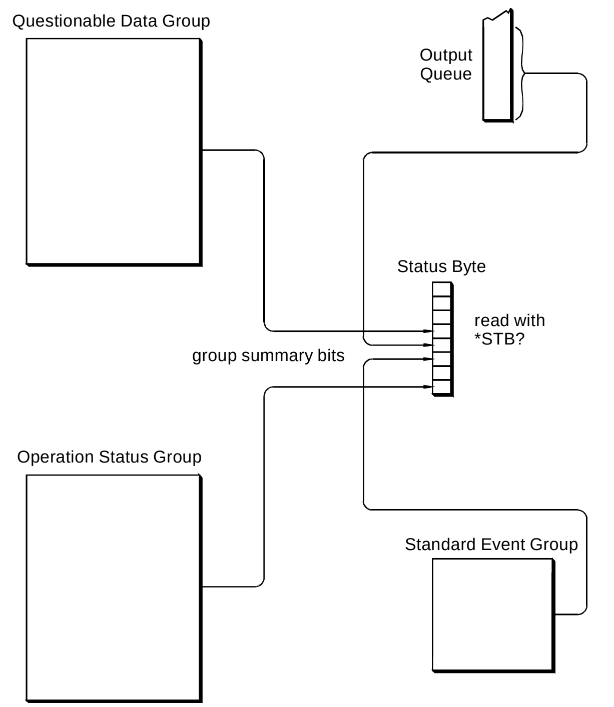


Figure 4-9. Simplified Status System Diagram

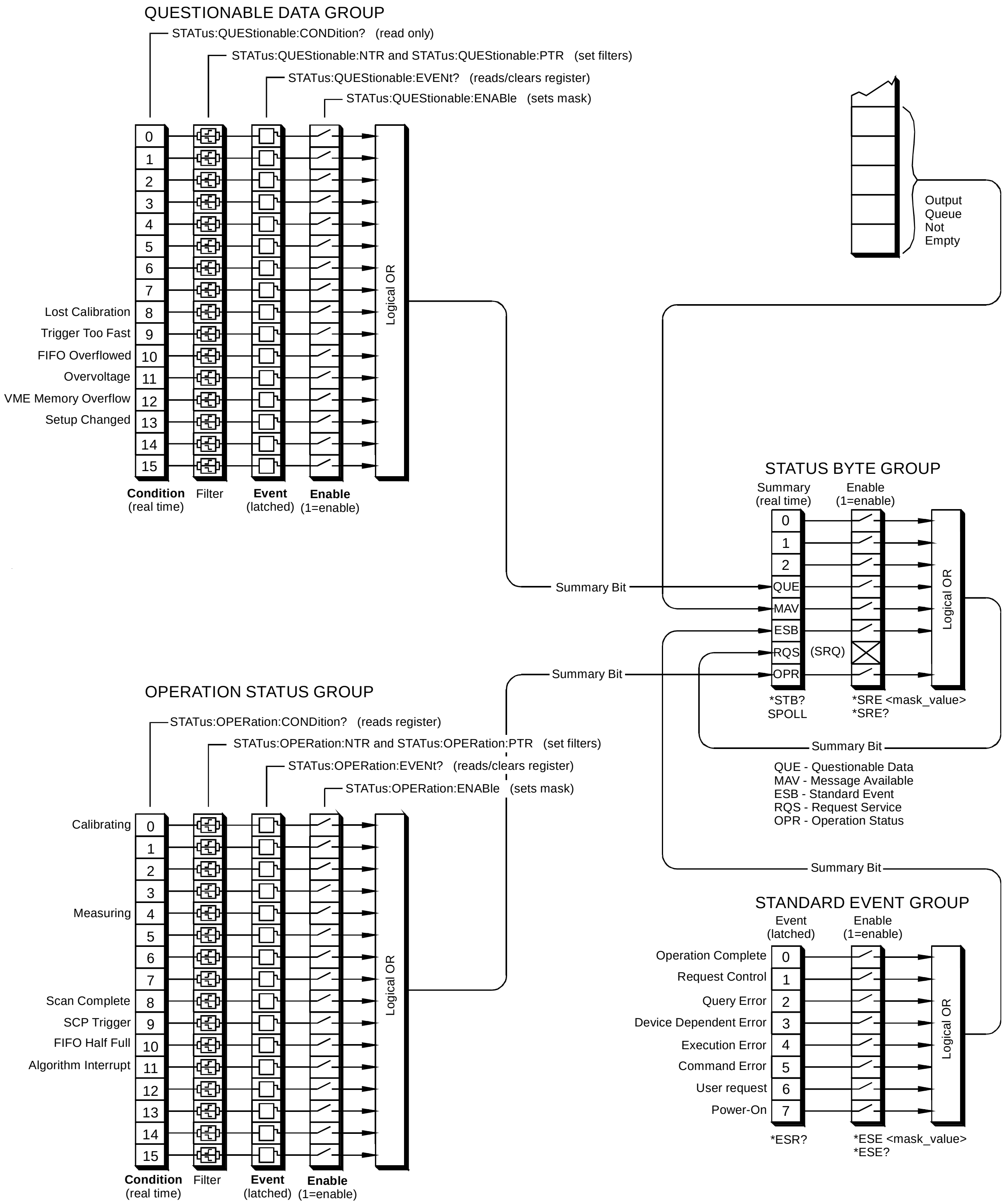


Figure 4-10. HP E1422A Status System

Status Bit Descriptions

Questionable Data Group			
Bit	**Bit Value**	**Event Name**	**Description**
8	256	Lost Calibration	At *RST or Power-on Control Processor has found a checksum error in the Calibration Constants. Read error(s) with SYST:ERR? command and re-calibrate areas that lost constants.
9	512	Trigger Too Fast	Scan not complete when another trigger event received.
10	1024	FIFO Overflowed	Attempt to store more than 65,024 values in FIFO.
11	2048	Overvoltage (Detected on Input)	If the input protection jumper has not been cut, the input relays have been opened and *RST is required to reset the module. Overvoltage will also generate an error.
12	4096	VME Memory Overflow	The number of values taken exceeds VME memory space.
13	8192	Setup Changed	Channel Calibration in doubt because SCP setup may have changed since last *CAL? or CAL:SETup command. (*RST always sets this bit.)

Operation Status Group			
Bit	**Bit Value**	**Event Name**	**Description**
0	1	Calibrating	Set by CAL:TARE, and CAL:SETup. Cleared by CAL:TARE?, and CAL:SETup?. Set while *CAL? executing, then cleared.
4	16	Measuring	Set when instrument INITiated. Cleared when instrument returns to Trigger Idle State.
8	256	Scan Complete	Set when each pass through a Scan List is completed
9	512	SCP Trigger	Reserved for future HP SCPs
10	1024	FIFO Half Full	FIFO contains at least 32,768 values
11	2048	Algorithm Interrupt	The interrupt() function was called in an executing algorithm

Standard Event Group			
Bit	**Bit Value**	**Event Name**	**Description**
0	1	Operation Complete	*OPC command executed and instrument has completed all pending operations.
1	2	Request Control	Not used by HP E1422
2	4	Query Error	Attempting to read empty output queue or output data lost.
3	8	Device Dependent Error	A device dependent error occurred. See Appendix B page 401.
4	16	Execution Error	Parameter out of range! or instrument cannot execute a proper command because it would conflict with another instrument setting.
5	32	Command Error	Unrecognized command or improper parameter count or type.
6	64	User Request	Not used by HP E1422
7	128	Power-On	Power has been applied to the instrument

Enabling Events to be Reported in the Status Byte

There are two sets of registers that individual status conditions must pass through before that condition can be reported in the instrument's Status Byte. These are the Transition Filter Registers and the Enable registers. They provide selectivity in recording and reporting module status conditions.

Configuring the Transition Filters

Figure 4-10 shows that the Condition Register outputs are routed to the input of the Negative Transition and Positive Transition Filter Registers. For space reasons they are shown together but are controlled by individual SCPI commands. It is important to understand that whether an event from the Condition Register was negative-going (NTR bit 1), or positive-going (PTR bit 1), the Event Register always records the event by setting a bit to 1. The only way Event Register Bits are changed from 1 to 0 is with the STAT:...:EVENt?, STAT:PRESet, *CLS or *RST commands. Here is the truth table for the Transition Filter Registers:

Condition Reg Bit	PTRansition Reg Bit	NTRansition Reg Bit	Event Reg Bit
0⟶1	0	0	no change
1⟶0	0	0	no change
0⟶1	1	0	1
1⟶0	1	0	no change
0⟶1	0	1	no change
1⟶0	0	1	1
0⟶1	1	1	1
1⟶0	1	1	1

The Power-on default condition is: All Positive Transition Filter Register bits set to one and all Negative Transition Filter Register bits set to 0. This applies to both the Operation and Questionable Data Groups.

An Example using the Operation Group

Suppose that you wanted the module to report via the Status System when it had completed executing *CAL?. The "Calibrating" bit (bit 0) in the Operation Condition Register goes to 1 when *CAL? is executing and returns to 0 when *CAL? is complete. In order to record only the negative transition of this bit in the STAT:OPER:EVEN register you would send:

STAT:OPER:PTR 32766	*All ones in Pos Trans Filter register except bit 0=0*
STAT:OPER:NTR 1	*All zeros in Neg Trans Filter register except bit 0=1*

Now when *CAL? completes and Operation Condition Register bit zero goes from 1 to 0, Operation Event Register bit zero will become a 1.

Configuring the Enable Registers

Figure 4-10 you will note that each Status Group has an Enable Register. These control whether or not the occurrence of an individual status condition will be reported by the group's summary bit in the Status Byte.

Questionable Data Group Examples

If you only wanted the "FIFO Overflowed" condition to be reported by the QUE bit (bit 3) of the Status Byte, you would execute;

STAT:QUES:ENAB 1024 *1024=decimal value for bit 10*

If you wanted the "FIFO Overflowed" and "Setup Changed" conditions to be reported you would execute;

STAT:QUES:ENAB 9216 *9216=decimal sum of values for bits 10 and 13*

Operation Status Group Examples

If you only wanted the "FIFO Half Full" condition to be reported by the OPR bit (bit 7) of the Status Byte, you would execute;

STAT:OPER:ENAB 1024 *1024=decimal value for bit 10*

If you wanted the "FIFO Half Full" and "Scan Complete" conditions to be reported you would execute;

STAT:OPER:ENAB 1280 *1280=decimal sum of values for bits 10 and 8*

Standard Event Group Examples

If you only wanted the "Query Error", "Execution Error", and "Command Error" conditions to be reported by the ESB bit (bit 5) of the Status Byte, you would execute;

*ESE 52 *52=decimal sum of values for bits 2, 4, and 5*

Reading the Status Byte

To check if any enabled events have occurred in the status system, you first read the Status Byte using the *STB? command. If the Status Byte is all zeros, there is no summary information being sent from any of the status groups. If the Status Byte is other than zero, one or more enabled events have occurred. You interpret the Status Byte bit values and take further action as follows:

Bit 3 (QUE) bit value 8_{10}

Read the Questionable Data Group's Event Register using the STAT:QUES:EVENT? command. This will return bit values for events which have occurred in this group. After reading, the Event Register is cleared.

Note that bits in this group indicate error conditions. If bit 8, 9 or 10 is set, error messages will be found in the Error Queue. If bit 7 is set, error messages will be in the error queue following the next *RST or cycling of power. Use the SYST:ERR? command to read the error(s).

Bit 4 (MAV)
bit value 16_{10} There is a message available in the Output Queue. You should execute the appropriate query command.

Bit 5 (ESB)
bit value 32_{10} Read the Standard Event Group's Event Register using the *ESR? command. This will return bit values for events which have occurred in this group. After reading, this status register is cleared.

Note that bits 2 through 5 in this group indicate error conditions. If any of these bits are set, error messages will be found in the Error Queue. Use the SYST:ERR? command to read these.

Bit 7 (OPR)
bit value 128_{10} Read the Operation Status Group's Event Register using the STAT:OPER:EVENT? command. This will return bit values for events which have occurred in this group. After reading, the Event Register is cleared.

Clearing the Enable Registers

To clear the Enable Registers execute:

STAT:PRESET	*for Operation Status and Questionable Data Groups*
*ESE 0	*for the Standard Event Group*
*SRE 0	*for the Status Byte Group*

The Status Byte Group's Enable Register

The Enable Register for the Status Byte Group has a special purpose. Notice in Figure 4-10 how the Status Byte Summary bit wraps back around to the Status Byte. The summary bit sets the RQS (request service) bit in the Status Byte. Using this Summary bit (and those from the other status groups) you can poll the Status Byte and check the RQS bit to determine if there are any status conditions which need attention. In this way the RQS bit is like the HP-IB's SRQ (Service Request) line. The difference is that while executing an HP-IB serial poll (SPOLL) releases the SRQ line, executing the *STB? command does not clear the RQS bit in the Status Byte. You must read the Event Register of the group who's summary bit is causing the RQS.

Reading Status Groups Directly

You may want to directly poll status groups for instrument status rather than poll the Status Byte for summary information.

Reading Event Registers

The Questionable Data, Operation Status, and Standard Event Groups all have Event Registers. These Registers log the occurrence of even temporary status conditions. When read, these registers return the sum of the decimal values for the condition bits set, then are cleared to make them ready to log further events. The commands to read these Event Registers are:

STAT:QUES:EVENT?	*Questionable Data Group Event Register*
STAT:OPER:EVENT?	*Operation Status Group Event Register*
*ESR?	*Standard Event Group Event Register*

Clearing Event Registers

To clear the Event Registers without reading them execute:

*CLS	*clears all group's Event Registers*

Reading Condition Registers

The Questionable Data and Operation Status Groups each have a Condition Register. The Condition Register reflects the group's status condition in "real-time". These registers are not latched so transient events may be missed when the register is read. The commands to read these registers are:

STAT:QUES:COND?	*Questionable Data Group Condition Register*
STAT:OPER:COND?	*Operation Status Group Condition Register*

HP E1422 Background Operation

The HP E1422 inherently runs its algorithms and calibrations in the background mode with no interaction required from the driver. All resources needed to run the measurements are controlled by the on board Control Processor (DSP).

The driver is required to setup the type of measurement to be run, modify algorithm variables, and to unload data from the card after it appears in the CVT or FIFO. Once the INIT[:IMM] command is given, the HP E1422 is initiated and all functions of the trigger system, measurement scanning, and algorithm execution are controlled by its on-board control processor. The driver returns to waiting for user commands. No interrupts are required for the HP E1422 to complete its measurement.

While the module is scanning and/or running algorithms, the driver can be queried for its status, and data can be read from the FIFO and CVT. The ABORT command may be given to force continuous execution to complete. Any changes to the measurement setup will not be allowed until the TRIG:COUNT is reached, or an ABORT command is given. Of course any commands or queries can be given to other instruments while the HP E1422 is running algorithms.

Updating the Status System and VXIbus Interrupts

The driver needs to update the status system's information whenever the status of the HP E1422 changes. This update is always done when the status system is accessed, or when CALibrate, INITiate, or ABORt commands are executed. Most of the bits in the OPER and QUES registers represent conditions which can change while the HP E1422 is measuring (initiated). In many circumstances it is sufficient to have the status system bits updated the next time the status system is accessed, or the INIT or ABORt commands are given. When it is desired to have the status system bits updated closer in time to when the condition changes on the HP E1422, the HP E1422 interrupts can be used.

The HP E1422 can send VXI interrupts upon the following conditions:

- Trigger too Fast condition is detected. Trigger comes prior to trigger system being ready to receive trigger.
- FIFO overflowed. In either FIFO mode, data was received after the FIFO was full.
- Overvoltage detection on input. If the input protection jumper has not been cut, the input relays have all been opened, and a *RST is required to reset the HP E1422.
- Scan complete. The HP E1422 has finished a scan list.
- SCP trigger. A trigger was received from an SCP.
- FIFO half full. The FIFO contains at least 32768 values.
- Measurement complete. The trigger system exited the "Wait-For-Arm". This clears the Measuring bit in the OPER register.
- Algorithm executes an "interrupt()" statement.

These HP E1422 interrupts are not always enabled since, under some circumstances, this could be detrimental to the users system operation. For example, the Scan Complete, SCP triggers, FIFO half full, and Measurement complete interrupts could come repetitively, at rates that would cause the operating system to be swamped processing interrupts. These conditions are dependent upon the user’s overall system design, therefore the driver allows the user to decide which, if any, interrupts will be enabled.

The way the user controls which interrupts will be enabled is via the *OPC, STATUS:OPER/QUES:ENABLE, and STAT:PRESET commands.

Each of the interrupting conditions listed above, has a corresponding bit in the QUES or OPER registers. If that bit is enabled via the STATus:OPER/QUES:ENABle command to be a part of the group summary bit, it will also enable the HP E1422 interrupt for that condition. If that bit is not enabled, the corresponding interrupt will be disabled.

Sending the STAT:PRESET will disable all the interrupts from the HP E1422.

Sending the *OPC command will enable the measurement complete interrupt. Once this interrupt is received and the OPC condition sent to the status system, this interrupt will be disabled if it was not previously enabled via the STATUS:OPER/QUES:ENABLE command.

The above description is always true for a downloaded driver. In the C-SCPI driver, however, the interrupts will only be enabled if cscpi_overlap mode is ON when the enable command is given. If cscpi_overlap is OFF, the user is indicating they do not want interrupts to be enabled. Any subsequent changes to cscpi_overlap will not change which interrupts are enabled. Only sending *OPC or STAT:OPER/QUES:ENAB with cscpi_overlap ON will enable interrupts.

In addition the user can enable or disable all interrupts via the SICL calls, iintron() and iintroff().

Creating and Loading Custom EU Conversion Tables

The HP E1422 provides for loading custom EU conversion tables. This allows you to have on-board conversion of transducers not otherwise supported by the HP E1422.

Standard EU Operation

The EU conversion tables built into the HP E1422 are stored in a "library" in the module's non-volatile Flash Memory. When you link a specific channel to a standard EU conversion using the [SENSe:]FUNC:… command, the module copies that table from the library to a segment of RAM allocated to the specified channel. When a single EU conversion is specified for multiple channels, multiple copies of that conversion table are put in RAM, one copy into each channel's Table RAM Segment. The conversion table-per-channel arrangement allows higher speed scanning since the table is already loaded and ready to use when the channel is scanned.

Custom EU Operation

Custom EU conversion tables are loaded directly into a channel's Table RAM Segment using the DIAG:CUST:MXB and DIAG:CUST:PIEC commands. The DIAG:CUST:… commands can specify multiple channels. To "link" custom conversions to their tables you would execute the [SENSe:]FUNC:CUST <*range*>,(@<*ch_list*>) command. Unlike standard EU conversions, the custom EU conversions are already linked to their channels (tables loaded) before you execute the [SENSe:]FUNC:CUST command but the command allows you to specify the A/D range for these channels.

NOTE The *RST command clears all channel Table RAM segments. Custom EU conversion tables must be re-loaded using the DIAG:CUST:… commands.

Custom EU Tables

The HP E1422 uses two types of EU conversion tables, linear and piecewise. The linear table describes the transducer's response slope and offset (y=mx+b). The piecewise conversion table gets its name because it is actually an approximation of the transducer's response curve in the form of 512 linear segments whose end-points fall on the curve. Data points that fall between the end-points are linearly interpolated. The built-in EU conversions for thermistors, thermocouples, and RTDs use this type of table.

Custom Thermocouple EU Conversions

The HP E1422 can measure temperature using custom characterized thermocouple wire of types E, J, K, N, R, S, and T. The custom EU table generated for the individual batch of thermocouple wire is loaded to the appropriate channels using the DIAG:CUST:PIEC command. Since thermocouple EU conversion requires a "reference junction compensation" of the raw thermocouple voltage, the custom EU table is linked to the channel(s) using the command [SENSe:]FUNCtion:CUSTom:TCouple <*type*>[,<*range*>],(@<*ch_list*>).

The <*type*> parameter specifies the type of thermocouple wire so that the correct built-in table will be used for reference junction compensation. Reference junction compensation is based on the reference junction temperature at the time the custom channel is measured. For more information see “Thermocouple Reference Temperature Compensation” on page 106.

Custom Reference Temperature EU Conversions

The HP E1422 can measure reference junction temperatures using custom characterized RTDs and thermistors. The custom EU table generated for the individually characterized transducer is loaded to the appropriate channel(s) using the DIAG:CUST:PIEC command. Since the EU conversion from this custom EU table is to be considered the "reference junction temperature", the channel is linked to this EU table using the command [SENSe:]FUNCtion:CUSTom:REFerence [<*range*>,](@<*ch_list*>).

This command uses the custom EU conversion to generate the reference junction temperature as explained in the section “Thermocouple Reference Temperature Compensation” on page 106.

Creating Conversion Tables

Contact your Hewlett-Packard System Engineer for more information on Custom Engineering Unit Conversion for your application.

Loading Custom EU Tables

There is a specific location in the E1422’s memory for each channel’s EU Conversion table. When standard EU conversions are specified, the E1422 loads these locations with EU conversion tables copied from its non-volatile FLASH Memory. For Custom EU conversions you must load these table values using either of two SCPI commands.

Loading Tables for Linear Conversions

The DIAGnostic:CUSTom:MXB <*slope*>,<*offset*>, (@<*ch_list*>) sends the <slope> and <offset> parameters that allow the driver to calculate and download a custom linear Engineering Unit Conversion table to the HP E1422A for each channel specified.

- <*slope*> specifies the linear function’s "slope": $(f_{outp1} - f_{outp0}) / (V_{in1} - V_{in0})$
- <*offset*> specifies the conversion offset at zero input volts. This parameter is also commonly known as the "Y-intercept".
- <*ch_list*> specifies which channels will have this custom EU table loaded.

Usage Example

Your program puts table constants into array *table_block*
DIAG:CUST:MXB 2.2,19,(@132:163) *send table for chs 32-63 to HP E1422*

SENS:FUNC:CUST 1,(@132:163) *link custom EU with chs 32-63 and set the 1V A/D range*

INITiate then TRIGger module

Loading Tables for Non Linear Conversions

The DIAGnostic:CUSTom:PIECewise <*table_range*>,<*table_block*>, (@<*ch_list*>) command downloads a custom piecewise Engineering Unit Conversion table to the HP E1422 for each channel specified.

- <*table_block*> is a block of 1,024 bytes that define 512 16-bit values. SCPI requires that <*table_block*> include the definite length block data header. The VXIplug&play function hpe1422_sendBlockInt16(...) adds the header for you. Contact your Hewlett-Packard System Engineer for more information on creating piecewise custom EU tables

- <*table_range*> specifies the range of input voltage that the table covers (from **-**<*table_range*> to +<*table_range*>). The value you specify must be within 5% of: .015625 | .03125 | .0625 | .125 | .25 | .5 | 1 | 2 | 4 | 8 | 16 | 32 | 64.

- <*ch_list*> specifies which channels will have this custom EU table loaded.

Usage Example

Your program puts table constants into array *table_block*
DIAG:CUST:PIEC *table_block*,1,(@124:131) *send table for chs 24-31 to HP E1422*

SENS:FUNC:CUST 1,(@124:131) *link custom EU with chs 24-31 and set the 1V A/D range*

INITiate then TRIGger module

Summary

The following points describe the capabilities of custom EU conversion:

- A given channel only has a single active EU conversion table assigned to it. Changing tables requires loading it with a DIAG:CUST:… command.

- The limit on the number of different custom EU tables that can be loaded in an HP E1422 is the same as the number of channels.

- Custom tables can provide the same level of accuracy as the built-in tables. In fact the built-in resistance function uses a linear conversion table, and the built -in temperature functions use the piecewise conversion table.

Compensating for System Offsets

System Wiring Offsets

The HP E1422 can compensate for offsets in your system's field wiring. Apply shorts to channels at the Unit-Under-Test (UUT) end of your field wiring, and then execute the CAL:TARE (@<*ch_list*>) command. The instrument will measure the voltage at each channel in <*ch_list*> and save those values in RAM as channel Tare constants.

Important Note for Thermocouples

- You must not use CAL:TARE on field wiring that is made up of thermocouple wire. The voltage that a thermocouple wire pair generates can not be removed by introducing a short anywhere between its junction and its connection to an isothermal panel (either the HP E1422's Terminal Module or a remote isothermal reference block). Thermal voltage is generated along the entire length of a thermocouple pair where there is any temperature gradient along that length. To CAL:TARE thermocouple wire this way would introduce an unwanted offset in the voltage/temperature relationship for that thermocouple. If you inadvertently CAL:TARE a thermocouple wire pair, see "Resetting CAL:TARE" on page 146.
- You should use CAL:TARE to compensate wiring offsets (copper wire, not thermocouple wire) between the HP E1422 and a remote thermocouple reference block. Disconnect the thermocouples and introduce copper shorting wires between each channel's HI and LO, then execute CAL:TARE for these channels.

Residual Sensor Offsets

To remove offsets like those in an unstrained strain gage bridge, execute the CAL:TARE command on those channels. The module will then measure the offsets and as in the wiring case above, remove these offsets from future measurements. In the strain gage case, this "balances the bridge" so all measurements have the initial unstrained offset removed to allow the most accurate high speed measurements possible.

Operation

After CAL:TARE <*ch_list*> measures and stores the offset voltages, it then performs the equivalent of a *CAL? operation. This operation uses the Tare constants to set a DAC which will remove each channel offset as "seen" by the module's A/D converter.

The absolute voltage level that CAL:TARE can remove is dependent on the A/D range. CAL:TARE will choose the lowest range that can handle the existing offset voltage. The range that CAL:TARE chooses will become the lowest usable range (range floor) for that channel. For any channel that has been "CAL:TAREd" Autorange will not go below that range floor and selecting a manual range below the range floor will return an Overload value (see the table "Maximum CAL:TARE Offsets" on page 146).

As an example assume that the system wiring to channel 0 generates a +0.1 Volt offset with 0 Volts (a short) applied at the UUT. Before CAL:TARE the module would return a reading of 0.1 Volt for channel 0. After CAL:TARE (@100), the module will return a reading of 0 Volts with a short applied at the UUT and the system wiring offset will be removed from all measurements of the signal to channel 0. Think of the signal applied to the instrument's channel input as the *gross* signal value. CAL:TARE removes the *tare* portion leaving only the *net* signal value.

Because of settling times, especially on filtered channels, CAL:TARE can take a number of minutes to execute.

The tare calibration constants created during CAL:TARE are stored in and are usable from the instrument's RAM. If you want the Tare constants to be stored in non-volatile Flash Memory you can execute the CAL:STORE TARE command.

NOTE The HP E1422's Flash Memory has a finite lifetime of approximately ten thousand write cycles (unlimited read cycles). While executing CAL:STOR once every day would not exceed the lifetime of the Flash Memory for approximately 27 years, an application that stored constants many times each day would unnecessarily shorten the Flash Memory's lifetime.

Resetting CAL:TARE

If you wish to "undo" the CAL:TARE operation, you can execute CAL:TARE:RESet then *CAL?/CAL:SET. If current Tare calibration constants have been stored in Flash Memory, execute CAL:TARE:RESET, then CAL:STORE TARE.

Special Considerations

Here are some things to keep in mind when using CAL:TARE.

Maximum Tare Capability

The tare value that can be compensated for is dependent on the instrument range and SCP channel gain settings. The following table lists these limits

Maximum CAL:TARE Offsets

A/D range ±V F.Scale	Offset V Gain x1	Offset V Gain x8	Offset V Gain x16	Offset V Gain x64
16	3.2213	.40104	.20009	.04970
4	.82101	.10101	.05007	.01220
1	.23061	.02721	.01317	.00297
.25	.07581	.00786	.00349	.00055
.0625	.03792	.00312	.00112	n/a

Changing Gains or Filters

If you decide to change a channel's SCP setup after a CAL:TARE operation you must perform a *CAL? operation to generate new DAC constants and reset the "range floor" for the stored Tare value. You must also consider the tare capability of the range/gain setup you are changing to. For instance if the actual offset present is 0.6 Volts and was "Tared" for a 4 Volt range/Gain x1 setup, moving to a 1 Volt range/Gain x1 setup will return Overload values for that channel since the 1 Volt range is below the range floor as set by CAL:TARE. See Table 6-1 on page 258 for more on values returned for Overload readings.

Unexpected Channel Offsets or Overloads

This can occur when your HP E1422's Flash Memory contains CAL:TARE offset constants that are no longer appropriate for its current application. Execute CAL:TARE:RESET then *CAL? to reset the tare constants in RAM. Measure the affected channels again. If the problems go away, you can now reset the tare constants in Flash memory by executing CAL:STORE TARE.

Detecting Open Transducers

Most of the HP E1422's analog input SCPs provide a method to detect open transducers. When Open Transducer Detect (OTD) is enabled, the SCP injects a small current into the HIGH and LOW input of each channel. The polarity of the current pulls the HIGH inputs toward +17 volts and the LOW inputs towards -17 volts. If a transducer is open, measuring that channel will return an over-voltage reading. OTD is available on a per SCP basis. All eight channels of an SCP are enabled or disabled together. See Figure 4-11 for a simplified schematic diagram of the OTD circuit.

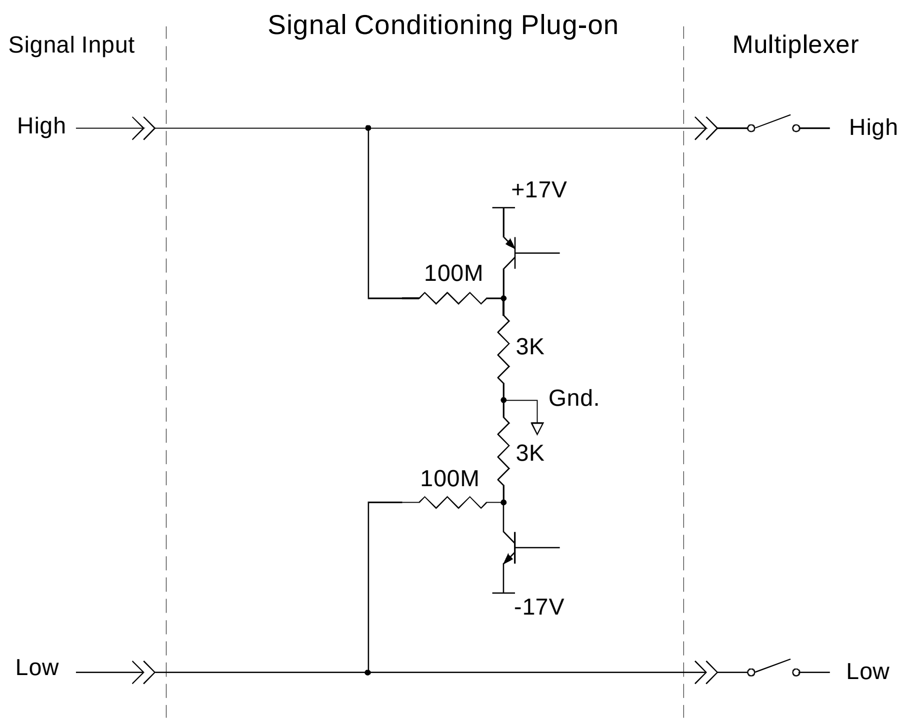


Figure 4-11. Simplified Open Transducer Detect Circuit

NOTES

1. When OTD is enabled, the inputs have up to 0.2µA injected into them. If this current will adversely affect your measurement, but you still want to check for open transducers, you can enable OTD, run your algorithms, check analog input variables for measurement values that indicate an open transducer, then disable OTD and run your algorithms without it. The HP E1422's accuracy specifications apply only when OTD is off.
2. When a channel's SCP filtering is enabled, allow 15 seconds after turning on OTD for the filters capacitors to charge before checking for open transducers.

To enable or disable Open Transducer Detection, use the DIAGnostic:OTDetect[:STATe] <*enable*>, (@<*ch_list*>) command.

- The *enable* parameter can specify ON or OFF
- An SCP is addressed when the *ch_list* parameter specifies a channel number contained on the SCP. The first channel on each SCP is: 0, 8, 16, 24, 32, 40, 48, and 56

To enable Open Transducer Detection on all channels on SCPs 1 and 3:

```
DIAG:OTD ON, (@100,116)
```
0 is on SCP 1 and 16 is on SCP3

To disable Open Transducer Detection on all channels on SCPs 1 and 3:

```
DIAG:OTD OFF, (@100,116)
```

More On Auto Ranging

There are rare circumstances where your input signal can be difficult for the HP E1422 to auto range correctly. The module completes the range selection based on your input signal about 6 µsec before the actual measurement is made on that channel. If during that period your signal becomes greater than the selected range can handle, the module will return an overflow reading (±INFinity).

To cure this problem, use the DIAGnostic:FLOor <*range*>,(@<*ch_list*>) command. Include the problem channel(s) in <*ch_list*> and specify the lowest range you want auto range to select for those channels. This will set a range "floor" for these channels that auto range can't go below while still allowing auto range to select higher ranges as necessary. If you need to specify more than one range floor for different channel sets, execute the DIAG:FLOOR command multiple times.

The DIAGnostic:FLOor:DUMP command sends the current range floor settings for all 64 channels to the FIFO. Use DATA:FIFO:PART? 64 to read these values.

The auto range floor settings remain until another DIAG:FLOOR command changes them, or a *RST resets them to the lowest range for all channels.

Settling Characteristics

Some sequences of input signals as determined by their order of appearance in a scan list can be a challenge to measure accurately. This section is intended to help you determine if your system presents any of these problems and how best to eliminate them or reduce their effect.

Background

While the HP E1422 can auto-range, measure, and convert a reading to engineering units as fast as once every 10 µs, measuring a high level signal followed by a very low level signal may require some extra settling time. As seen from the point of view of the HP E1422's Analog-to-Digital converter and its Range Amplifier, this situation is the most difficult to measure. For example lets look at two consecutive channels; the first measures a power supply at 15.5 volts, the next measures a thermocouple temperature. First the input to the Range Amplifier is at 15.5 volts (near its maximum) with any stray capacitances charged accordingly, then it immediately is switched to a thermocouple channel and down-ranged to its .0625 volt range. On this range, the resolution is now 1.91 µvolt per Least Significant Bit (LSB). Because of this sensitivity, the time to discharge these stray capacitances may have to be considered.

Thus far in the discussion, we've assumed that the low-level channel measured after a high-level channel has presented a low impedance path to discharge the A/D's stray capacitances (path was the thermocouple wire). The combination of a resistance measurement through an HP E1501 Direct Input SCP presents a much higher impedance path. A very common measurement like this would be the temperature of a thermistor. If measured through a Direct Input SCP, the source impedance of the measurement is essentially the value of the thermistor (the output impedance of the current source is in the gigohm region). Even though this is a higher level measurement than the previous example, the settling time can be even longer due to the slower discharge of the stray capacitances. The simple answer here is to always use an SCP that presents a low impedance buffered output to the HP E1422's Range Amp and A/D. The HP E1503, 8, 9, 10, 12, and 14 through 17 SCPs all provide this capability.

Checking for Problems

The method we'll use to quickly determine if any of your system's channels needs more settling time is to simply apply some settling time to every channel. Use this procedure:

1. First run your system to make a record of its current measurement performance.

2. Then use the SAMPle:TIMer command to add a significant settling delay to every measurement in the scan list. Take care that the sample time multiplied by the number of channels in the scan list doesn't exceed the time between triggers.

3. Now run your system and look primarily for low level channel measurements (like thermocouples) who's DC value changes somewhat. If you find channels that respond to this increase in sample

period, you may also notice that these channels may return slightly quieter measurements as well. The extra sample period reduces or removes the affected channels coupling to the value of the channel measured just before it.

4. If you see some improvement, increase the sample period again and perform another test. When you increase the sample period and no improvement is seen, you have found the maximum settling delay that any single channel requires.

5. If the quality of the measurements does not respond to this increase in sample period, then inadequate settling time is not likely to be causing measurement problems.

Fixing the Problem

If your system scans fast enough with the increased sample period, your problem is solved. Your system is only running as fast as the slowest channel allows but if its fast enough that's OK. If on the other hand, getting quality readings has slowed your scan rate too much, there are two other methods that will, either separately or in combination, have your system making good measurements as fast as possible.

Use Amplifier SCPs

Amplifier SCPs can remove the need to increase settling delays. How? Each gain factor of 4 provided by the SCP amplifier allows the Range Amplifier to be set one range higher and still provide the same measurement resolution. Amplifier SCPs for the HP E1422 are available with gains of .5, 8, 16, 64, and 512. Lets return to our earlier example of a difficult measurement where one channel is measuring 15.5 volts on the 16 volt range, and the next a thermocouple on the .0625 range. If our thermocouple channel is amplified through an SCP with a gain of 16, the Range Amplifier can be set to the 1 volt range. On this range the A/D resolution drops to around 31 µvolt per LSB so the stray capacitances discharging after the 15.5 volt measurement are now only one sixteenth as significant and thus reduce any required settling delay. Of course for most thermocouple measurements we can use a gain of 64 and set the Range Amplifier to the 4 volt range. At this setting the A/D resolution for one LSB drops to about 122 µvolts and further reduces or removes any need for additional settling delay. This improvement is accomplished without any reduction of the overall measurement resolution.

NOTE Filter-amplifier SCPs can provide improvements in low-level signal measurements that go beyond just settling delay reduction. Amplifying the input signal at the SCP allows using less gain at the Range Amplifier (higher range) for the same measurement resolution. Since the Range Amplifier has to track signal level changes (from the multiplexer) at up to 100 KHz, its bandwidth must be much higher than the bandwidth of individual filter-amplifier SCP channels. Using higher SCP gain along with lower Range Amplifier gain can significantly increase normal-mode noise rejection.

Adding Settling Delay for Specific Channels

This method adds settling time only to individual problem measurements as opposed to the SAMPle:TIMer command that introduces extra time for all analog input channels. If you see problems on only a few channels, you can use the SENS:CHAN:SETTLING <*num_samples*>,(@<*ch_list*>) command to add extra settling time for just these problem channels. What SENS:CHAN:SETTLING does is instruct the HP E1422 to replace single instances of a channel in the Scan List with multiple repeat instances of that channel if it is specified in (@<*ch_list*>). The number of repeats is set by <*num_samples*>.

Example:

Normal Scan List:
100, 101, 102, 103, 104

Scan List after SENS:CHAN:SETT 3,(@100,103)
100, 100, 100, 101, 102, 103, 103, 103, 104

When the algorithms are run, channels 0 and 3 will be sampled 3 times, and the final value from each will be sent to the Channel Input Buffer. This provides extra settling time while channels 1, 2, and 4 are measured in a single sample period and their values also sent to the Channel Input Buffer.

Chapter 5

Creating and Running Algorithms

Learning Hint This chapter builds upon the "HP E1422 Programming Model" information presented in Chapter 4. You should read that section before moving on to this one.

About This Chapter

This chapter describes how to write algorithms that apply the HP E1422's measurement, calculation, and control resources. It describes these resources and how you can access them with the HP E1422's Algorithm Language. This manual assumes that you have programming experience already. Ideally, you have programmed in the 'C' language since the HP E1422's Algorithm Language is based on 'C'. Following the tutorial sections of this chapter is an Algorithm Language Reference. The contents of this chapter are:

Overview of the Algorithm Language

The HP 1422A's Algorithm Language is a limited version of the 'C' programming language. It is designed to provide the necessary control constructs and algebraic operations to support measurement and control algorithms. There are no loop constructs, multi-dimensional arrays, or transcendental functions. Further, an algorithm must be completely contained within a single function subprogram 'ALGn'. The algorithm can not call another user-written function subprogram.

It is important to note, that while the HP E1422A's Algorithm Language has a limited set of intrinsic arithmetic operators, it also provides the capability to call special user defined functions "f(x)". Appendix E page 419shows you how to convert user defined functions into piece-wise linear interpolated tables which can be downloaded into the HP E1422A. The HP E1422A can extract function values from these tables in approximately 18µseconds regardless of the function's original complexity. This method provides faster algorithm execution by moving the complex math operations off-board.

This section assumes that you already program in some language. If you are already a 'C' language programmer, this chapter is all you'll probably need to create your algorithm. If you are not familiar with the C programming language, you should study the "Program Structure and Syntax" section before you begin to write your algorithms.

This section will present a quick look at the Algorithm Language. The complete language reference is provided later in this chapter.

Arithmetic Operators: add **+**, subtract **-**, multiply *****, divide **/**
NOTE: See "Calling User Defined Functions" on page 162.

Assignment Operator: =

Comparison Functions: less than **<**, less than or equal **<=**, greater than **>**, greater than or equal **>=**, equal to **==**, not equal to **!=**

Boolean Functions: and **&&**, or **||**, not **!**

Variables: scalars of type **static float**, and single dimensioned arrays of type **static float** limited to 1024 elements.

Constants:
32-bit decimal integer; **Dddd**... where **D** and **d** are decimal digits but **D** is not zero. No decimal point or exponent specified.
32-bit octal integer; **0oo**... where **0** is a leading zero and **o** is an octal digit. No decimal point or exponent specified.
32-bit hexadecimal integer; **0Xhhh**... or **0xhhh**... where **h** is a hex digit.
32-bit floating point; **ddd.**, **ddd.ddd**, **ddde±dd**, **dddE±dd**, **ddd.dddedd**, or **ddd.dddEdd** where d is a decimal digit.

Flow Control: conditional construct **if(){ } else { }**

Intrinsic Functions:
Return minimum; **min(***<expr1>***,***<expr2>***)**

Return maximum;   **max(**_<expr1>_**,**_<expr2>_**)**
User defined function;   _<user_name>_**(**_<expr>_**)**
Write value to CVT element;   **writecvt(**_<expr>_**,**_<expr>_**)**
Write value to FIFO buffer;   **writefifo(**_<expr>_**)**
Write value to both CVT and FIFO;   **writeboth(**_<expr>_**,**_<expr>_**)**

Example Language Usage

Here are examples of some Algorithm Language elements assembled to show them used in context. Later sections will explain any unfamiliar elements you see here:

Example 1;

```
/*** get input from channel 8, calculate output, check limits, output to ch 16 & 17 ***/
static float output_max = .020;         /* 20 mA max output */
static float output_min = .004;         /* 4 mA min output */
static float input_val, output_val;     /* intermediate I/O vars */
static float remote_input_val;          /* I/O var for remote channel*/

input_ val = I108;                      /* get value from input buffer channel 8*/
remote_input_ val = I14001;             /* get value from input buffer channel 4001*/
output_val = 12.5 * input_val;          /* calculate desired output */
if ( output_val > output_max )          /* check output greater than limit */
        output_val = output_max;        /* if so, output max limit */
else if( output_val < output_min)       /* check output less than limit */
        output_val = output_min;        /* if so, output min limit */
O116 = output_val / 2;                  /* split output_val between two SCP  */
O117 = output_val / 2;                  /* channels to get up to 20mA max    */
writecvt(remote_input_val,501);         /* remote chan val to CVT element 501 */
```

Example 2;

```
/*** same function as example 1 above but shows a different approach ***/
static float max_output = .020;         /* 20 mA max output */
static float min_output = .004;         /* 4 mA min output */

/* following lines input, limit output between min and max_output, and outputs.    */
/* output is split to two current output channels wired in parallell to provide 20mA */
/* write cvt is just to show access to remote channel
O116 = max( min_output, min( max_output, (12.5 * I108) / 2 ) );
O117 = max( min_output, min( max_output, (12.5 * I108) / 2 ) );
writecvt(I14001,501);
```

The Algorithm Execution Environment

This section describes the execution environment that the HP E1422 provides for your algorithms. Here we describe the relationship of your algorithm to the **main()** function that calls it.

The Main Function

All 'C' language programs consist of one or more functions. A 'C' program must have a function called **main()**. In the HP E1422, the **main()** function is usually generated automatically by the driver when you execute the INIT command. The **main()** function executes each time the module is triggered, and controls execution of your algorithm functions. See Figure 5-1 for a partial listing of **main()**.

How Your Algorithms Fit In

When the module is INITiated, a set of control variables and a function calling sequence is created for all algorithms you have defined. The value of variable "State_n" is set with the ALGorithm:STATe command and determines whether the algorithm will be called. The value of "Ratio_n" is set with the ALGorithm:SCAN:RATio command and determines how often the algorithm will be called (relative to trigger events).

Since the function-calling interface to your algorithms is fixed in the **main()** function, the "header" of your algorithm function is also pre-defined. This means that unlike standard 'C' language programming, your algorithm program (a function) need not (must not) include the function declaration header, opening brace "{", and closing brace "}". You only supply the "body" of your function, the HP E1422's driver supplies the rest.

Think of the program space in the HP E1422 in the form of a source file with any global variables first, then the **main()** function followed by as many algorithms as you have defined. Of course what is really contained in the HP E1422's algorithm memory are executable codes that have been translated from your downloaded source code. While not an exact representation of the algorithm execution environment, Figure 5-1 shows the relationship between a normal 'C' program and two HP E1422 algorithms.

Global variables area

First_loop declared by HP E1422's driver

Begin main() function (built by HP E1422's driver)

End main() function

Begin algorithm "shells" (built by HP E1422's driver)

Your algorithms go here

```
/* GLOBALS you define with ALG:DEF GLOBALS...  go here */

/* global variable First_loop equals 1 until all algorithms called  */
static float First_loop; /* global value set to 1 at each INIT */
/**************************** function main() ****************************/
/*The HP E1422 driver creates main() at INIT time. This example
shows a main created after 2 algorithms have been defined. */
main()
{
  /********* declaration of variables local to main() ********/
  static float State_1, Ratio_1, Count_1;  /* created if alg1 defined */
  static float State_2, Ratio_2, Count_2;  /* created if alg2 defined */

  /********* this section created if ALG1 is defined ********/
  Count_1 = Count_1 - 1; /* Count_1 used for ALG:SCAN:RATIO */
  if (Count_1 <= 0) {        /* test for ratio met (<=0 means execute)*/
     Count_1 = Ratio_1;   /* Count_1 = ALG:SCAN:RATIO setting  */
     if (State_1) alg1();     /* if ALG:STATE ALG1,ON, call alg1       */
  }

  /********* this section created if ALG2 is defined ********/
  Count_2 = Count_2 - 1; /* Count_2 used for ALG:SCAN:RATIO */
  if (Count_2 <= 0) {        /* test for ratio met (<=0 means execute)*/
     Count_2 = Ratio_2;   /* Count_2 = ALG:SCAN:RATIO setting  */
     if (State_2) alg2();     /* if ALG:STATE ALG2,ON, call alg2       */
  }
  First_loop = 0;   /* reset First_loop after last alg has been called */
}
/* ************************ end function main() ************************/

ALG1()  /* this function shell created by ALG:DEF 'ALG1'....    */
{
const int ALG_NUM = 1; /* created by driver to ID this algorithm */

/********* Your algorithm code goes here **********/
}

ALG2()  /* this function shell created by ALG:DEF 'ALG2'....    */
{
const int ALG_NUM = 2; /* created by driver to ID this algorithm */

/********* Your algorithm code goes here **********/
}
```

Figure 5-1. Source Listing of Function mai

Accessing the E1422's Resources

This section describes how your algorithm accesses hardware and software resources provided by the HP E1422. The following is a list of these resources:

- I/O channels.
- Global variables defined before your algorithm is defined.
- The constant ALG_NUM which the HP E1422 makes available to your algorithm. ALG_NUM = 1 for ALG1, 2 for ALG2 etc.
- User defined functions defined with the ALG:FUNC:DEF command.
- The Current Value Table (CVT), and the data FIFO buffer (FIFO) to output algorithm data to your application program.
- VXIbus Interrupts.

Accessing I/O Channels

In the Algorithm Language, channels are referenced as pre-defined variable identifiers. Because input channels could be from Remote Signal Conditioning Units (RSCUs), there are two forms of input channel syntax. The general on-board input channel identifier syntax is "I1cc" where cc is a channel number from 00 (channel 0) through 63 (channel 63). The Remote input channel syntax is "I1ccrr" where cc is the SCP channel number (one of 00, 01, 08, 09, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, or 57), and rr is the channel (0 through 31) on the RSCU see the heading "Remote Channels:" on page 199 for more information. For output channels the syntax is "O1cc" where cc is a channel number from 00 (channel 0) through 63 (channel 63). Like all HP E1422 variables, channel identifier variables always contain 32-bit floating point values even when the channel is part of a digital I/O SCP. If the digital I/O SCP has 8-bit channels (like the HP E1533), the channel's identifiers (O1cc and I1cc) can take on the values 0 through 255. To access individual bit values you may append ".Bn" to the normal channel syntax; where n is the bit number (0 through 7). If the Digital I/O SCP has single-bit channels (like the HP E1534), its channel identifiers can only take on the values 0 and 1. Examples:

O100 = 1;	*assign value to output chan 0 on HP E1534.*
Inp_val = I108;	*from 8-bit channel on HP E1533 Inp_val will be 0. to 255. np_val will be 0. to 255.*
Bit_4 = I109.B4;	*assign HP E1533 chan 9 bit 4 to variable Bit_4*

Output Channels

Output channels can appear on either or both sides of an assignment operator. They can appear anywhere other variables can appear. Examples:

O100 = 12.5;	*send value to output channel buffer element 0*
O108.B4 = ! O108.B4;	*compliment value found in output channel buffer element 8, bit 4 each time algorithm is executed.*
writecvt(O116,350);	*send value of output channel 16 to CVT element 350*

Input Channels

Input channel identifiers can only appear on the right side of assignment operators. It doesn't make sense to output values to an input channel. Other than that, they can appear anywhere other variables can appear. Examples:

dig_bit_value = I108.B0;	*retrieve value from Input Channel Buffer element 8, bit 0*
inp_value = I124;	*retrieve value from Input Channel Buffer element 24*
rscu_value = I12422;	*retrieve value from RSCU Channel Buffer element 2422*
O156 = 4 * I124;	*retrieve value from Input Channel Buffer element 24, multiply by 4 and send result to*

```
writefifo(I124);
```

Output Channel Buffer element 56

send value of input channel 24 to FIFO buffer

Defined Input and Output Channels

Your algorithm "references" channels. It can reference input or output channels. But, in order for these channels to be available to your algorithm they must be "defined". What we mean by "defined" is that an SCP or RSCU must be installed, and an appropriate SOURce or SENSe :FUNCtion must explicitly (or implicitly, in the case of HP E1531&32 SCPs) be tied to the channels. If your algorithm references an input channel identifier that is not configured as an input channel, or an output channel identifier that is not configured as an output channel, the driver will generate an error when your algorithm is defined with ALG:DEF.

Defining and Accessing Global Variables

Global variables are those declared outside of both the **main()** function and any algorithms (see Figure 5-1). A global variable can be read or changed by any algorithm. To declare global variables you use the command:

ALG:DEF 'GLOBALS','<*source_code*>'

where <*source_code*> is Algorithm Language source limited to constructs for declaring variables. It must not contain executable statements. Examples:

declare single variable without assignment;

```
ALG:DEF 'GLOBALS','static float glob_scal_var;'
```

declare single variable with assignment;

```
ALG:DEF 'GLOBALS','static float glob_scal_var = 22.53;'
```

declare one scalar variable and one array variable;

```
ALG:DEF 'GLOBALS','static float glob_scal_var, glob_array_var[12];'
```

You access global variables within your algorithm like any other variable.

```
glob_scal_var = P_factor * I108
```

NOTES

1. All variables must be declared static float.
2. Array variables cannot be assigned a value when declared.
3. All variables declared within your algorithm are local to that algorithm. If you locally declare a variable with the same identifier as an existing global variable, your algorithm will access the local variable only.

Determining First Execution (First_loop)

The HP E1422 always declares the global variable *First_loop*. *First_loop* is set to 1 each time INIT is executed. After **main()** calls all enabled algorithms it sets *First_loop* to 0. By testing *First_loop*, your algorithm can determine if it is being called for the first time since an INITiate command was received. Example:

```
static float scalar_var;
static float array_var [ 4 ];

/* assign constants to variables on first pass only */
if ( First_loop )
{
        scalar_var = 22.3;
        array_var[0] = 0;
        array_var[1] = 0;
        array_var[2] = 1.2;
        array_var[3] = 4;
}
```

Initializing Variables

Variable initialization can be performed during three distinct HP E1422 operations.

1. When you define algorithms with the ALG:DEFINE command. A declaration initialization statement is a command to the driver's translator function and doesn't create an executable statement. The value assigned during algorithm definition is not re-assigned when the algorithm is run with the INIT command. Example statement:

   ```
   static float my_variable = 22.95;/* tells translator to allocate space for this */
                                    /* variable and initialize it to 22.95           */
   ```

2. Each time the algorithm executes. By placing an assignment statement within your algorithm. This will be executed each time the algorithm is executed. Example statment.

   ```
   my_variable = 22.95;/* reset variable to 22.95 every pass        */
   ```

3. When the algorithm first executes after an INIT command. By using the global variable *First_loop* your algorithm can distinguish the first execution since an INIT command was sent. Example statement:

   ```
   if( First_loop ) my_variable = 22.95  /* reset variable only when INIT starts alg */
   ```

Sending Data to the CVT and FIFO

The Current Value Table (CVT) and FIFO data buffer provide communication from your algorithm to your application program (running in your VXIbus controller). The three algorithm functions; writecvt(), writefifo(), and writeboth() provide the means to place data into the FIFO or CVT. These special functions may be called up to 512 times.

Writing a CVT element

The CVT provides 502 addressable elements where algorithm values can be stored (see Figure 6-4 on page 288). To send a value to a CVT element, you will execute the intrinsic Algorithm Language statement **writecvt(<*expression*>,<*cvt_element*>)**, where <*cvt_element*> can take the value 10 through 511. The following is an example algorithm statement:

```
writecvt(O124, 330);  /* send output channel 24's value to CVT element 330 */
```

Each time your algorithm writes a value to a CVT element the previous value in that element is overwritten.

Important! There is a fixed relationship between channel number and CVT element for values from channels placed in the Scan List with ROUT:SEQ:DEF. When you are mixing Scan List data acquisition with algorithm data storage, be careful not to overwrite Scan List generated values with algorithm generated values. See “ROUTe:SEQuence:DEFine” on page 287. for controlling CVT entries from the analog scan list.

Reading CVT elements

Your application program reads one or more CVT elements by executing the SCPI command **[SENSe:]DATA:CVT? (@<*element_list*>)**, where <*element_list*> specifies one or more individual elements and/or a range of contiguous elements. The following example command will help to explain the <*element_list*> syntax.

DATA:CVT? (@10,20,30:33,40:43,330)	*Return elements 10, 20, 30-33, 40-43. and element 330.*

Individual element numbers are isolated by commas. A contiguous range of elements is specified by: <starting element>colon<ending element>.

Writing values to the FIFO

The FIFO, as the name implies is a First-In-First-Out buffer. It can buffer up to 65,024 values. This capability allows your algorithm to send a continuous stream of data values related in time by their position in the buffer. This can be thought of as an electronic strip-chart recorder. Each value is sent to the FIFO by executing the Algorithm Language intrinsic statement **writefifo(<*expression*>)**. The following in an example algorithm statement:

```
writefifo(O124);  /* send output channel 24's value to the FIFO */
```

Since you can determine the actual algorithm execution rate (see “Programming the Trigger Timer” on page 122), the time relationship of readings in the FIFO is very deterministic.

Reading values from the FIFO

For a discussion on reading values from the FIFO, see “Reading Running Algorithm Values” on page 125.

Writing values to the FIFO and CVT

The **writeboth(<*expression*>,<*cvt_element*>)** statement sends the value of <*expression*> both to the FIFO and to a <*cvt_element*>. Reading these values is done the same way as mentioned for **writefifo()** and **writecvt()**.

Setting a VXIbus Interrupt

The algorithm language provides the function **interrupt()** to force a VXIbus interrupt. This **interrupt()** function can be used to set bit 11 of the STATus:OPERation register from within an algorithm. This bit could then be enabled to generate an SRQ to the controller (see “STATus:OPERation:ENABle” on page 334 and “*SRE” on page 356). The following example algorithm code tests an input channel value and sets an interrupt if it is higher or lower than set limits.

```
static float upper_limit = 1.2, lower_limit = 0.2;
if( I124 > upper_limit || I124 < lower_limit ) interrupt();
```

Determining Your Algorithm’s Identity (ALG_NUM)

When you define your algorithm with the ALG:DEF 'ALGn',... command, the E1422's driver makes available to your algorithm the constant *ALG_NUM*. *ALG_NUM* has the value **n** from "ALG**n**". For instance, if you defined an algorithm with <*alg_name*> equal to "ALG3", then *ALG_NUM* within that algorithm would have the value 3.

What can you do with this value? To give you an idea, here's a short example of the code that uses *ALG_NUM*:

```
writecvt ( inp_channel, (ALG_NUM * 10) + 0 );
writecvt ( Error, (ALG_NUM * 10) + 1 );
writecvt ( outp_channel, (ALG_NUM * 10) + 2 );
writecvt ( Status, (ALG_NUM * 10) + 3 );
```

This code writes algorithm values into CVT elements 10 through 13 for ALG1, CVT elements 20 through 23 for ALG2, CVT elements 30 through 33 for ALG3 etc.

Using *ALG_NUM* allows you to write identical code that can take different actions depending on the name it was given when defined.

Calling User Defined Functions

Access to user defined functions is provided to avoid complex equation calculation within your algorithm. Essentially what is provided with the HP E1422 is a method to pre-compute user function values outside of algorithm execution and place these values in tables, one for each user function. Each function table element contains a slope and offset to calculate an mx+b over the interval (x is the value you provide to the function). This allows the DSP to linearly interpolate the table for a given input value and return the function's value much faster than if a transcendental function's equation were arithmetically evaluated using a power series expansion.

User functions are defined by downloading function table values with the ALG:FUNC:DEF command and can take any name that is a valid 'C' identifier like 'haversine', 'sqr', 'log10' etc. To find out how to generate table values from your function equation, see "Generating User Defined Functions" in Appendix E page 419. For details on the ALG:FUNC:DEF command, see page 217 in the Command Reference.

User defined functions are global in scope. A user function defined with ALG:FUNC:DEF is available to all defined algorithms. Up to 32 functions can be defined in the HP E1422. You call your function with the syntax <func_name>(<expression>). Example:

for user function pre-defined as square root with name ’sqrt’

```
O108 = sqrt( I100);  /* channel 8 outputs square root of input channel 0's value */
```

NOTE A user function must be defined (ALG:FUNC:DEF) before any algorithm is defined (ALG:DEF) that references it.

A VXIplug&play program that shows the use of a user defined function is supplied on the examples disc in file "tri_sine.cpp". The program is on the CD supplied with your instrument. View the readme.txt file provided with the VXIplug&play driver for example program file location.

Operating Sequence

This section explains another important factor in your algorithm's execution environment. Figure 5-2 shows the same overall sequence of operations that you saw in Chapter 3, but also includes a block diagram to show you which parts of the HP E1422 are involved in each phase of the control sequence.

Overall Sequence

Here, the important things to note about this diagram are:

- All algorithm referenced input channel values are stored in the Channel Input Buffer (Input Phase) BEFORE algorithms are executed during the Calculate Phase.
- The execution of all defined algorithms (Calculate Phase) is complete BEFORE output values from algorithms, stored in the Channel Output Buffer, are used to update the output channel hardware during the Output Phase.

In other words, algorithms don't actually read inputs at the time they reference input channels, and they don't send values to outputs at the time they reference output channels. Algorithms read channel values from an input buffer, and write (and can read) output values to/from an output buffer. Here are example algorithm statements to describe operation:

```
inp_val = I108;/* inp_val is assigned a value from input buffer element 8   */
O116 = 22.3;/* output buffer element 16 assigned the value 22.3              */
O125 = O124;/* output buffer [24] is read and assigned to output buffer [25] */
```

A Common Error to Avoid

Since the buffered input, algorithm execution, buffered output sequence is probably not a method many are familiar with, a programming mistake associated with it is easy to make. Once you see it here, you won't do this in your programs. The following algorithm statements will help explain:

```
O124.B0 = 1;/* digital output bit on HP E1533 in SCP position 3 */
O124.B0 = 0;
```

Traditionally you expect the first of these two statements to set output channel 24, bit 0 to a digital 1, then after the time it takes to execute the second statement, the bit would return to a digital 0. Because both of these statements are executed BEFORE any values are sent to the output hardware, only the last statement has any effect. Even if these two statements were in separate algorithms, the last one executed would determine the output value. In this example the bit would never change. The same applies to analog outputs.

Algorithm Execution Order

The buffered I/O sequence explained previously can be used to advantage. Multiple algorithms can access the very same buffered channel input value without having to pass the value in a parameter. Any algorithm can read and use as its input, the value that any other algorithm has sent to the output buffer. In order for these features to be of use you must know the order in which your algorithms will be executed. When you define your algorithms you give them one of 32 pre-defined algorithm names. These range from ’ALG1’ to ALG32’. Your algorithms will execute in order of its name. For instance if you define ’ALG5’, then ’ALG2’, then ’ALG8’, and finally ’ALG1’, when you run them they will execute in the order ’ALG1’, ’ALG2’, ’ALG5’, and 'ALG8'. For more on input and output value sharing, see “Algorithm to Algorithm Communication” on page 171.

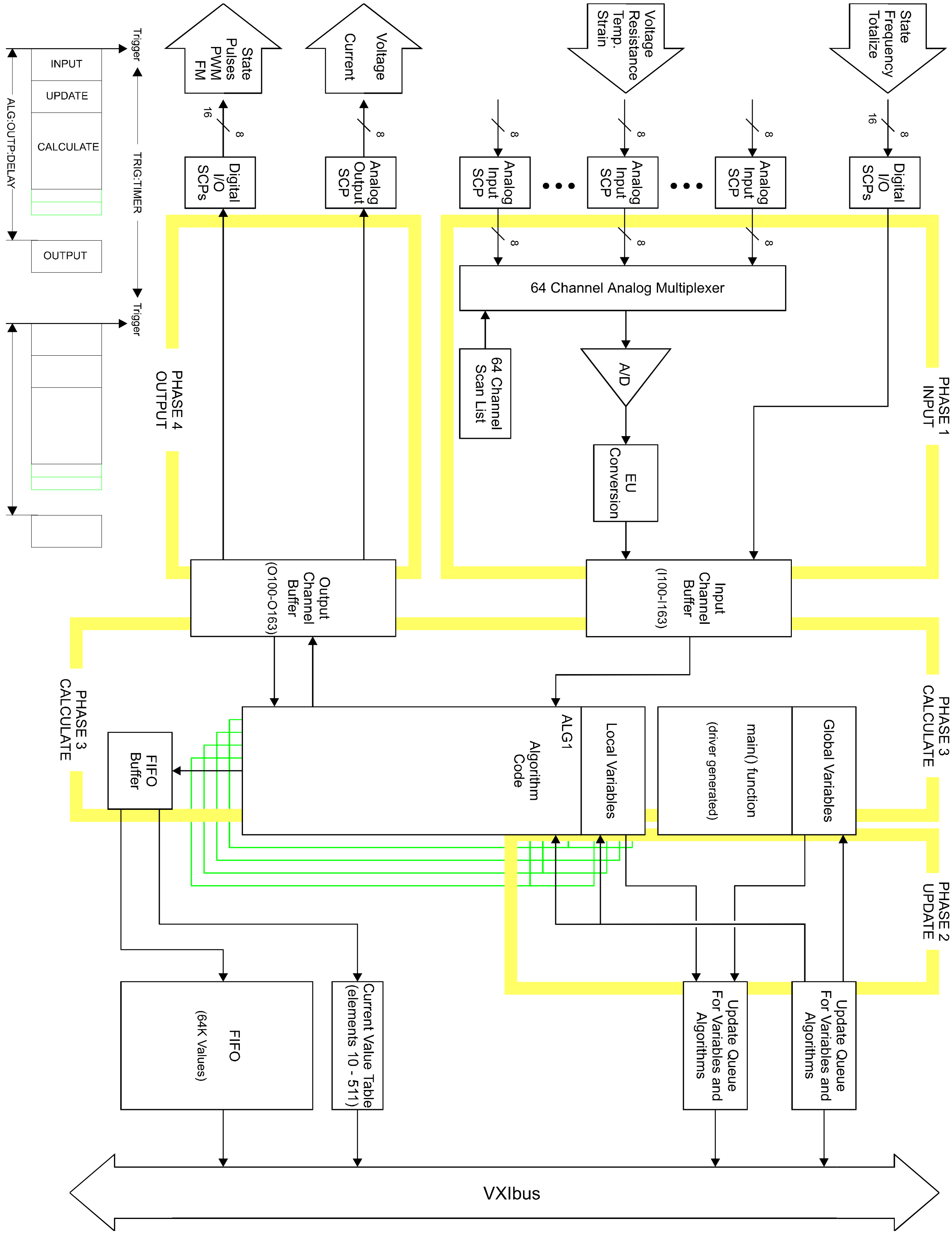


Figure 5-2. Algorithm Operating Sequence Diagram

Defining Algorithms (ALG:DEF)

This section discusses how to use the ALG:DEFINE command to define algorithms. Later sections will discuss "what to define".

Note for VXIplug&play users

While the following discussion of algorithm definition is useful for plug&play users as regards the coding of the algorithm or global variable definition, the method of generating the algorithm code and actually down-loading it to the HP E1422 becomes much easier because of plug&play hpe1422.exe Soft Front Panel program, and hpe1422_downloadAlg(...) plug&play driver function.

Using the SFP "Algorithm Panel", you can create and test your algorithm, and then store it to a file. The hpe1422_downloadAlg(...) plug&play driver function was created specifically to download algorithms from files into your HP E1422A as part of your application program.

ALG:DEFINE in the Programming Sequence

*RST erases all previously defined algorithms. You must erase all algorithms before you begin to re-define them (except in the special case described in "Changing an Algorithm While it's Running" later in this section).

ALG:DEFINE's Three Data Formats

The ALG:DEFINE '*<alg_name>*','*<source_code>*' command sends the algorithm's source code to the HP E1422's driver for translation to executable code. The *<source_code>* parameter can be sent in one of two forms:

1. SCPI Quoted String: For short segments (single lines) of code, enclose the code string within single (apostrophes), or double quotes. Because of string length limitations within SCPI and some programming platforms, we recommend that the quoted string length not exceed a single program line. Example:

```
ALG:DEF 'ALG1','if(First_loop) O108=0; O108=O108+.01;'
```

2. SCPI Indefinite Length Block Program Data: This form terminates the data transfer when it received an End Identifier with the last data byte. Use this form only when you are sure your controller platform will include the End Identifier. If it is not included, the ALG:DEF command will "swallow" whatever data follows the algorithm code. The syntax for this parameter type is:
   #0<data byte(s)><null byte with End Identifier>
   Example from "Quoted String" above:
   ALG:DEF 'ALG1',#0O108=I100;∅ (where "∅" is a null byte)

NOTE For Block Program Data, the Algorithm Parser requires that the *source_code* data end with a null (0) byte. You must append the null byte to the end of the block's <data byte(s)>. If the null byte is not included within the block, the error "Algorithm Block must contain termination '\0'" will be generated.

Indefinite Length Block Data Example

Retrieve algorithm source code from file and send to HP E1422 in indefinite length format using VTL/VISA instrument I/O libraries:

```
int byte_count, file_handle;
char source_buffer[8096], null = 0;
file_handle = open( "<filename>", O_RDONLY + O_BINARY);
byte_count = read( file_handle, source_buffer, sizeof( source_buffer ) );
close( file_handle );
source_buffer[ byte_count ] = 0; /* null to terminate source buffer string */
viPrintf( e1422,  "ALG:DEF  'ALG8',#0%s%c\n", source_buffer, null );
```

See the section "Running the Algorithm" later in this chapter for more on loading algorithms from files.

Changing an Algorithm While it's Running

The HP E1422 has a feature that allows you to specify that a given algorithm can be swapped with another even while it is executing. This is useful if, for instance, you needed to alter the function of an algorithm that is currently controlling a process and you don't want to leave that process uncontrolled. In this case, when you define the original algorithm, you can enable it to be swapped.

Defining an Algorithm for Swapping

The ALG:DEF command has an optional parameter that is used to enable algorithm swapping. The command's general form is:

```
ALG:DEF '<alg_name>'[,<swap_size>],'<source_code>'
```

Note the parameter <*swap_size*>. With <*swap_size*> you specify the amount of algorithm memory to allocate for algorithm <*alg_name*>. Make sure to allocate enough space for the largest algorithm you expect to define for <*alg_name*>. Here is an example of defining an algorithm for swapping:

define ALG3 so it can be swapped with an algorithm as large as 1000 words

```
ALG:DEF 'ALG3',1000,#41698<1698char_alg_source>
```

NOTE The number of characters (bytes) in an algorithm's <*source_code*> parameter is not well related to the amount of memory space the algorithm requires. Remember this parameter contains the algorithm's source code, not the executable code it will be translated into by the ALG:DEF command. Your algorithm's source might contain extensive comments, none of which will be in the executable algorithm code after it is translated.

How Does it Work?

We'll use the example algorithm definition above for this discussion. When you specify a value for <*swap_size*> at algorithm definition, the HP E1422 allocates two identical algorithm spaces for ALG3, each the size specified by <*swap_size*> (in this example 1000 words). This is called a "double buffer". We'll just call these space A and space B. The algorithm is loaded into ALG3's space A at first definition. Later, while algorithms are running you can "replace" ALG3 by again executing

```
ALG:DEF ALG3,#42435<2435char_alg_source>
```

Notice that <swap_size> is not (must not be) included this time. This ALG:DEF works like an Update Request. The HP E1422 translates and downloads the new algorithm into ALG3's space B while the old ALG3 is still running from space A. When the new algorithm has been completely loaded into space B and an ALG:UPDATE command has been sent, the HP E1422 simply switches to executing ALG3's new algorithm from space B at the next Update Phase (see Figure 5-2. If you were to send yet another ALG3, it would be loaded and executed from ALG3's space A.

Determining an Algorithm's Size

In order to define an algorithm for swapping, you will need to know how much algorithm memory to allocate for it or any of its replacements. You can query this information from the HP E1422. Use the following sequence:

1. Define the algorithm without swapping enabled. This will cause the HP E1422 to allocate only the memory actually required by the algorithm.
2. Execute the ALG:SIZE? <*alg_name*> command to query the amount of memory allocated. You now know the minimum amount of memory required for the algorithm.
3. Repeat 1 and 2 for each of the algorithms you want to be able to swap with the original. From this you know the minimum amount of memory required for the largest.
4. Execute *RST to erase all algorithms.
5. Re-define one of the algorithms with swapping enabled and specify <swap_size> at least as large as the value from step 3 above (and probably somewhat larger because as alternate algorithms declare different variables, space is allocated for total of all variables declared).
6. Swap each of the alternate algorithms for the one defined in step 5, ending with the one you want to run now. Remember, you don't send the <swap_size> parameter with these. If you don't get an "Algorithm too big" error, then the value for <swap_size> in step 5 was large

enough.

7. Define any other algorithms in the normal manner.

NOTES

1. Channels referenced by algorithms when they are defined, are only placed in the channel list before INIT. The channel list cannot be changed after INIT. If you re-define an algorithm (by swapping) after INIT, and it references channels not already in the channel list, it will not be able to access the newly referenced channels. No error message will be generated. To make sure all required channels will be included in the channel list, define <*alg_name*> and re-define all algorithms that will replace <*alg_name*> by swapping them before you send INIT. This insures that all channels referenced in these algorithms will be available after INIT.

2. The driver only calculates overall execution time for algorithms defined before INIT. This calculation is used to set the default output delay (same as executing ALG:OUTP:DELAY AUTO). If an algorithm is swapped after INIT that take longer to execute than the original, the output delay will behave as if set by ALG:OUTP:DEL 0, rather than AUTO (see ALG:OUTP:DEL command). Use the same procedure from note 1 to make sure the longest algorithm execution time is used to set ALG:OUTP:DEL AUTO before INIT.

An example program file named "swap.cpp" on the drivers CD shows how to swap algorithms while the module is running. See Appendix F page 423 for program listings. View the readme.txt file provided with the VXIplug&play driver for example program file location.

A Very Simple First Algorithm

This section will show you how to create and download an algorithm that simply sends the value of an input channel to a CVT element. It includes an example application program that configures the HP E1422, downloads (defines) the algorithm, starts and then communicates with the running algorithm.

Writing the Algorithm

The most convenient method of creating your algorithm is to use the hpe1422.exe soft front panel program. Use the Algorithms Panel to create, edit, and save the algorithm to a file called "mxplusb.c". The following algorithm source code is on the examples disc in a file called "mxplusb.c".

```
/* Example algorithm that calculates 4 Mx+B values upon
 * signal that sync == 1.  M and B terms set by application
 * program.
 */
 static float M, B, x, sync;
 if ( First_loop ) sync = 0;
 if ( sync == 1 ) {
        writecvt(  M*x+B,   10 );
        writecvt(-(M*x+B),  11 );
        writecvt( (M*x+B)/2,12 );
        writecvt( 2*(M*x+B),13 );
        sync = 2;
 }
```

Running the Algorithm

A C-SCPI example program "file_alg.cpp" shows how to retrieve the algorithm source file "mxplusb.c" and use it to define and execute an algorithm. When you have compiled "file_alg.cpp", type file_alg mxplusb.c to run the example and load the algorithm. View the readme.txt file provided with the VXIplug&play driver for example program file location.

Modifying an Example PID Algorithm

While the example PID algorithms supplied as source files with your HP E1422A can provide excellent general closed loop process control, there will be times when your process has specialized requirements that are not addressed by the as-written form of these PID algorithms. In this section we show you how to copy and modify an example PID algorithm.

PIDA with digital On-Off Control

The example PID algorithms are written to supply control outputs through analog output SCPs. While it would not be an error to specify a digital channel as the PID control output, the PID algorithm as written would not operate the digital channel as you would desire.

The value you write to a digital output bit is evaluated as if it were a boolean value. That is, if the value represents a boolean true, the digital output is set to a binary 1. If the value represents a boolean false, the digital output is set to a binary 0. The HP E1422's Algorithm Language (like 'C') specifies that a value of 0 is a boolean false (0), any other value is considered true (1). With that in mind we'll analyze the operation of an example PIDA with a digital output as its control output.

How the Example PIDA Operates

A PID algorithm is to control a bath temperature at 140 degrees Fahrenheit. With the Setpoint at 140 and the process variable (PV) reading 130, the value sent to the output is a positive value which drives the digital output to 1 (heater on). When the process value reading reaches 140 the "error term" would equal zero so the value sent to the digital output would be 0 (heater off). Fine so far, but as the bath temperature coasts even minutely above the setpoint, a small negative value will be sent to the digital output which represents a boolean true value. At this point the output will again be 1 (heater on) and the bath temperature will continue go up rather than down. This process is now out of control!

Modifying the Example PIDA

This behavior is easy to fix. We'll just modify the example PIDA algorithm source code (supplied with your HP E1422 in the file PIDA.C) and then define it as an algorithm. Use the following steps.

1. Load the source file for the example PIDA algorithm into your favorite text editor.
2. Find the line of code near the end of PIDA that reads:

   ```
   outchan = Error * P_factor + I_out + D_factor * (Error - Error_old)
   ```

   and insert this line below it:

   ```
   if ( outchan <= 0 ) outchan = 0; /* all values not positive are now zero */
   ```

3. going back to the beginning of the file, change all occurrences of "inchan" to the input channel specifier of your choice (e.g. I100).
4. As in step 3, change all occurrences of "outchan" to the digital output channel/bit identifier of your choice (e.g. O108.B0).
5. Now save this algorithm source file as "ONOFFPID.C".

Algorithm to Algorithm Communication

The ability for one algorithm to have access to values from another can be very important particularly in more complex control situations. One of the important features of the HP E1422 is that this communication can take place entirely within the algorithms' environment. Your application program is freed from having to retrieve values from one algorithm and then send those values to another algorithm.

Communication Using Channel Identifiers

The value of all defined input and output channels can be read by any algorithm. Here is an example of inter-algorithm channel communication.

Implementing Multivariable Control

In this example, two PID algorithms each control part of a process and due to the process dynamics are interactive. This situation can call for what is known as a "decoupler". The job of the decoupler is to correct for the "coupling" between these two process controllers. Figure 5-3 shows the two PID controllers and how the de-coupler algorithm fits into the control loops. As mentioned before, algorithm output statements don't write to the output SCP channels but are instead buffered in the Output Channel Buffer until the Output Phase occurs. This situation allows easy implementation of decouplers because it allows an algorithm following the two PIDs to inspect their output values and make adjustments to them before they are sent to output channels. The decoupler algorithm's *Decoupl_factor1* and Decouple_factor2 variables (assumes a simple interaction) are local and can be independently set using ALG:SCALAR:

```
/* decoupler algorithm. (must follow the coupled algorithms in execution sequence)  */
static float Decouple_factor1, Decouple_factor2;
O124 = O124 + Decouple_factor2 * O125;
O125 = O125 + Decouple_factor1 * O124;
```

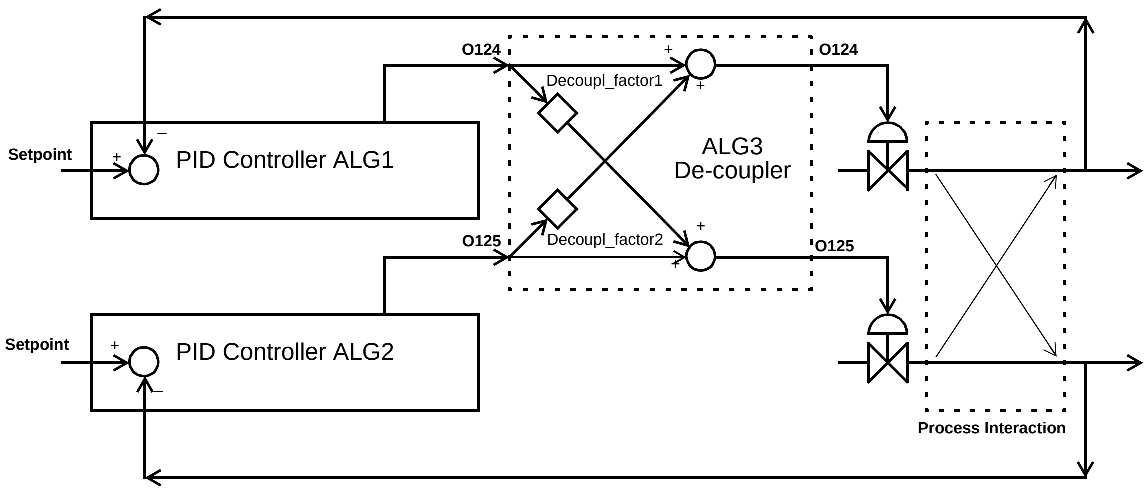


Figure 5-3. Algorithm Communication with Channels

Communication Using Global Variables

A more traditional method of inter-algorithm communication uses global variables. Global variables are defined using the ALG:DEF command in the form:
ALG:DEF 'GLOBALS','<variable_declaration_statements>'

Example of global declaration

ALG:DEF 'GLOBALS','static float cold_setpoint;'

Implementing Feed Forward Control

In this example two algorithms mix hot and cold water supplies in a ratio that results in a tank being filled to a desired temperature. The temperature of the make-up supplies is assumed to be constant. Figure 5-4 shows the process diagram.

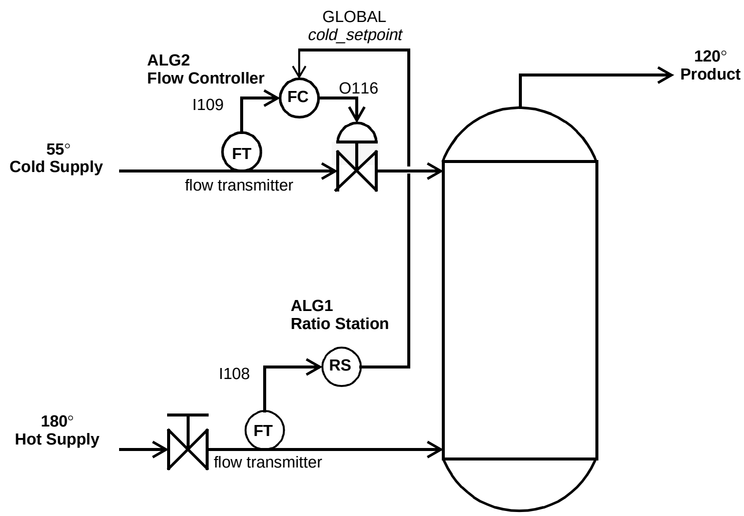


Figure 5-4. Inter-algorithm Communication with Globals

To set up the algorithms for this example:

1. Define the global variable *cold_setpoint*

```
ALG:DEF ’GLOBALS’,’static float cold_setpoint;’
```

2. Define the following algorithm language code as ALG1, the ratio station algorithm.

```
static float hot_flow, cold_hot_ratio;
static float cold_temp = 55, hot_temp = 180, product_temp = 120;
hot_flow = I108;     /* get flow rate of cold supply */
/* following line calculates cold to hot ratio from supply and product temps */
cold_hot_ratio = (hot_temp - product_temp) / (cold_temp - product_temp);
cold_setpoint = hot_flow * cold_hot_ratio;  /* output flow setpoint for ALG2 */
```

3. Modify a PIDA algorithm so its setpoint variable is the global variable *cold_setpoint*, its input channel is I109, and its output channel is O116, and Define as ALG2, the cold supply flow controller:

```
/*  Modified PIDA Algorithm; comments stripped out, setpoint from global,
    inchan = I109, outchan = O116
*/
     /* the setpoint is not declared so it will be global */
    static float P_factor = 1;
    static float I_factor = 0;
    static float D_factor = 0;
    static float I_out;
    static float Error;
    static float Error_old;
```

```
/* following line includes global setpoint var, and hard coded input chan */
Error = Cold_setpoint - I109;
if (First_loop)
{
     I_out = Error * I_factor;
     Error_old = Error;
}
else /* not First trigger */
{
     I_out = Error * I_factor + I_out;  /* output channel hard coded here */
}
O116 = Error * P_factor + I_out + D_factor * (Error - Error_old);
Error_old = Error;
```

Non-Control Algorithms

Process Monitoring Algorithm

Another function the HP E1422 performs well is monitoring input values and testing them against pre-set limits. If an input value exceeds its limit, the algorithm can be written to supply an indication of this condition by changing a CVT value, or even forcing a VXIbus interrupt. The following example shows acquiring one analog input value from channel 0, and one HP E1533 digital channel from channel 16, and limit testing them.

```
/* Limit test inputs , send values to CVT, and force interrupt when exceeded */
static float Exceeded;
static float Max_chan0, Min_chan0, Max_chan1, Min_chan1;
static float Max_chan2, Min_chan2, Max_chan3, Min_chan3;
static float Mask_chan16;
if ( First_loop ) Exceeded = 0;  /* initialize Exceeded on each INIT */
writecvt( I100, 330); /* write analog value to CVT */
Exceeded = ( ( I100 > Max_chan0 ) || ( I100 < Min_chan0 ) ); /* limit test analog */
writecvt( I101, 331); /* write analog value to CVT */
Exceeded = Exceeded + ( ( I101 > Max_chan1 ) || ( I101 < Min_chan1 ) );
writecvt( I102, 332); /* write analog value to CVT */
Exceeded = Exceeded + ( ( I102 > Max_chan2 ) || ( I102 < Min_chan2 ) );
writecvt( I103, 333); /* write analog value to CVT */
Exceeded = Exceeded + ( ( I103 > Max_chan3 ) || ( I103 < Min_chan3 ) );
writecvt( I116, 334); /* write 8-bit value to CVT */
Exceeded = Exceeded + ( I116 != Mask_chan16); /* limit test digital */
If ( Exceeded ) interrupt( );
```

Implementing Setpoint Profiles

A setpoint profile is a sequence of setpoints you wish to input to a control algorithm. A normal setpoint is either static or modified by operator input to some desired value where it will then become static again. A setpoint profile is used when you want to cycle a device under test through some operating range, and the setpoint remains for some period of time before changing. The automotive industry uses setpoint profiles to test their engines and drive trains. That is, each new setpoint is a simulation of an operator sequence that might normally be encountered.

A setpoint profile can either be calculated for each interval or pre-calculated and placed into an array. If calculated, the algorithm is given a starting setpoint and an ending setpoint. A function based upon time then calculates each new desired setpoint until traversing the range to the end point. Some might refer to this technique as setpoint ramping.

Most setpoint profiles are usually pre-calculated by the application program and downloaded into the instrument performing the sequencing. In that case, an array affords the best alternative for several reasons:

- Arrays can hold up to 1024 points.
- Arrays can be downloaded quickly while the algorithm is running.
- Time intervals can be tied to trigger events and each N trigger events can simply access the next element in the array.
- Real-time calculations of setpoint profiles by the algorithm itself complicates the algorithm.
- The application program has better control over time spacing and the complexity and range of the data. For example; succesive points in the array could be the same value just to keep the setpoint at that position for extra time periods.

The following is an example program that sequences data from an array to an Analog Output. There are some unique features illustrated here that you can use:

- The application program can download new profiles while the application program is running. The algorithm will continue to sequence through the array until it reaches the end of the array. At which time, it will set its index back to 0 and toggle a Digital Output bit to create an update channel condition on a Digital Input. Then at the next trigger event, the new array values will take effect before the algorithm executes. As long as the new array is download into memory before the index reaches 1023, the switch to the new array elements will take place. If the array is downloaded AFTER the index reaches 1023, the same setpoint profile will be executed until index reaches 1023 again.
- The application program can monitor the index value with ALG:SCAL? "alg1","index" so it can keep track of where the profile sequence is currently running. The interval can also be made shorter or longer by changing the num_events variable.

```
SOUR:FUNC:COND (@141)
```

make Digital I/O channel 141 a digital output. The default condition for 140 is digital input.

define algorithm

```
ALG:DEF ’alg1’,’
static float setpoints[ 1024 ], index, num_events, n;
if ( First_loop ) {
        index = 0;    /* array start point */
        n = num_events;/* preset interval */
}
n = n - 1;              /* count trigger events */
if ( n <= 0 ) {
        O100 = setpoints[ index ]; /* output new value */
```

```
        index = index + 1;   /* increment index */
        if ( index > 1023 ) {    /* look for endpoint */
                index = 0;
                O140.B0 = !O140.B0; /* toggle update bit */
        }
        n = num_events;   /* reset interval count */
}'
```

ALG:SCAL "alg1","num_events", 10	*output change every 10msec*
ALG:ARRAY "alg1","setpoints",<block_data>	*set first profile*
ALG:UPD	*force change*
TRIG:TIMER .001	*trigger event at 1msec*
TRIG:SOUR TIMER	*trigger source timer*
INIT	*start algorithm*

Download new setpoint profile and new timer interval:

ALG:SCAL "alg1","num_events", 20	*output change every 20msec*
ALG:ARRAY "alg1","setpoints",<block data>	*set first profile*
ALG:UPD:CHAN "I140.B0"	*change takes place with change in bit 0 of O140.*

This example program was configured using Digital Output and Digital Inputs for the express reason that multiple E1422A's may be used in a system. In this case, the E1422A toggling the digital bit would be the master for the other E1422A's in the system. They all would be monitoring one of their digital input channels to signal a change in setpoint profiles.

Algorithm Language Reference

This section provides a summary of reserved keywords, operators, data types, constructs, intrinsic functions and statements.

Standard Reserved Keywords

The list of reserved keywords is the same as ANSI 'C'. You may not create your own variables using these names. Note that the keywords that are shown underlined and bold are the only ANSI 'C' keywords that are implemented in the HP E1422.

auto	double	int	struc
break	**<u>else</u>**	long	switch
case	enum	register	typeof
char	extern	**<u>return</u>**	union
const	**<u>float</u>**	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	**<u>if</u>**	**<u>static</u>**	while

NOTE While all of the ANSI 'C' keywords are reserved, only those keywords that are shown in bold are actually implemented in the HP E1422.

Special HP E1422 Reserved Keywords

The HP E1422 implements some additional reserved keywords. You may not create variables using these names:

abs	interrupt	writeboth
Bn (n=0 through 9)	max	writecvt
Bnn (nn=10 through 15)	min	writefifo

Identifiers

Identifiers (variable names) are significant to 31 characters. They can include alpha, numeric, and the underscore character "_". Names must begin with an alpha character, or the underscore character.

Alpha: a b c d e f g h i j k l m n o p q r s t u v w x y z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Numeric: 0 1 2 3 4 5 6 7 8 9
Other: _

NOTE Identifiers are case sensitive. The names My_array and my_array reference different variables.

Special Identifiers for Channels

Channel identifiers appear as variable identifiers within the algorithm and have a fixed, reserved syntax. The identifiers I100 to I163 specify on-board input channel numbers. The identifiers I10000 to I15731 specify remote input channel numbers. The "I" must be upper case. They may only appear on the right side of an assignment operator. The identifiers O100 to O163 specify output channel numbers. The "O" must be upper case. They can appear on either or both sides of the assignment operator.

NOTE Trying to declare a variable with a channel identifier will generate an error.

Operators

The HP E1422’s Algorithm Language supports the following operators:

Assignment Operator

=	(assignment)	example;	c = 1.2345

Arithmetic Operators

+	(addition)	examples;	c = a + b
-	(subtraction)		c = a - b
*	(multiplication)		c = a * b
/	(division)		c = a / b

Unary Operators

-	(unary minus)		c = a + (-b)
+	(unary plus)		c = a + (+b)

Comparison Operators

==	(is equal to)	examples;	a == b
!=	(is not equal to)		a != b
<	(is less than)		a < b
>	(is greater than)		a > b
<=	(is less than or equal to)		a <= b
>=	(is greater than or equal to)		a >= b

Logical Operators

||	(or)	examples;	(a == b) || (a == c)
&&	(and)		(a == b) && (a == c)

Unary Logical Operator

!	(not)	example;	!b

The result of a comparison operation is a boolean value. It is still a type **float** but its value is either 0 (zero) if false, or 1 (one) if true. You may test any variable with the **if** statement. A value of zero tests false, if any other value it tests true. For example:

/* if my_var is other than 0, increment count_var */

```
if(my_var) count_var=count_var+1;
```

Intrinsic Functions and Statements

The following functions and statements are provided in the HP E1422's Algorithm Language:

Functions:

abs(*expression*)	return absolute value of expression
max(*expression1*,*expression2*)	return largest of the two expressions
min(*expression1*,*expression2*)	return smallest of the two expressions

Statements:

interrupt()	sets bit 11 of STAT:OPER register
writeboth(*expression*,*cvt_loc***)**	write expression result to FIFO and CVT element specified.
writecvt(*expression*,*cvt_loc***)**	write expression result to CVT element specified.
writefifo(*expression***)**	write expression result to FIFO.

Note The sum of the number of calls to writefifo(), writecvt() and writeboth() must not exceed 512.

Program Flow Control

Program flow control is limited to the conditional execution construct using **if** and **else**, and **return**. Looping inside an algorithm function is not supported. The only "loop" is provided by repeatedly triggering the HP E1422. Each trigger event (either external, or internal Trigger Timer) executes the **main()** function which calls each defined and enabled algorithm function. There is no **goto** statement.

Conditional Constructs

The HP E1422 Algorithm Language provides the **if-else** construct in the following general form:

if (*expression***)** *statement1* **else** *statement2*

If *expression* evaluates to non-zero *statement1* is executed. If *expression* evaluates to zero, *statement2* is executed. The else clause with its associated *statement2* is optional. Statement1 and/or statement2 can be compound statement. That is { *statement; statement; statement; ...* }.

Exiting the Algorithm

The **return** statement allows terminating algorithm execution before reaching the end by returning control to the main() function. The **return** statement can appear anywhere in your algorithm. You are not required to include a **return** statement to end an algorithm. The translator treats the end of your algorithm as an implied return.

Data Types

The data type for variables is always **static float**. However decimal constant values without a decimal point or exponent character (".", "E", or "e"), as well as Hex and Octal constants are treated as 32-bit integer values. This treatment of constants is consistent with ANSI 'C'. To understand what this can mean you must understand that not all arithmetic statements in your algorithm are actually performed within the HP E1422's DSP chip at algorithm run-time. Where expressions can be simplified, the HP E1422's translator (a function of the driver invoked by ALG:DEF) performs the arithmetic operations before downloading the executable code to the algorithm memory in the HP E1422. For example look at the statement;

```
a = 5 + 8;
```

When the HP E1422's translator receives this statement, it simplifies it by adding the two integer constants (5 and 8) and storing the sum of these as the float constant 13. At algorithm run-time, the float constant 13 is assigned to the variable "a". No surprises so far. Now analyze this statement;

```
a = ( 3 / 4 ) * 12;
```

Again the translator simplifies the expression by performing the integer divide for 3 / 4. This results in the integer value 0 being multiplied by 12 which results in the float constant 0.0 being assigned to the variable "a" at run-time. This is obviously not what you wanted but is exactly what your algorithm instructed.

You can avoid these subtle problems by specifically including a decimal point in decimal constants where an integer operation is not what you want. For example, if you had made either of the constants in the division above a float constant by including a decimal point, the translator would have promoted the other constant to a float value and performed a float divide operation resulting in the expected 0.75 * 12, or the value 8.0 So the statement;

```
a = ( 3. / 4 ) * 12;
```

will result in the value float 8.0 being assigned to the variable "a".

The Static Modifier

All HP E1422 variables, local or global, must be declared as **static**. An example:

```
static float gain_var, integer_var, deriv_var;    /* three vars declared */
```

In 'C', local variables that are not declared as **static** lose their values once the function completes. The value of a local **static** variable remains unchanged between calls to your algorithm. Treating all variables this way allows your algorithm to "remember" its previous state. The static variable is local in scope, but otherwise behaves as a global variable. Also note that you may not declare variables within a compound statement.

Data Structures

The HP E1422 Algorithm Language allows the following data structures:

- Simple variables of type **float**:
  Declaration
  ```
  static float simp_var, any_var;
  ```
  Use

```
simp_var = 123.456;
any_var = -23.45;
Another_var = 1.23e-6;
```

Storage

Each simple variable requires four 16-bit words of memory.

- Single-dimensioned arrays of type **float** with a maximum of 1024 elements:

Declaration

```
static float array_var [3];
```

Use

```
array_var [0] = 0.1;
array_var [1] = 1.2;
array_var [2] = 2.34;
array_var [3] = 5;
```

Storage

Arrays are "double buffered". This means that when you declare an array, twice the space required for the array is allocated, plus one more word as a buffer pointer. The memory required is:

*words of memory = (8 * num_elements) + 1*

This double buffered arrangement allows the ALG:ARRAY command to download all elements of the array into the "B" buffer while your algorithm is accessing values from the "A" buffer. Then an ALG:UPDATE command will cause the buffer pointer word to point to the newly loaded buffer between algorithm executions.

Bitfield Access

The HP E1422 implements bitfield syntax that allows you to manipulate individual bit values within a variable. This syntax is similar to what would be done in 'C', but doesn't require a structure declaration. Bitfield syntax is supported only for the lower 16 bits (bits 0-15) of simple (scalar) variables and channel identifiers.

Use

```
if(word_var.B0 || word_var.B3)   /* if either bit 0 or bit 3 true ... */
    word_var.B15 = 1;            /* set bit 15                        */
```

NOTES

1. You don't have to declare a bitfield structure in order to use it. In the Algorithm Language the bitfield structure is assumed to be applicable to any simple variable including channel identifiers.

2. Unlike 'C', the Algorithm Language allows you both bit access and "whole" access to the same variable. Example:

```
static float my_word_var;
my_word_var = 255     /* set bits 0 through 7    */
```

```
my_word_var.B3 = 0     /* clear bit 3                */
```

Declaration Initialization

You may only initialize simple variables (not array members) in the declaration statement:

```
static float my_var = 2;
```

NOTE! The initialization of the variable only occurs when the algorithm is first defined with the ALG:DEF command. The first time the algorithm is executed (module INITed and triggered), the value will be as initialized. But when the module is stopped (ABORT command), and then re-INITiated, the variable will not be re-initialized but will contain the value last assigned during program execution. In order to initialize variables each time the module is re-INITialized, see “Determining First Execution (First_loop)” on page 159.

Global Variables

To declare global variables you execute the SCPI command ALG:DEF 'GLOBALS',<*program_string*>. The <*program_string*> can contain simple variable and array variable declaration/initialization statements. The string must not contain any executable source code.

Language Syntax Summary

This section documents the HP E1422's Algorithm Language elements.

Identifier:

first character is A-Z, a-z, or "_", optionally followed by characters; A-Z, a-z, 0-9 or "_". Only the first 31 characters are significant. For example; a, abc, a1, a12, a_12, now_is_the_time, gain1

Decimal Constant:

first character is 0-9 or "."(decimal point). Remaining characters if present are 0-9, a "."(one only), a single "E"or"e", optional "+" or "-", 0-9. For example; 0.32, 2, 123, 123.456, 1.23456e-2, 12.34E3

NOTE Decimal constants without a decimal point character are treated by the translator as 32-bit integer values. See “Data Types” on page 179.

Hexadecimal Constant:

first characters are 0x or 0X. Remaining characters are 0-9 and A-F or a-f. No "." allowed.

Octal Constant:

first character is 0. Remaining characters are 0-7. If ".", "e", or "E" is found, the number is assumed to be a Decimal Constant as above.

Primary-expression:

constant
(*expression***)**
scalar-identifier
*scalar-identifier***.***bitnumber*
*array-identifier***[***expression***]**
abs(*expression***)**
max(*expression***,***expression***)**
min(*expression***,***expression***)**

Bit-number:

B*n* where *n*=0-9
B*nn* where *nn*=10-15

Unary-expression:

primary-expression
unary-operator unary-expression

Unary-operator:

+
-
!

Multiplicative-expression:

unary-expression
multiplicative-expression multiplicative-operator unary-expression

Multiplicative-operator:

*
/

Additive-expression:

multiplicative-expression
additive-expression additive-operator multiplicative-expression

Additive-operator:

+
-

Relational-expression:

additive-expression
relational-expression relational-operator additive-expression

Relational-operator:

<
>
<=
>=

Equality-expression:

relational-expression
equality-expression equality-operator relational-expression

Equality-operator:

==
!=

Logical-AND-expression:

equality-expression
logical-AND-expression **&&** *equality-expression*

Expression:

logical-AND-expression
expression **||** *logical-AND-expression*

Declarator:

identifier
identifier **[** *integer-constant-expression* **]**
NOTE: integer-constant expression in array identifier above must not exceed 1023

Init-declarator:

declarator
declarator = *constant-expression*
NOTES: 1. May not initialize array declarator.
2. Arrays limited to single dimension of 1024 maximum.

Init-declarator-list:

init-declarator
init-declarator-list **,** init-declarator

Declaration:

static float *init-declarator-list;*

Declarations:

declaration
declarations declaration

Intrinsic-statement:

interrupt ( )
writefifo (*expression)*
writecvt (*expression* **,** *constant-expression* **)**
writeboth(*expression* **,** *constant-expression* **)**
exit (*expression* **)**

Expression-statement:

scalar-identifier = expression ;
scalar-identifier **.** *bit-number = expression* **;**
array-identifier **[** *integer-constant expression]* = expression **;**
intrinsic-statement **;**

Selection-statement:

if (*expression)* *statement*
if (*expression)* *statement* **else** *statement*

Compound-statement:

{ *statement-list* **}**
{ }
NOTE: Variable declaration not allowed in compound statement

Statement:

expression-statement
compound-statement
selection-statement

Statement-list:

statement
statement-list statement

Algorithm-definition:

declarations statement-list
statement-list

Program Structure and Syntax

In this section you will learn the portion of the 'C' programming language that is directly applicable to the HP E1422' Algorithm Language. To do this we will compare the 'C' Algorithm Language elements with equivalent BASIC language elements.

Declaring Variables

In BASIC you usually use the DIM statement to name variables and allocate space in memory for them. In the Algorithm Language you specify the variable type and a list of variables:

BASIC	'C'
DIM a, var, array(3)	static float a, var, array[3];

Here we declared three variables. Two simple variables; **a**, and **var**, and a single dimensioned array; **array**.

Comments:

- Note that the 'C' language statement must be terminated with the semicolon "**;**".
- Although in the Algorithm Language all variables are of type float, you must explicitly declare them as such.
- All variables in your algorithm are **static**. This means that each time your algorithm is executed, the variables "remember" their values from the previous execution. The **static** modifier must appear in the declaration.
- Array variables must have a single dimension. The array dimension specifies the number of elements. The lower bound is always zero (0) in the Algorithm Language. Therefore the variable My_array from above has three elements; My_array [0] through My_array[2].

Assigning Values

BASIC and 'C' are the same here. In both languages you use the symbol "=" to assign a value to a simple variable or an element of an array. The value can come from a constant, another variable, or an expression. Examples:

```
a = 12.345;
a = My_var;
a = My_array[ 2 ];
a = (My_array[ 1 ] + 6.2) / My_var;
```

NOTE In BASIC the assignment symbol "=" is also used as the comparison operator "is equal to". For example; IF a=b THEN ... . As you will read a little further on, 'C' uses a different symbol for this comparison.

The Operations Symbols

Many of the operation symbols are the same, and are used the same way as those in BASIC. However there are differences, and they can cause programming errors until you get used to them.

The Arithmetic Operators

The arithmetic operators available to the HP E1422 are the same as those equivalents in BASIC:

+	(addition)	**-**	(subtraction)
*	(multiplication)	/	(division)

Unary Arithmetic Operator

Again same as BASIC:

-	(unary minus)	Examples:	a = b + (-c)
+	(unary plus)		a = c + (+b)

The Comparison Operators

Here there are some differences.

BASIC		**'C'**	**Notes**
=	(is equal to)	==	Different (hard to remember)
<> or #	(is not equal to)	!=	Different but obvious
>	(is greater than)	>	Same
<	(is less than)	>	Same
>=	(is greater than or equal to)	>=	Same
<=	(is less than or equal to)	<=	Same

A common 'C' programming error for BASIC programmers is to inadvertently use the assignment operator "=" instead of the comparison operator "==" in an **if** statement. Fortunately, the HP E1422 will flag this as a Syntax Error when the algorithm is loaded.

The Logical Operators

There are three operators. They are very different from those in BASIC.

<table>
<tr><th>BASIC</th><th>Examples</th><th>'C'</th><th>Examples</th></tr>
<tr><td>AND</td><td>IF A=B AND B=C</td><td>&&</td><td>if((a == b)&&(b == c))</td></tr>
<tr><td>OR</td><td>IF A=B OR A=C</td><td>| |</td><td>if((a == b) | | (a == c))</td></tr>
<tr><td>NOT</td><td>IF NOT B</td><td>!</td><td>if (! b)</td></tr>
</table>

Conditional Execution

The HP E1422 Algorithm Language provides the **if - else** construct for conditional execution. The following figure compares the elements of the 'C' **if - else** construct with the BASIC **if - then - else - end if** construct. The general form of the **if - else** construct is:

if(*expression*) *statement1* **else** *statement2*

where *statement1* is executed if *expression* evaluates to non-zero (true), and *statement2* is executed if *expression* evaluates to zero (false). *Statement1* and/or *statement2* can be compound statements. That is, multiple simple statements within curly braces. See Figure 5-5

BASIC Syntax	Comments	'C' Syntax
IF *boolean_expression* THEN statement	Simplest form (used often)	if(*boolean_expression) statement*;
IF *boolean_expression* THEN *statement* END IF	Two-line form (not recommended; use multiple line form instead)	if(*boolean_expression)* *statement*;
IF boolean_expression THEN *statement* *statement* *statement* *END IF*	Multiple line form (used often)	if(*boolean_expression*){ *statement;* *statement;* *statement;* *}*
IF *boolean_expression* THEN *statement* *statement* *ELSE* *statement* END IF	Multiple line form with else (used often)	*if(boolean_expression)* *{* *statement;* *statement;* *}* *else* *{* *statement*; }

Figure 5-5. The if Statement 'C' versus BASIC

Note that in BASIC the *boolean_expression* is delimited by the IF and the THEN keywords. In 'C' the parentheses delimit the expression. In 'C' , the ")" is the implied THEN. In BASIC the END IF keyword terminates a multi-line IF. In 'C', the **if** is terminated at the end of the following statement when no **else** clause is present, or at the end of the statement following the **else** clause. Figure 5-6 shows examples of these forms:

Note that in 'C' "else" is part of the closest previous "if"statement. So the example:

if(x) if(y) z = 1; else z = 2;

executes like:

```
if( x ){
    if( y ){
        z = 1;
    }
    else{
        z = 2;
    }
}
```

not like:

```
if( x ){
    if ( y ){
        z = 1;
    }
}
else{
    z = 2;
}
```

BASIC Syntax ← Examples → 'C' Syntax

BASIC Syntax:

```
IF A<=0 THEN C=ABS(A)

IF A<>0 THEN
  C=B/A
END IF

IF A<>B AND A<>C THEN
  A=A*B
  B=B+1
  C=0
END IF

IF A=5 OR B=-5 THEN
  C=ABS(C)
  C= 2/C
ELSE
  C= A*B
END IF
```

'C' Syntax:

```
if(a <= 0) c=abs(a);

if(a != 0)
  c = b / a;

if((a != b) && (a != c))
{
  a = a * b;
  b = b + 1;
  c = 0;
}

if((a == 5) || (b == -5))
{
  c = abs(c);
  c = 2 / c;
}
else
{
  c = a * b;
}
```

Figure 5-6. Examples of 'C' and BASIC if Statements

Comment Lines

Probably the most important element of programming is the comment. In older BASIC interpreters the comment line began with "REM" and ended at the end-of-line character(s) (probably carriage return then linefeed). Later BASICs allowed comments to also begin with various "shorthand" characters such as "!", or "'". In all cases a comment ended when the end-of-line is encountered. In 'C' and the Algorithm Language, comments begin with the the two characters "/*" and continue until the two characters "*/" are encountered. Examples:

```
/* this line is solely a comment line */
if ( a != b) c = d + 1;  /* comment within a code line */
/*  This comment is composed of more than one line.
    The comment can be any number of lines long and
    terminates when the following two characters appear
*/
```

About the only character combination that is not allowed within a comment is "*/", since this will terminate the comment.

Overall Program Structure

The preceding discussion showed the differences between individual statements in BASIC and 'C'. Here we will show how the HP E1422's Algorithm Language elements are arranged into a program.

Here is a simple example algorithm that shows most of the elements discussed so far.

```
/*      Example Algorithm to show language elements in the context of a complete
        custom algorithm.

        Program variables:

            user_flag        Set this value with the SCPI command ALG:SCALAR.
            user_value       Set this value with the SCPI command ALG:SCALAR.

        Program Function:

        Algorithm returns user_flag in CVT element 330 and another value in CVT element 331
        each time the algorithm is executed.
        When user_flag = 0, returns zero in CVT 331.
        When user_flag is positive, returns user_value * 2 in CVT 331
        When user_flag is negative, returns user_value / 2 in CVT 331 and in FIFO

        Use the SCPI command ALGorithm:SCALar followed by ALGorithm:UPDate to set
        user_flag and user_value.
*/
static float user_flag;            /* Declaration statements (end with  ; ) */
static float user_value;

        writecvt (user_flag,330);  /* Always write user_flag in CVT (statement ends with ; ) */

        if (user_flag )            /* if statement (note no ; ) */
        {                          /* brace opens compound statement */
            if (user_flag > 0) writecvt (user_value * 2,331);   /* one-line if statement (writecvt ends with ; ) */
            else                   /* else immediately follows complete if-statement construct */
            {                      /* open compound statement for else clause */
                writecvt (user_value / 2,331);  /* each simple statement ends in ; (even within compound ) */
                writefifo (user_value);  /* these two statements could combine with writeboth () */
            }                      /* close compound statement for else clause */
        }                          /* close compound statement for first if */
        else writecvt (0,331);/* else clause goes with first if statement. Note single line else */
```

Chapter 6
HP E1422 Command Reference

Using This Chapter

This chapter describes the **Standard Commands for Programmable Instruments** (SCPI) command set and the **IEEE-488.2 Common Commands** for the HP E1422.

Overall Command Index

SCPI Commands

Common Commands

Command Fundamentals

Commands are separated into two types: IEEE-488.2 Common Commands and SCPI Commands. The SCPI command set for the HP E1422 is 1990 compatible

Common Command Format

The IEEE-488.2 standard defines the Common commands that perform functions like reset, self-test, status byte query, etc. Common commands are four or five characters in length, always begin with the asterisk character (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of Common commands are:

```
*RST
*ESR 32
*STB?
```

SCPI Command Format

The SCPI commands perform functions like configuring channels, setting up the trigger system, and querying instrument states or retrieving data. A subsystem command structure is a hierarchical structure that usually consists of a top level (or root) command, one or more lower level commands, and their parameters. The following example shows part of a typical subsystem:

```
MEMory
    :VME
        :ADDRess  <A24_address>
        :ADDRess?
        :SIZE  <mem_size>
        :SIZE?
```

MEMory is the root command, :VME is the second level command, and :ADDRess, and SIZE are third level commands.

Command Separator

A colon (**:**) always separates one command from the next lower level command as shown below:

ROUTE:SEQUENCE:DEFINE?

Colons separate the root command from the second level command (ROUTE:SEQUENCE) and the second level from the third level (SEQUENCE:DEFINE?). If parameters are present, the first is separated from the command by a space character. Additional parameters are separated from each other by a commas.

Abbreviated Commands

The command syntax shows most commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, send the entire command. The instrument will accept either the abbreviated form or the entire command.

For example, if the command syntax shows SEQuence, then SEQ and SEQUENCE are both acceptable forms. Other forms of SEQuence, such as SEQUEN or SEQU will generate an error. You may use upper or lower case letters. Therefore, SEQUENCE, sequence, and SeQuEnCe are all acceptable.

Implied Commands

Implied commands are those which appear in square brackets (**[ ]**) in the command syntax. (Note that the brackets are not part of the command, and are not sent to the instrument.) Suppose you send a second level command but do not send the preceding implied command. In this case, the instrument assumes you intend to use the implied command and it responds as if you had sent it. Examine the INITiate subsystem shown below:

```
INITiate
    [:IMMediate]
```

The second level command :IMMediate is an implied command. To set the instrument's trigger system to INIT:IMM, you can send either of the following command statements:

INIT:IMM *or* INIT

Variable Command Syntax

Some commands will have what appears to be a variable syntax. As an example:
OUTPut:TTLTrg<n>:STATe ON

In these commands, the "<n>" is replaced by a number. No space is left between the command and the number because the number is not a parameter. The number is part of the command syntax. The purpose of this notation is to save a great deal of space in the Command Reference. In the case of …TTLTrg<n>…, n can be from 0 through 7. An example command statement:

OUTPUT:TTLTRG2:STATE ON

Parameters

Parameter Types. The following section contains explanations and examples of parameter types you will see later in this chapter.

Parameter Types	Explanations and Examples
Numeric	Accepts all commonly used decimal representations of numbers including optional signs, decimal points, and scientific notation: 123, 123E2, -123, -1.23E2, .123, 1.23E-2, 1.23000E-01. Special cases include MIN, MAX, and INFinity. A parameter that represents units may also include a units suffix. These are: Volts; V, mv=10^{-3}, uv=10^{-6} Ohms; ohm, kohm=10^{3}, mohm=10^{6} Seconds; s, msec=10^{-3}, usec=10^{-6} Hertz; hz, khz=10^{3}, mhz=10^{6}, ghz=10^{9} The Comments section within the Command Reference will state whether a numeric parameter can also be specified in hex, octal, and/or binary; #H7B, #Q173, #B1111011

Boolean Represents a single binary condition that is either true or false:

ON, OFF, 1, 0.

Discrete Selects from a finite number of values. These parameters use mnemonics to represent each valid setting.

An example is the TRIGger:SOURce <*source*> command where <*source*> can be;

BUS, EXT, HOLD, IMM, SCP,TIMer, or TTLTrg<n>.

Channel List (Standard Form) The general form of a single channel specification is:

cnn

where *c* represents the card number and *nn* represents the channel number.

On-board Channels: Since the HP E1422A has an on-board 64 channel multiplexer, the card number will always be 1 and the channel number can range from 00 to 63. Some example channel specifications:
channel 0=100, channel 5=105, channel 54=154

Remote Channels: The HP E1422A uses the HP E1539A SCP to support Remote Signal Conditioning Units like the HP E1529A. Through these SCPs, the HP E1422A can address up-to 512 remote channels. The HP E1539A SCP has 2 measurement channels, that can each be externally multiplexed to 32 channels. The remote channel syntax is similar to the on-board syntax but with the addition of 2 more channel reference digits:

cnnee

where *c* is the card number and again is always 1, nn references only the first or second channel at any SCP position in the HP E1422A. So, nn can be any of 00, 01, 08, 09, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, 57. The additional digits *ee* reference one of 32 channels (00 through 31) on the Remote Signal Conditioning unit connected to the on-board channel *nn*. An example of an RSC Unit is the HP E1529A Remote Strain Bridge Conditioning Unit.

Some example remote channel specifications:
10000 = RSC channel 00 connected to HP E1422A channel 00
10100 = RSC channel 00 connected to HP E1422A channel 01
10122 = RSC channel 22 connected to HP E1422A channel 01
12522 = RSC channel 22 connected to HP E1422A channel 25

Specifying a range of Channels: The General form of a channel range specification is:

1*nn*[*ee*]:1*nn*[*ee*] (colon separator)

[*ee*] means optional remote channels. The second channel must

be greater than the first.
Examples:
On-board channels 0 through 15=100:115
Remote channels 0000 through 0131=10000:10131

By using commas to separate them, individual and range specifications can be combined into a single channel list:
0, 5, 8 through 3331, and 45=(@100,105,108:13331,145)
Notice that the range specified (108:13331) included a mix of on-board channels as well as remote channels. The HP E1422A will correctly address all channels in the range according to the type of SCP installed at the channel position (remote for HP E1539A SCPs and on-board for all other SCP models.

Note When a channel range includes both on-board and remote channel references, the command that specifies this range must be applicable to the function of the SCPs installed at those addresses, or a 3007 "Invalid signal conditioning plug-on"error will be generated.

Channel List (Relative Form)

The standard SCPI Relative Channel specification syntax is:
(@*cc*(*nn*,*nn*,*nn*:*nn* ...))
where *cc* = card number, and *nn* = channel number. Notice that with this form the card number digit moves from in front of each channel number, to outside of the inner parentheses.
Example: 0, 5, 6 through 32, and 45 = (@1(0,5,6:32,45))

The Relative form has special meaning when used in the HP E1422's **ROUTe:SEQuence:DEFine (@<ch_list>)** command. For the HP E1422A, the syntax changes to:
(@*d*(1*nn*,1*nnee*,1*nn*:1*nn*,1*nn*:1*nnee*,1*nnee*:1*nnee*,1*nnee*:1*nn* ...)
Notice that cc (standard form card number) has changed to *d* where *d* is now the "Data Destination" digit. The value of *d* controls the destination of the values read from the specified channels in the following manner:

Data Destination	Effect on Reading
1	Reading sent to Current Value Table (CVT)
2	Reading sent to FIFO Buffer
3	Reading sent to CVT and FIFO (the default)
0	Reading not recorded (neither CVT or FIFO)

Getting back to the relative channel syntax above, notice also that all channels start with "1".

Notes

1. Because the "card number" digit has been changed to mean Data Destination, the Relative Channel form is only allowed in the ROUTe:SEQuence:DEFine command. Usage in other commands will generate an error message.
2. Note that for both forms, a channel list is always contained within

"(@" and ")". The Command Reference always shows the "(@" and ")" punctuation: (@<*ch_list*>)

3. For the ROUT:SEQ:DEF command, the HP E1422A has to transfer remote channels lists to the RSC units they reference. This transfer will be much more efficient if channels for particular RSCs are grouped toghether in the list. (@10025,10031,10120,10820,10810,10903) is better than (@10810,10025,10903,10031,10820,10903)

Arbitrary Block Program and Response Data

This parameter or data type is used to transfer a block of data in the form of bytes. The block of data bytes is preceded by a preamble which indicates either 1) the number of data bytes which follow (definite length), or 2) that the following data block will be terminated upon receipt of a New Line message, and for HP-IB operation, with the EOI signal true (indefinite length).

The syntax for this parameter is:

Definite Length; #<non-zero digit><digit(s)><data byte(s)>

Where the value of <non-zero digit> is 1-9 and represents the number of <digit(s)>. The value of <digit(s)> taken as a decimal integer indicates the number of <data byte(s)> in the block.

Example of sending or receiving 1024 data bytes:
#41024<byte><byte1><byte2><byte3><byte4>…
…<byte1021><byte1022><byte1023><byte1024>

OR

Indefinite Length; #0<data byte(s)><NL^END>

Example of sending or receiving 4 data bytes:
#0<byte><byte><byte><byte><NL^END>

Optional Parameters

Parameters shown within square brackets (**[ ]**) are optional parameters. (Note that the brackets are not part of the command, and should not be sent to the instrument.) If you do not specify a value for an optional parameter, the instrument chooses a default value. For example, consider the FORMAT:DATA <*type*>[,<*length*>] command. If you send the command without specifying <*length*>, a default value for <*length*> will be selected depending on the <*type*> of format you specify. For example:

FORMAT:DATA ASC will set [,<*length*>] to the default for ASC of 7
FORMAT:DATA REAL will set [,<*length*>] to the default for REAL of 32
FORMAT:DATA REAL, 64 will set [,<*length*>] to 64

Be sure to place a space between the command and the first parameter.

Linking Commands

Linking commands is used when you want to send more than one complete command in a single command statement.

Linking IEEE-488.2 Common Commands with SCPI Commands. Use a semicolon between the commands. For example:

*RST;OUTP:TTLT3 ON *or* TRIG:SOUR IMM;*TRG

Linking Multiple complete SCPI Commands. Use both a semicolon and a colon between the commands. For example:

OUTP:TTLT2 ON;:TRIG:SOUR EXT

The semicolon as well as separating commands tells the SCPI parser to expect the command keyword following the semicolon to be at the same hierarchical level (and part of the same command branch) as the keyword preceding the semicolon. The colon immediately following the semicolon tells the SCPI parser to reset the expected hierarchical level to Root.

Linking a complete SCPI Command with other keywords from the same branch and level. Separate the first complete SCPI command from next partial command with the semicolon only. For example take the following portion of the [SENSE] subsystem command tree (the FUNCtion branch):

```
[SENSe:]
    FUNCtion
        :RESistance  <range>,(@<ch_list>)
        :TEMPerature  <sensor>[,<range>,](@<ch_list>)
        :VOLTage[:DC]  [<range>,](@<ch_list>)
```

Rather than send a complete SCPI command to set each function, you could send:

FUNC:RES 10000,(@100:107);TEMP RTD, 92,(@108:115);VOLT (@116,123)

This sets the first 8 channels to measure resistance, the next 8 channels to measure temperature, and the next 8 channels to measure voltage.

Note The command keywords following the semicolon must be from the same command branch and level as the complete command preceding the semicolon or a -113,"Undefined header" error will be generated.

Data Types

The following table shows the allowable type and sizes of parameter data sent to the module and query data returned by the module. The parameter and returned value type is necessary for programming and is documented in each command in this chapter

.

Data Types	Description
int16	Signed 16-bit integer number.
int32	Signed 32-bit integer number.
uint16	Unsigned 16-bit integer number.
uint32	Unsigned 32-bit integer number.
float32	32-bit floating point number.
float64	64-bit floating point number.
string	String of characters (null terminated)

SCPI Command Reference

The following section describes the SCPI commands for the HP E1422. Commands are listed alphabetically by subsystem and also within each subsystem. A command guide is printed in the top margin of each page. The guide indicates the current subsystem on that page.

ABORt

The ABORt subsystem is a part of the HP E1422's trigger system. ABORt resets the trigger system from its Wait For Trigger state to its Trigger Idle state.

Subsystem Syntax

ABORt

Caution **ABORT stops execution of a running algorithm. The control output is left at the last value set by the algorithm. Depending on the process, this uncontrolled situation could even be dangerous. Make certain that you have put your process into a safe state before you halt execution of a controlling algorithm.**

Comments

- ABORt does not affect any other settings of the trigger system. When the INITiate command is sent, the trigger system will respond just as it did before the ABORt command was sent.
- **Related Commands:** INITiate[:IMMediate], TRIGger…
- ***RST Condition:** TRIG:SOUR HOLD
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

ABORT	*If INITed, goes to Trigger Idle state. If scanning and/or running algorithms, stops and goes to Trigger Idle State.*

ALGorithm

The ALGorithm command subsystem provides:

- Definition of user defined control algorithms
- Communication with algorithm array and scalar variables
- Controls to enable or disable individual loop algorithms
- Control of ratio of number of scan triggers per algorithm execution
- Control of loop algorithm execution speed
- Easy definition of algorithm data conversion functions

Subsystem Syntax

```
ALGorithm
    [:EXPLicit]
        :ARRay  '<alg_name>','<array_name>',<array_block>
        :ARRay?  '<alg_name>','<array_name>'
        :DEFine  '<alg_name>'[,<swap_size>],<program_block>
        :SCALar  '<alg_name>','<var_name>',<value>
        :SCALar?  '<alg_name>','<var_name>'
        :SCAN:RATio  '<alg_name>',<value>
        :SCAN:RATio?  '<alg_name>'
        :SIZe?  '<alg_name>'
        [:STATe]  '<alg_name>',ON | OFF
        [:STATe]?  '<alg_name>'
        :TIME?  '<alg_name>'

:FUNCtion:DEFine  '<function_name>',<range>,<offset>,<block_data>
    :OUTPut:DELay  <usec> | AUTO
    :OUTPut:DELay?
    :UPDate
        [:IMMediate]
        :CHANnel  <channel_item>
        :WINDow  <num_updates>
        :WINDow?
```

ALGorithm[:EXPLicit]:ARRay

ALGorithm[:EXPLicit]:ARRay '<*alg_name*>','<*array_name*>',<*array_block*> places values of <*array_name*> for algorithm <*alg_name*> into the Update Queue. This update is then pending until ALG:UPD is sent or an update event (as set by ALG:UPD:CHANNEL) occurs.

Note ALG:ARRAY places a variable update request in the Update Queue. You can not place more update requests in the Update Queue than are allowed by the current setting of ALG:UPD:WINDOW or a "Too many updates -- send ALG:UPDATE command' error message will be generated.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32 | GLOBALS	none
array_name	string	valid 'C' variable name	none
array_block	block data	block of IEEE-754 64-bit floating point numbers	none

Comments

- To send values to a Global array, set the <*alg_name*> parameter to "GLOBALS". To define a global array see the ALGorithm:DEFine command.
- An error is generated if <*alg_name*> or <*array_name*> is not defined.
- When an array is defined (in an algorithm or in 'GLOBALS'), the HP E1422 allocates twice the memory required to store the array. When you send the ALG:ARRAY command, the new values for the array are loaded into the second space for this array. When you send the ALG:UPDATE, or ALG:UPDATE:CHANNEL commands, the HP E1422 switches a pointer to the space containing the new array values. This is how even large arrays can be "updated" as if they were a single update request. If the array is again updated, the new values are loaded into the original space and the pointer is again switched.
- When this command is sent textually to an HP E1406A command module, the Definit Length Arbitrary Block <*array_block*> parameter must always use "Big Endian" (Motorola) byte ordering for the packed 64-bit float values.
- <*progname*> is not case sensitive. However, <*array_name*> is case sensitive.
- **Related Commands:** ALG:DEFINE, ALG:ARRAY?
- ***RST Condition:** No algorithms or variables are defined.
- **Use VXIplug&play function:** hpe1422_algArray(...)

Usage

send array values to my_array in ALG4
ALG:ARR 'ALG4','my_array',<*block_array_data*>
send array values to the global array glob_array
ALG:ARR 'GLOBALS','glob_array',<*block_array_data*>
ALG:UPD *force update of variables*

ALGorithm[:EXPLicit]:ARRay?

ALGorithm[:EXPLicit]:ARRay? '<*alg_name*>','<*array_name*>' returns the contents of <*array_name*> from algorithm <*alg_name*>. ALG:ARR? can return contents of global arrays when <*alg_name*> specifies 'GLOBALS'.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32 | GLOBALS	none
array_name	string	valid 'C' variable name	none

Comments

- An error is generated if <*alg_name*> or <*array_name*> is not defined.
- When this command is sent to an HP E1406A command module, the Definite Length Abitrary Block response data will always use "Big Endian" (Motorola) byte ordering for the packed 64-bit float values.
- **Returned Value:** Definite length block data of IEEE-754 64-bit float
- **Send with VXIplug&play Function:** hpe1422_cmdReal64Arr_Q(...)

ALGorithm[:EXPLicit]:DEFine

ALGorithm[:EXPLicit]:DEFine '<*alg_name*>',[<*swap_size*>,] '<*source_code*>' is used to define control algorithms, and global variables.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32 | GLOBALS	none
swap_size	numeric (uint16)	0 - Max Available Algorithm Memory	words
source_code	string or block data see Comments	PIDA... | PIDB... | algorithm source	none

Comments

- **Use VXIplug&play function:** hpe1422_downloadAlg(...). This function loads an algorithm from a file. The VXIplug&play Soft Front Panel program allows you to create and test algorithms on-line, then store them to files.
- The <*alg_name*> must be one of ALG1, ALG2, ALG3 etc. through ALG32 or GLOBALS. The parameter is not case sensitive. 'ALG1' and 'alg1' are equivalent as are 'GLOBALS' and 'globals'.
- The <*swap_size*> parameter is optional. Include this parameter with the first definition of <*alg_name*> when you will want to change <*alg_name*> later while it is running. The value can range up to about 23Kwords (ALG:DEF will then allocate 46K words as it creates two spaces for this algorithm).

  -- If included, <*swap_size*> specifies the number of words of memory to allocate for the algorithm specified by <*alg_name*>. The HP E1422 will then allocate this much memory again, as an update buffer for this algorithm. Note that this doubles the amount of memory space requested. Think of this as "space1" and "space2" for algorithm <*alg_name*>. When you later send a replacement algorithm (must be sent <u>without</u> the <*swap_size*> parameter), it will be placed in "space2". You must send an ALG:UPDATE command for execution to switch from the original, to the replacement algorithm. If you again change the algorithm for <*alg_name*>, it will be executed from "space1" and so on. Note that <*swap_size*> must be large enough to contain the original executable code derived from <*source_code*> and any subsequent replacement for it or an error 3085 "Algorithm too big" will be generated.

-- If <*swap_size*> is not included, the HP E1422 will allocated just enough memory for algorithm <*alg_name*>. Since there is no swapping buffer allocated, this algorithm cannot be changed until a *RST command is sent to clear all algorithms. See "When Accepted and Usage".

- The <*source_code*> parameter contents can be:

-- When <*alg_name*> is 'ALG1' through 'ALG32':

a. 'PIDA(<*inp_channel*>,<*outp_channel*>)', or
'PIDB(<*inp_channel*>,<*outp_channel*>,<*alarm_channel*>)'
<   *_channel*> parameters can specify actual input and output channels or they can specify global variables. This can be useful for inter-algorithm communication. Any global variable name used in this manner must have already been defined before this algorithm.

```
ALG:DEF  'ALG3','PIDB(I100,O124,O132.B2)'
```

b. Algorithm Language source code representing a custom algorithm.

```
ALG:DEF  'ALG5','if( First_loop ) O116=0; O116=O116+0.01;,
```

-- When <*alg_name*> is 'GLOBALS', Algorithm Language variable declarations. A variable name must not be the same as an already define user function.

```
ALG:DEF  'GLOBALS','static float my_glob_scalar, my_glob_array[24];'
```

The Algorithm Language source code is translated by the HP E1422's driver into an executable form and sent to the module. For 'PIDA', and 'PIDB' the driver sends the stored executable form of these PID algorithms.

- The <*source_code*> parameter can be one of three different SCPI types:

-- Quoted String: For short segments (single lines) of code, enclose the code string within single (apostrophes), or double quotes. Because of string length limitations within SCPI and some programming platforms, we recommend that the quoted string length not exceed a single program line. Examples:

ALG:DEF 'ALG1','O108=I100;' or ALG:DEF 'ALG3','PIDA(I100,O124)'

Definite Length Block Program Data: For longer code segments (like complete custom algorithms) this parameter works well because it specifies the exact length of the data block that will be transferred. The syntax for this parameter type is:

<non-zero digit><digit(s)><data byte(s)> preceded by #:

#<non-zero digit><digit(s)><data byte(s)>

Where the value of <non-zero digit> is 1-9 and represents the number of <digit(s)>. The value of <digit(s)> taken as a decimal integer indicates the number of <data byte(s)> in the block. Example from "Quoted String" above:

ALG:DEF 'ALG1',#211O108=I100;∅ (where "∅" is a null byte)

Note For Block Program Data, the Algorithm Parser requires that the *source_code* data end with a null (0) byte. You must append the null byte to the end of the block's <data byte(s)>, and account for it in the byte count <digit(s)> from above. If the null byte is not included, or <digit(s)> doesn't include it, the error "Algorithm Block must contain termination '\0'" will be generated.

Indefinite Length Block Program Data: This form terminates the data transfer when it received an End Identifier with the last data byte. Use this form only when you are sure your controller platform will include the End Identifier. If it is not included, the ALG:DEF command will "swallow" whatever data follows the algorithm code. The syntax for this parameter type is:

#0<data byte(s)><null byte with End Identifier>

Example from "Quoted String" above:

ALG:DEF 'ALG1',#0O108=I100;∅ (where "∅" is a null byte)

Note For Block Program Data, the Algorithm Parser requires that the *source_code* data end with a null (0) byte. You must append the null byte to the end of the block's <data byte(s)>. The null byte is sent with the End Identifier. If the null byte is not included, the error "Algorithm Block must contain termination '\0'" will be generated.

When accepted and Usage

4. If <*alg_name*> is not enabled to swap (not originally defined with the <*swap_size*> parameter included) then both of the following conditions must be true:

   a. Module is in Trigger Idle State (after *RST, or ABORT, and before INIT).

```
OK
*RST
ALG:DEF 'GLOBALS','static float My_global;'
ALG:DEF 'ALG2','PIDA(I100,O108)'
ALG:DEF 'ALG3','My_global = My_global + 1;'

Error
INIT
ALG:DEF 'ALG5','PIDB(I101,O109,O124.B0)'
```

"Can't define new algorithm while running"

b. The <*alg_name*> has not already been defined since a *RST command. Here <*alg_name*> specifies either an algorithm name or 'GLOBALS'.

```
OK
*RST
ALG:DEF 'GLOBALS','static float My_global;'

Error
*RST
ALG:DEF 'GLOBALS','static float My_global;'
"No error"
ALG:DEF 'GLOBALS','static float A_different_global'
"Algorithm already defined"                    Because 'GLOBALS' already defined

Error
*RST
ALG:DEF 'ALG3','PIDA(I100,O108)'
"No error"
ALG:DEF 'ALG3','PIDB(I100,O108,O124.B0)'
"Algorithm already defined"                    Because 'ALG3' already defined
```

5. If <*alg_name*> has been enabled to swap (originally defined with the <*swap_size*> parameter included) then the <*alg_name*> can be re-defined (do not include <*swap_size*> now) either while the module is in the Trigger Idle State, or while in Waiting For Trigger State (INITed). Here <*alg_name*> is an algorithm name only, <u>not</u> 'GLOBALS'.

```
OK
*RST
ALG:DEF 'ALG3',200,'if(O108<15.0) O108=O108 + 0.1; else O108 = -15.0;'
INIT                                           starts algorithm
ALG:DEF 'ALG3','if(O108<12.0) O108=O108 + 0.2; else O108 = -12.0;'
ALG:UPDATE                                     Required to cause new code to run
"No error"

Error
*RST
ALG:DEF 'ALG3',200,'if(O108<15.0) O108=O108 + 0.1; else O108 = -15.0;'
INIT                                           starts algorithm
ALG:DEF 'ALG3',200,'if(O108<12.0) O108=O108 + 0.2; else O108 = -12.0;'
"Algorithm swapping already enabled; Can't change size"
Because <swap_size> included at re-definition
```

Notes

1. Channels referenced by algorithms when they are defined, are only placed in the channel list before INIT. The list cannot be changed after INIT. If you re-define an algorithm (by swapping) after INIT, and it references channels not already in the channel list, it will not be able to access the newly referenced channels. No error message will be generated. To make sure all

required channels will be included in the channel list, define <*alg_name*> and re-define all algorithms that will replace <*alg_name*> by swapping them before you send INIT. This insures that all channels referenced in these algorithms will be available after INIT.

2. If you re-define an algorithm (by swapping) after INIT, and it declares an existing variable, the declaration-initialization statement (e.g. `static float my_var = 3.5`) will not change the current value of that variable.
3. The driver only calculates overall execution time for algorithms defined before INIT. This calculation is used to set the default output delay (same as executing ALG:OUTP:DELAY AUTO). If an algorithm is swapped after INIT that take longer to execute than the original, the output delay will behave as if set by ALG:OUTP:DEL 0, rather than AUTO (see ALG:OUTP:DEL command). Use the same procedure from note 1 to make sure the longest algorithm execution time is used to set ALG:OUTP:DEL AUTO before INIT.

ALGorithm[:EXPLicit]:SCALar

ALGorithm[:EXPLicit]:SCALar '<*alg_name*>','<*var_name*>',<*value*> sets the value of the scalar variable <*var_name*> for algorithm <*alg_name*> into the Update Queue. This update is then pending until ALG:UPD is sent or an update event (as set by ALG:UPD:CHANNEL) occurs.

Note ALG:SCALAR places a variable update request in the Update Queue. You can not place more update requests in the Update Queue than are allowed by the current setting of ALG:UPD:WINDOW or a "Too many updates -- send ALG:UPDATE command" error message will be generated.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32 or GLOBALS	none
var_name	string	valid 'C' variable name	none
value	numeric (float32)	IEEE-754 32-bit floating point number	none

Comments

- To send values to a global scalar variable, set the <*alg_name*> parameter to 'GLOBALS'. To define a scalar global variable see the ALGorithm:DEFine command.
- An error is generated if <*alg_name*> or <*var_name*> is not defined.
- **Related Commands:** ALG:DEFINE, ALG:SCAL?, ALG:UPDATE
- ***RST Condition:** No algorithms or variables are defined.

- **Use VXIplug&play function:** hpe1422_algExpScal(...)

Usage

ALG:SCAL 'ALG1','my_var',1.2345	*1.2345 to variable my_var in ALG1*
ALG:SCAL 'ALG1','another',5.4321	*5.4321 to variable another also in ALG1*
ALG:SCAL 'ALG3','my_global_var',1.001	*1.001 to global variable*
ALG:UPD	*update variables from update queue*

ALGorithm[:EXPLicit]:SCALar?

ALGorithm[:EXPLicit]:SCALar? '<*alg_name*>','<*var_name*>' returns the value of the scalar variable <*var_name*> in algorithm <*alg_name*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32	none
var_name	string	valid 'C' variable name	none

Comments

- An error is generated if <*alg_name*> or <*var_name*> is not defined.
- **Returned Value:** numeric value. The type is **float32**.
- **Use VXIplug&play function:** hpe1422_algExpScal_Q(...)

ALGorithm[:EXPLicit]:SCAN:RATio

ALGorithm[:EXPLicit]:SCAN:RATio '<*alg_name*>',<*num_trigs*> specifies the number of scan triggers that must occur for each execution of algorithm <*alg_name*>. This allows you to execute the specified algorithm less often than other algorithms. This can be useful for algorithm tuning.

Notes

1. The command ALG:SCAN:RATio <*alg_name*>,<*num_trigs*> does not take effect until an ALG:UPDATE, or ALG:UPD:CHAN command is received. This allows you to send multiple ALG:SCAN:RATIO commands and then synchronize their effect with ALG:UPDATE.
2. ALG:SCAN:RATio places a variable update request in the Update Queue. You can not place more update requests in the Update Queue than are allowed by the current setting of ALG:UPD:WINDOW or a "Too many updates -- send ALG:UPDATE command" error message will be generated.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32	none
num_trigs	numeric (int16)	1 to 32,767	none

Comments Specifying a value of 1 (the default) causes the named algorithm to be executed each time a trigger is received. Specifying a value of n will cause the algorithm to be executed once every n triggers. All enabled algorithms execute on the first trigger after INIT.

- The algorithm specified by <*alg_name*> may or may not be currently defined. The specified setting will be used when the algorithm is defined.
- **Related Commands:** ALG:UPDATE, ALG:SCAN:RATIO?
- **When Accepted: Both before and after INIT**. Also accepted before and after the algorithm referenced is defined.
- ***RST Condition:** ALG:SCAN:RATIO = 1 for all algorithms
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage ALG:SCAN:RATIO ’ALG4’,16 *ALG4 executes once every 16 triggers.*

ALGorithm[:EXPLicit]:SCAN:RATio?

ALGorithm[:EXPLicit]:SCAN:RATio? ’<*alg_name*>’ returns the number of triggers that must occur for each execution of <*alg_name*>.

Comments

- Since ALG:SCAN:RATIO is valid for an undefined algorithm, ALG:SCAN:RATIO? will return the current ratio setting for <*alg_name*> even if it is not currently defined.
- **Returned Value:** numeric, 1 to 32,768. The type is **int16**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

ALGorithm[:EXPLicit]:SIZE?

ALGorithm[:EXPLicit]:SIZE? ’<*alg_name*>’ returns the number of words of memory allocated for algorithm <*alg_name*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32	none

Comments

- Since the returned value is the memory allocated to the algorithm, it will only equal the actual size of the algorithm if it was defined by ALG:DEF without its [<*swap_size*>] parameter. If enabled for swapping (if <*swap_size*> included at original definition), the returned value will be equal to (<*swap_size*>)*2.

Note If <*alg_name*> specifies an undefined algorithm, ALG:SIZE? returns 0. This can be used to determine whether algorithm <*alg_name*> is defined.

- **Returned Value:** numeric value up to the maximum available algorithm memory (this approximately 40K words). The type is **int32**.
- ***RST Condition:** returned value is 0.
- **Send with VXIplug&play Function:** hpe1422_cmdInt32_Q(...)

ALGorithm[:EXPLicit][:STATe]

ALGorithm[:EXPLicit][:STATe] '<*alg_name*>',<*enable*> specifies that algorithm <*alg_name*> , when defined, should be executed (ON), or not executed (OFF) during run-time.

Notes

1. The command ALG:STATE <*alg_name*>, ON | OFF does not take effect until an ALG:UPDATE, or ALG:UPD:CHAN command is received. This allows you to send multiple ALG:STATE commands and then synchronize their effect.
2. ALG:STATE places a variable update request in the Update Queue. You can not place more update requests in the Update Queue than are allowed by the current setting of ALG:UPD:WINDOW or a "Too many updates -- send ALG:UPDATE command" error message will be generated.

Caution **When ALG:STATE OFF disables an algorithm, its control output is left at the last value set by the algorithm. Depending on the process, this uncontrolled situation could even be dangerous. Make certain that you have put your process into a safe state before you halt execution of a controlling algorithm.**

The HP E1535 Watchdog Timer SCP was specifically developed to automatically signal that an algorithm has stopped controlling a process. Use of the Watchdog Timer is recommended for critical processes.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32	none
enable	boolean (uint16)	0 | 1 | ON | OFF	none

Comments

- The algorithm specified by <*alg_name*> may or may not be currently defined. The setting specified will be used when the algorithm is defined.
- ***RST Condition:** ALG:STATE ON
- **When Accepted: Both before and after INIT**. Also accepted before and after the algorithm referenced is defined.
- **Related Commands:** ALG:UPDATE, ALG:STATE?, ALG:DEFINE
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

ALG:STATE 'ALG2',OFF *disable ALG2*

ALGorithm[:EXPLicit][:STATe]?

ALGorithm[:EXPLicit][:STATe]? '<*alg_name*>' returns the state (enabled or disabled) of algorithm <*alg_name*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32	none

Comments

- Since ALG:STATE is valid for an undefined algorithm, ALG:STATE? will return the current state for <*alg_name*> even if it is not currently defined.
- **Returned Value:** Numeric, 0 or 1. Type is **uint16**.
- ***RST Condition:** ALG:STATE 1
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

ALGorithm[:EXPLicit]:TIME?

ALGorithm[:EXPLicit]:TIME? '<*alg_name*>' computes and returns a worst-case execution time estimate in seconds.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
alg_name	string	ALG1 - ALG32 or MAIN	none

Comments

- When <*alg_name*> is ALG1 through ALG32, ALG:TIME? returns only the time required to execute the algorithm's code.
- When <*alg_name*> is 'MAIN', ALG:TIME? returns the worst-case execution time for an entire measurement & control cycle (sum of MAIN, all enabled algorithms, analog and digital inputs, and control outputs).
- If triggered more rapidly than the value returned by ALG:TIME? 'MAIN', the HP E1422 will generate a "Trigger too fast" error.

Note If <*alg_name*> specifies an undefined algorithm, ALG:TIME? returns 0. This can be used to determine whether algorithm <*alg_name*> is defined.

This command forces algorithms to run internally. If an algorithm contains a run-time error, no data can be returned and the command will not complete (will "hang").

- **When Accepted: Before INIT only**.
- **Returned Value:** numeric value. The type is **float32**
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

ALGorithm:FUNCtion:DEFine

ALGorithm:FUNCtion:DEFine '<*function_name*>',<*range*>,<*offset*>, <*func_data*> defines a custom function that can be called from within a custom algorithm. See “Generating User Defined Functions” on page 419 for full information.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
function_name	string	valid 'C' identifier (if not already defined in 'GLOBALS')	none
range	numeric (float32)	see comments	none
offset	numeric (float32)	see comments	none
func_data	512 element array of uint16	see comments	none

Comments

- By providing this custom function capability, the HP E1422's algorithm language can be kept simple in terms of mathematical capability. This increases speed. Rather than having to calculate high-order polynomial approximations of non-linear functions, this custom function scheme loads a pre-computed look-up table of values into memory. This method allows computing virtually any transcendental or non-linear function in only 17µseconds. Resolution is 16 bits.

- <*function_name*> is a global identifier and cannot be the same as a previously define global variable. A user function is globally available to all defined algorithms.

- You generate values for <*range*>, <*offset*>, and <*func_data*> with a program supplied with your HP E1422. It is provided in C-SCPI, and HP Basic forms. For full information see Appendix E "Generating User Defined Functions" on page 419

- <*range*>, and <*offset*> define the allowable input values to the function (domain). If values input to the function are equal to or outside of (±<*range*>+<*offset*>), the function may return ±INF in IEEE-754 format. For example; <*range*> = 8 (-8 to 8), <*offset*> = 12. The allowable input values must be greater than 4 and less than 20.

- <*func_data*> is a 512 element array of type uint16.

- The algorithm syntax for calling is: <*function_name*> **(** <*expression*> **)**. for example:

```
O116 = squareroot( 2 * Input_val );
```

- Functions must be defined before defining algorithms that reference them.

- **When Accepted: Before INIT only**.

Usage

ALG:FUNC:DEF 'F1',8,12,<*block_data*> — *send range, offset and table values for function F1*

- **Use VXIplug&play function:** hpe1422_sendBlockUInt16(...)

ALGorithm:OUTPut:DELay

ALGorithm:OUTPut:DELay <*delay*> sets the delay from Scan Trigger to start of output phase.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
delay	numeric (float32)	0 - .081 | AUTO (2.5µs resolution)	seconds

Comments

- The algorithm output statements (e.g. O115 = Out_val) DO NOT program outputs when they are executed. Instead, these statements write to an intermediate Output Channel Buffer which is read and used for output AFTER all algorithms have executed AND the algorithm output delay has expired (see Figure 6-1). Also note that not all outputs will occur at the same time but will take approximately 10usec per channel to write.

- When <*delay*> is 0, the Output phase begins immediately after the Calculate phase. This provides the fastest possible execution speed while potentially introducing variations in the time between trigger and beginning of the Output phase. The variation can be caused by conditional execution constructs in algorithms, or other execution time variations.

- If you set <*delay*> to less time than is required for the Input + Update + Calculate ALG:OUTP:DELAY? will report the time you set, but the effect will revert to the same that is set by ALG:OUTP:DELAY 0 (Output begins immediately after Calculate).

- When <*delay*> is AUTO, the delay is set to the worst-case time required to execute phases 1 through 3. This provides the fastest execution speed while maintaining a fixed time between trigger and the OUTPUT phase.

- When you want to set the time from trigger to the beginning of OUTPUT, use the following procedure. After defining all of your algorithms, execute:

ALG:OUTP:DEL AUTO	*sets minimum stable delay*
ALG:OUTP:DEL?	*returns this minimum delay*
ALG:OUTP:DEL <*minimum+additional*>	*additional = **desired** - minimum*

Note that the delay value returned by ALG:OUTP:DEL? is valid only until another algorithm is loaded. After that, you would have to re-issue the ALG:OUTP:DEL AUTO and ALG:OUTP:DEL? commands to determine the new delay that includes the added algorithm.

- **When Accepted:** Before INIT only.

- ***RST Condition:** ALG:OUTP:DELAY AUTO

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

ALGorithm:OUTPut:DELay?

ALGorithm:OUTPut:DELay? returns the delay setting from ALG:OUTP:DEL.

Comments

- The value returned will be either the value set by ALG:OUTP:DEL <*delay*>, or the value determined by ALG:OUTP:DEL AUTO.

- **When Accepted:** Before INIT only.

- ***RST Condition:** ALG:OUTP:DEL AUTO, returns delay setting determined by AUTO mode.

- **Returned Value:** number of seconds of delay. The type is **float32**.
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

ALGorithm:UPDate[:IMMediate]

ALGorithm:UPDate[:IMMediate] requests an immediate update of any scalar, array, algorithm code, ALG:STATE, or ALG:SCAN:RATIO changes that are pending.

Comments

- Variables and algorithms can be accepted during Phase 1-INPUT or Phase 2-UPDATE in Figure 6-1 when INIT is active. All writes to variables and algorithms occur to their buffered elements upon receipt. However, these changes do not take effect until the ALG:UPD:IMM command is processed at the beginning of the UPDATE phase. The update command can be received at any time prior to the UPDATE phase and will be the last command accepted. Note that the ALG:UPD:WINDow command specifies the maximum number of updates to do. If no update command is pending when entering the UPDATE phase, then this time is dedicated to receiving more changes from the system.
- As soon as the ALG:UPD:IMM command is received, no further changes are accepted until all updates are complete. A query of an algorithm value following an UPDate command will not be executed until the UPDate completes; this may be a useful synchronizing method.

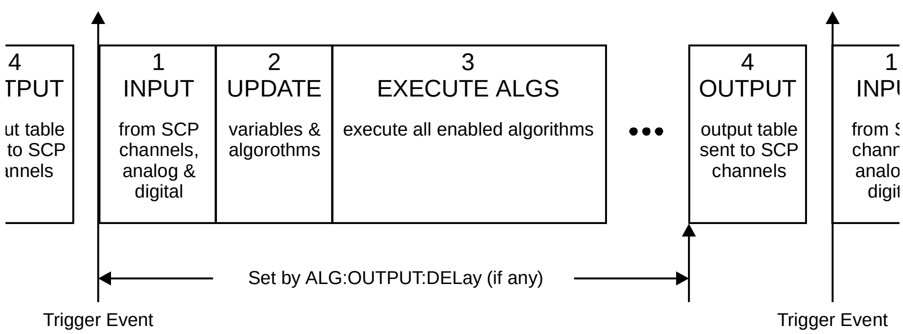


Figure 6-1. Updating Variables and Algorithms

- **When Accepted: Before or after INIT.**
- Related Commands: ALG:UPDATE:WINDOW, ALG:SCALAR, ALG:ARRAY, ALG:STATE, and ALG:SCAN:RATIO, ALG:DEF (with swapping enabled)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

The following example shows three scalars being written with the associated update command following. See ALG:UPD:WINDOW.

```
ALG:SCAL ALG1','Setpoint',25        provide 3 new alg scalar values
ALG:SCAL 'ALG1','P_factor',1.3
ALG:SCAL 'ALG2','P_factor',1.7
```

ALG:UPD	*update values in alg*
ALG:SCAL? 'ALG2','Setpoint'	*query for new updated scalar*

ALGorithm:UPDate:CHANnel

ALGorithm:UPDate:CHANnel <*dig_chan*> This command is used to update variables, algorithms, ALG:SCAN:RATIO, and ALG:STATE changes when the specified digital input level changes state. When the ALG:UPD:CHAN command is executed, the current state of the digital input specified is saved. The update will be performed at the next update phase (UPDATE in Figure 6-1), following the channel's change of digital state. This command is useful to synchronize multiple HP E1422s when you want all variable updates to be processed at the same time.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
dig_chan	Algorithm Language channel specifier (string)	Input channel for HP E1533: Iccc.Bb for HP E1534: Iccc where ccc=normal channel number and b=bit number (include ".B")	none

Comments

- The duration of the level change to the designated bit or channel MUST be at least the length of time between scan triggers. Variable and algorithm changes can be accepted during the INPUT or UPDATE phases (Figure 6-1) when INIT is active. All writes to variables and algorithms occur to their buffered elements upon receipt. However, these changes do not take effect until the ALG:UPD:CHAN command is processed at the beginning of the UPDATE phase. Note that the ALG:UPD:WINDow command specifies the maximum number of updates to do. If no update command is pending when entering the UPDATE phase, then this time is dedicated to receiving more changes from the system.

Note As soon as the ALG:UPD:CHAN command is received, the HP E1422 begins to closely monitor the state of the update channel and can not execute other commands until the update channel changes state to complete the update

- Note that an update command issued after the start of the UPDATE phase will be buffered but not executed until the beginning of the next INPUT phase. At that time, the current stored state of the specified digital channel is saved and used as the basis for comparison for state change. If at the beginning of the scan trigger the digital input state had changed, then at the beginning of the UPDATE phase the update command would detect a change from the previous scan trigger and the update process would begin.
- **When Accepted: Before and After INIT**.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

The following example shows three scalars being written with the associated update command following. When the ALG:UPD:CHAN command is received, it will read the current state of channel 108, bit 0. At the beginning of the UPDATE phase, a check will be made to determine if the stored state of channel 108 bit 0, is different from the current state. If so, the update of all three scalars take effect next Phase 2.

```
INIT
ALG:SCAL 'ALG1','Setpoint',25
ALG:SCAL 'ALG1','P_factor',1.3
ALG:SCAL 'ALG2','P_factor',1.7
ALG:UPD:CHAN 'I108.B0'
```

update on state change at bit zero of 8-bit channel 8

ALGorithm:UPDate:WINDow

ALGorithm:UPDate:WINDow <*num_updates*> specifies how many updates you may need to perform during phase 2 (UPDATE). The DSP will process this command and assign a constant window of time for UPDATE.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
num_updates	numeric (int16)	1 - 512	none

Comments

- The default value for num_updates is 20. If you know you will need fewer updates, specifying a smaller number will result in slightly greater loop execution speeds.
- This command creates a time interval in which to perform all pending algorithm and variable updates. To keep the loop times predictable and stable, the time interval for UPDATE is constant. That is, it exists for all active algorithms, each time they are executed whether or not an update is pending.
- ***RST Condition:** ALG:UPD:WIND 20
- **When Accepted: Before INIT only**.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

You decide you will need to update a maximum of 8 variables during run-time.

```
ALG:UPD:WIND 8
```

Notes

1. When the number of update requests exceeds the Update Queue size set with ALG:UPD:WINDOW by one, the module will refuse the request and will issue the error message "Too many updates in queue. Must send UPDATE command". Send ALG:UPDATE, then re-send the update request that caused the error.
2. The "Too many updates in queue..." error can occur before the module is INITialized. It's not uncommon with several algorithms defined, to have more

variables that need to be pre-set before INIT than you will change in one update after the algorithms are running. You may send INIT with updates pending. The INIT command automatically performs the updates before starting the algorithms.

ALGOrithm:UPDate:WINDow?

ALGOrithm:UPDate:WINDow? returns the number of variable, and algorithm updates allowed within the UPDATE window.

- **Returned Value:** number of updates in the UPDATEwindow. The type is **int16**
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

ARM

The ARM Subayatem is only useful when the TRIGer:SOURce is set to TIMer. With the HP E1422, when the TRIG:SOURCE is set to TIMer, an ARM event must occur to start the timer. This can be something as simple as executing the ARM[:IMMediate] command, or it could be another event selected by ARM:SOURCE.

Note The ARM subsystem may only be used then the TRIGger:SOURce is TIMer. If the TRIGger:SOURce is not TIMer and ARM:SOURce is set to anything other than IMMediate, an Error -221,"Settings conflict" will be generated.

The ARM command subsystem provides:

- An immediate software ARM (ARM:IMM).
- Selection of the ARM source (ARM:SOUR BUS | EXT | HOLD | IMM | SCP | TTLTRG<n>) when TRIG:SOUR is TIMer.

Figure 6-2 shows the overall logical model of the Trigger System.

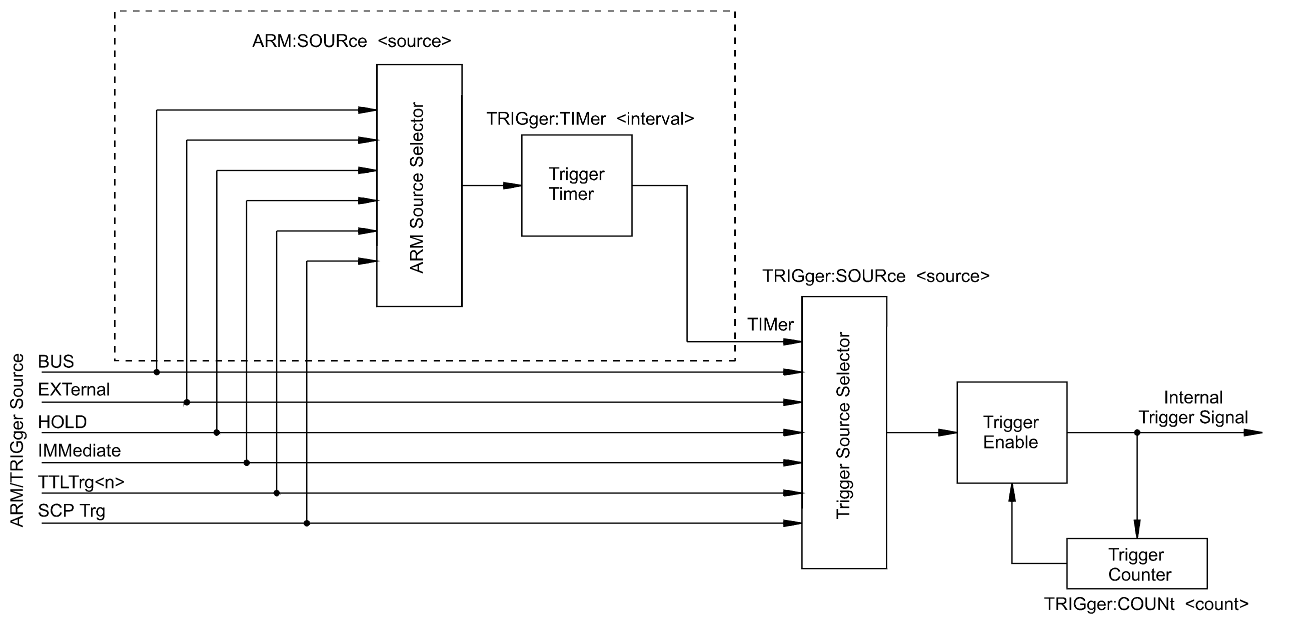


Figure 6-2. Logical Trigger Model

Subsystem Syntax

```
ARM
    [:IMMediate]
    :SOURce BUS | EXTernal | HOLD | IMMediate | SCP | TTLTrg<n>
    :SOURce?
```

ARM[:IMMediate]

ARM[:IMMediate] arms the trigger system when the module is set to the ARM:SOUR BUS or ARM:SOUR HOLD mode.

Comments

- **Related Commands:** ARM:SOURCE, TRIG:SOUR
- ***RST Condition:** ARM:SOUR IMM
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

ARM:IMM	*After INIT, system is ready for trigger event*
ARM	*Same as above (:IMM is optional)*

ARM:SOURce

ARM:SOURce <*arm_source*> configures the ARM system to respond to the specified source.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
arm_source	discrete (string)	BUS | EXT | HOLD | IMM | SCP | TTLTrg<n>	none

Comments

- The following table explains the possible choices.

BUS	ARM[:IMMediate]
EXTernal	“TRG” signal on terminal module
HOLD	ARM[:IMMediate]
IMMediate	The arm signal is always true (continuous arming).
SCP	SCP Trigger Bus (future HP or SCP Breadboard)
TTLTrg<n>	The VXIbus TTLTRG lines (n=0 through 7)

- See note about ARM subsystem on page 224.
- When TRIG:SOURCE is TIMER, an ARM event is required only to trigger the first scan. After that the timer continues to run and the module goes to the Waiting For Trigger State ready for the next Timer trigger. An ABORT command will return the module to the Trigger Idle State after the current scan

is completed. See TRIG:SOURce for more detail.

While ARM:SOUR is IMM, you need only INITiate the trigger system to start a measurement scan.

- When Accepted: Before INIT only.
- **Related Commands:** ARM:IMM, ARM:SOURCE?, INIT[:IMM], TRIG:SOUR
- ***RST Condition:** ARM:SOUR IMM
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

ARM:SOUR BUS	*Arm with ARM command*
ARM:SOUR TTLTRG3	*Arm with VXIbus TTLTRG3 line*

ARM:SOURce?

ARM:SOURce? returns the current arm source configuration. See the ARM:SOUR command for more response data information.

- **Returned Value:** Discrete, one of BUS, HOLD, IMM, SCP, or TTLT0 through TTLT7. The data type is **string**.
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

ARM:SOUR?	*An enter statement return arm source configuration*

CALibration

The Calibration subsystem provides for two major categories of calibration.

1. "A/D Calibration"; In these procedures, an external multimeter is used to calibrate the A/D gain on all 5 of its ranges. The multimeter also determines the value of the HP E1422's internal calibration resistor. The values generated from this calibration are then stored in nonvolatile memory and become the basis for "Working Calibrations. These procedures each require a sequence of several commands from the CALibration subsystem (**CAL:CONFIG**…, **CAL:VALUE**…, and **CAL:STORE ADC)**. Always execute *CAL? or a CAL:TARE operation after A/D Calibration.

2. "Working Calibration", of which there are three levels (see Figure 6-3):

-- "A/D Zero"; This function quickly compensates for any short term A/D converter offset drift. This would be called the auto-zero function in a conventional voltmeter. In the HP E1422 where channel scanning speed is of primary importance, this function is performed only when the **CAL:ZERO?** command is executed. Execute CAL:ZERO? as often as your control setup will allow.

-- "Channel Calibration"; This function corrects for offset and gain errors for each module channel. The internal current sources are also calibrated. This calibration function corrects for thermal offsets and component drift for each channel out to the input side of the Signal Conditioning Plug-On (SCP). All calibration sources are on-board and this function is invoked using either the ***CAL?** or **CAL:SETup** command.

-- "Channel Tare"; This function (**CAL:TARE**) corrects for voltage offsets in external system wiring. Here, the user places a short across transducer wiring and the voltage that the module measures is now considered the new "zero" value for that channel. The new offset value can be stored in non-volatile calibration memory (**CAL:STORE TARE**) but is in effect whether stored or not. System offset constants which are considered long-term should be stored. Offset constants which are measured relatively often would not require non-volatile storage. CAL:TARE automatically executes a *CAL?

-- "Remote Channel Calibration"; This function corrects for gain and offset errors in each channel of a Remote Signal Conditioning unit (RSC). Each RSC has its own calibration voltage source as well as shorting switches. The calibration source is measured through dedicated analog connections between the HP E1539A SCP and the RSC. The source is then used to stimulate the RSCs amplifiers to calibrate gain. The shorting switches provide a zero volt source to calibrate offset.

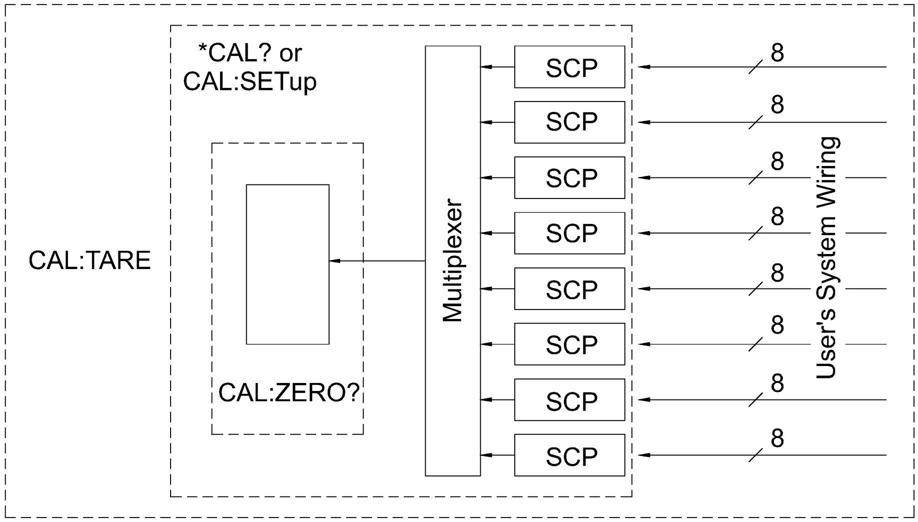


Figure 6-3. Levels of Working Calibration

Subsystem Syntax

CALibration
:CONFigure
:RESistance
:VOLTage <*range*>, ZERO | FS
:REMote?
:DATA <cal_data_block>
:DATA?
:STORe
:SETup
:SETup?
:STORe ADC | TARE
:TARE (@<*ch_list*>)
:RESet
:TARE?
:VALue
:RESistance <*ref_ohms*>
:VOLTage <*ref_volts*>
:ZERO?

CALibration:CONFigure:RESistance

CALibration:CONFigure:RESistance connects the on-board reference resistor to the Calibration Bus. A four-wire measurement of the resistor can be made with an external multimeter connected to the **HCAL**, **LCAL**, **HOHM**, and **LOHM** terminals on the Terminal Module, or the **V H**, **V L**, **Ω H**, and **Ω L** terminals on the Cal Bus connector when not using a terminal module.

Comments

- **Related Commands:** CAL:VAL:RES, CAL:STOR ADC
- **When Accepted:** Not while INITiated
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

CAL:CONF:RES	*connect reference resistor to Calibration Bus*
*OPC? or SYST:ERR?	*must wait for CAL:CONF:RES to complete*
(*now measure ref resistor with external DMM*)	
CAL:VAL:RES <measured value>	*Send measured value to module*
CAL:STORE ADC	*Store cal constants in non-volatile memory (used only at end of complete cal sequence)*

CALibration:CONFigure:VOLTage

CALibration:CONFigure:VOLTage <*range*>,<*zero_fs*> connects the on-board voltage reference to the Calibration Bus. A measurement of the source voltage can be made with an external multimeter connected to the **H Cal** and **L Cal** terminals on the Terminal Module, or the **V H**, **V L**, **Ω H**, and **Ω L** terminals on the Cal Bus connector when not using a terminal module. The *range* parameter controls the voltage level available when the *zero_fs* parameter is set to FSCale (full scale).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see comments	volts
zero_fs	discrete (string)	ZERO | FSCale	none

Comments

- The *range* parameter must be within ±5% of one of the 5 following values: .0625VDC, .25VDC, 1VDC, 4VDC, 16VDC
  range may be specified in millivolts (mv).
- The expected FSCALE output voltage of the calibration source will be approximately 90% of the nominal value for each range, except the 16V range where the output is 10V.
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:VAL:VOLT, STOR ADC
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

CAL:CONF:VOLTAGE .0625, ZERO	*connect voltage reference to Calibration Bus*
*OPC? or SYST:ERR?	*must wait for CAL:CONF:VOLT to complete*
(*now measure voltage with external DMM*)	
CAL:VAL:VOLT <measured value>	*Send measured value to module*

repeat above sequence for full-scale
repeat zero and full-scale for remaining ranges (.25, 1, 4, 16)
CAL:STORE ADC — *Store cal constants in non-volatile memory (used only at end of complete cal sequence)*

CALibration:REMote?

CALibration:REMote? (@<*ch_list*>) calibrates one or more entire Remote Signal Conditioning Units like the HP E1529A. Only a single channel per RSCU unit need be specified in <*ch_list*> and all channels on that RSC Unit will be calibrated. <ch_list> can contain multiple channels that specify multiple RSC Units. CAL:REM? returns a value when all RSC Units specified in <*ch_list*> have been calibrated (see comments below).

Note that the scope of the *CAL? and CAL:SETup commands is limited to the HP E1422A and the SCPs it contains. They do not calibrate Remote Signal Conditioning Units like the HP E1529A. You must use CAL:REMote in addition to *CAL?/CAL:SETup for RSCUs.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	channel list (string)	10000 - 15731	none

Comments

- Individual channels in *ch_list* must be for RSCUs, although channel ranges may span non-RSCU channels. If *ch_list* specifies a channel not connected through an HP E1539A SCP, a 3007 "Invalid signal conditioning plug-on" error is generated.

- **Returned Value:**

Value	Meaning	Further Action
0	Cal OK	None
1	Error during remote calibration	Error information in FIFO buffer. See faulure codes below.
-1	Couldn’t start remote calibration	Query the Error Queue (SYST:ERR?) See error messages starting on page 401

  The data type for this returned value is **int16**.

- Failure Information for +1 return: The FIFO buffer will contain pairs of values. The first value will be the failing channel, and the second value is the Failure Code for that channel. Failure Codes found in the FIFO buffer are:
  a. Offset exceeds limit. Failure code is 1000.0 + (the offset measured)
  b. Gain error exceeds limit. Failure code is 2000.0 + (ideal gain - actual gain)

- Immediately after CAL:REM?, the new calibration constants are used for subsequent measurements but are in volatile memory. Where these calibration values need to be retained for long periods, they can be stored into non-volatile memory using the CAL:REM:STORE command.

- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

CALibration:REMote:DATA

CALibration:REMote:DATA <cal_data_block> restores the remote calibration constants acquired using the CAL:REM:DATA? query after a remote calibration operation. These calibration constants go into effect immediately.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
cal_data_block	definite length block data (float32 array)	see comments	none

Comments

- CAL:REM:DATA sends to the HP E1422A a definite length block of 1024 float64 values that represent an offset and gain pair (in that order) for each of 512 possible remote channels. The block must always be 1024, float64 values (8192 bytes) regardless how many RSCUs are actually connected to the HP E1422A. Values for channel positions that are not installed are "place holders".

- ***RST Condition:** Stored calibration constants are unchanged

- **Send with VXIplug&play Function:** hpe1422_sendBlockReal64(...)

CALibration:REMote:DATA?

CALibration:REMote:DATA? extracts the remote calibration constants generated using the CAL:REMote? command.

Comments

- CAL:REM:DATA returns a definite length block of 1024 float32 values that represent a gain and offset pair for each of 512 possible remote channels. The block is always 1024, float64 values (8192 bytes) regardless how many RSCUs are actually connected to the HP E1422A. Values for channel positions where RSCUs are not installed are set to 0.000.

- **Returned Value:** the 1024 float64 values form 512 channel calibration pairs. A pair of calibration constants consists of first a channel offset value, then a channel gain value.

- ***RST Condition:** Stored calibration constants are unchanged

- **Send with VXIplug&play Function:** hpe1422_cmdReal64Arr_Q(...)

CALibration:REMote:STORe

CALibration:REMote:STORe (@<*ch_list*>) copies the calibration constants held in working RAM since remote calibration into the RSCU's non-volatile flash memory. Only a single channel per RSCU unit need be specified in <*ch_list*> and all cal constants for that RSC Unit will be stored. <*ch_list*> can contain multiple channels that specify multiple RSC Units.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	channel list (string)	10000 - 15731	none

Comments

- Individual channels in *ch_list* must be for RSCUs. If *ch_list* specifies a channel not connected through an HP E1539A SCP, a 3007 "Invalid signal conditioning plug-on" error is generated.

Note An RSCU's Flash Memory has a finite lifetime of approximately ten thousand write cycles (unlimited read cycles). While executing CAL:REM:STOR once every day would not exceed the lifetime of the Flash Memory for approximately 27 years, an application that stored constants many times each day would unnecessarily shorten the Flash Memory's lifetime. See Comments below.

- After remote calibration, an RSCUs calibration constants are in live (volatile) memory and are available for operation. If you plan to calibrate your RSCUs often, (especially after a line power failure, you don,t have to store them in flash memory.
- **Send with VXIplug&play Function:** hpe1422_cmd_Q(...)

CALibration:SETup

CALibration:SETup causes the Channel Calibration function to be performed for every module channel with an analog SCP installed (input or output). The Channel Calibration function calibrates the A/D Offset, and the Gain/Offset for these analog channels. This calibration is accomplished using internal calibration references. For more information see *CAL? on page 351.

Note that the scope of the *CAL? and CAL:SETup commands is limited to the HP E1422A and the SCPs it contains. They do not calibrate Remote Signal Conditioning Units like the HP E1529A. You must use CAL:REMote? in addition to *CAL?/CAL:SETup for RSCs.

Comments

- CAL:SET performs the same operation as the *CAL? command except that since it is not a query command it doesn't tie-up the driver waiting for response data from the instrument. If you have multiple HP E1422s in your system you can start a CAL:SET operation on each and then execute a CAL:SET? command to complete the operation on each instrument.

- **Related Commands:** CAL:SETup?, *CAL?
- **When Accepted:** Not while INITiated
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

CAL:SET	*start SCP Calibration on 1st HP E1422*
:	*start SCP Calibration on more HP E1422s*
CAL:SET	*start SCP Calibration on last HP E1422*
CAL:SET?	*query for results from 1st HP E1422*
:	*query for results from more HP E1422s*
CAL:SET?	*query for results from last HP E1422*

CALibration:SETup?

CALibration:SETup? Returns a value to indicate the success of the last CAL:SETup or *CAL? operation. CAL:SETup? returns the value only after the CAL:SETup operation is complete.

Comments

- **Returned Value:**

Value	Meaning	Further Action
0	Cal OK	None
-1	Cal Error	Query the Error Queue (SYST:ERR?) See error message “3026” on page 402 Also run *TST?
-2	No results available	No *CAL? or CAL:SETUP done

The data type for this returned value is **int16**.

- **Related Commands:** SYST:ERR?, CAL:SETup, *CAL?
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage see CAL:SETup

CALibration:STORe

CALibration:STORe <*type*> stores the HP E1422A’s most recently measured calibration constants into Flash Memory (Electrically Erasable Programmable Read Only Memory). When *type*=ADC, the module stores its A/D calibration constants as well as constants generated from *CAL?/CAL:SETup into Flash Memory. When *type*=TARE, the module stores the most recently measured CAL:TARE channel offsets into Flash Memory.

Note The HP E1422's Flash Memory has a finite lifetime of approximately ten thousand write cycles (unlimited read cycles). While executing CAL:STOR once every day would not exceed the lifetime of the Flash Memory for approximately 27 years, an application that stored constants many times each day would unnecessarily shorten the Flash Memory's lifetime. See Comments below.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	discrete (string)	ADC | TARE	none

Comments

- The Flash Memory Protect jumper (JM2201) must be set to the enable position before executing this command (See "Disabling Flash Memory Access (optional)" on page 27).
- Channel offsets are compensated by the CAL:TARE command even when not stored in the Flash Memory. There is no need to use the CAL:STORE TARE command for channels which are re-calibrated frequently.
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:VAL:RES, CAL:VAL:VOLT
- ***RST Condition:** Stored calibration constants are unchanged
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

CAL:STORE ADC	*Store cal constants in non-volatile memory after A/D calibration*
CAL:STORE TARE	*Store channel offsets in non-volatile memory after channel tare*

Command Sequence

Storing A/D cal constants

perform complete A/D calibration, then...
CAL:STORE ADC

Storing channel tare (offset) values

CAL:TARE <*ch_list*>	*to correct channel offsets*
CAL:STORE TARE	*Optional depending on necessity of long term storage*

CALibration:TARE

CALibration:TARE (@<*ch_list*>) measures offset (or tare) voltage present on the channels specified and stores the value in on-board RAM as a calibration constant for those channels. Future measurements made with these channels will be compensated by the amount of the tare value. Use CAL:TARE to compensate for

voltage offsets in system wiring and residual sensor offsets. Where tare values need to be retained for long periods, they can be stored in the module’s Flash Memory (Electrically Erasable Programmable Read Only Memory) by executing the CAL:STORe TARE command.
For more information See “Compensating for System Offsets” on page 145

Note for RSC Units

CAL:TARE does not remove offsets in an RSCs field wiring. For the HP E1529A, the SENS:STR:UNSTrained value is the equivalent of the tare offset. Any offset in the analog signal line between the HP E1529A SCP and an RSC is accounted for during the CAL:REMote? operation.

Notes For Thermocouples

1. You must not use CAL:TARE on field wiring that is made up of thermocouple wire. The voltage a thermocouple wire pair generates can not be removed by introducing a short anywhere between its junction and its connection to an isothermal panel (either the HP E1422's Terminal Module or a remote isothermal reference block). Thermal voltage is generated along the entire length of a thermocouple pair where there is any temperature gradient along that length. To CAL:TARE thermocouple wire this way would introduce an unwanted offset in the voltage/temperature relationship for that channel. If you inadvertently CAL:TARE a thermocouple wire pair, use CAL:TARE:RESET to reset all tare constants to zero.
2. You should use CAL:TARE to compensate wiring offsets (copper wire, not thermocouple wire) between the HP E1422 and a remote thermocouple reference block. Disconnect the thermocouples and introduce copper shorting wires between each channel's HI and LO, then execute CAL:TARE for these channels.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	channel list (string)	100 - 15731	none

Comments

- CAL:TARE also performs the equivalent of a *CAL? operation. This operation uses the Tare constants to set a DAC which will remove each channel offset as "seen" by the module’s A/D converter. As an example assume that the system wiring to channel 0 generates a +0.1Volt offset with 0Volts (a short) applied at the Unit Under Test (UUT). Before CAL:TARE the module would return a reading of 0.1Volts for channel 0. After CAL:TARE (@100), the module will return a reading of 0Volts with a short applied at the UUT and the system wiring offset will be removed from all measurements of the signal to channel 0.

- The CAL:TARE command may be issued to several HP E1422As to be later completed with a matching CAL:TARE? query sent to each instrument. Note if the DIAG:CAL:TARE:OTD:MODE is set to "1" then the CAL:TARE command will not return until the calibration is complete.

- Set Amplifier/Filter SCP gain before CAL:TARE. For best accuracy, choose the gain that will be used during measurements. If you decide to change the range or gain setup later, be sure to perform another *CAL?.

- If Open Transducer Detect (OTD) is enabled when CAL:TARE is executed, the module will disable OTD, wait 1 minute to allow channels to settle, perform the calibration, and then re-enable OTD. If your program turns off OTD before executing CAL:TARE, your application should also wait 1 minute for settling. If the DIAG:CAL:TARE:OTD:MODE is set to "1", the OTD will remain enabled throughout the TARE calibration. This allows the voltage generated by the OTD current to also be removed by the TARE cal.

- The maximum voltage that CAL:TARE can compensate for is dependent on the range chosen and SCP gain setting. The following table lists these values.

Maximum CAL:TARE Offsets				
A/D range ±V F.Scale	Offset V Gain x1	Offset V Gain x8	Offset V Gain x16	Offset V Gain x64
16	3.2213	.40104	.20009	.04970
4	.82101	.10101	.05007	.01220
1	.23061	.02721	.01317	.00297
.25	.07581	.00786	.00349	.00055
.0625	.03792	.00312	.00112	n/a

- Channel offsets are compensated by the CAL:TARE command even when not stored in the Flash Memory. There is no need to use the CAL:STORE TARE command for channels which are re-calibrated frequently.

- The HP E1422's Flash Memory has a finite lifetime of approximately ten thousand write cycles (unlimited read cycles). While executing CAL:STOR once every day would not exceed the lifetime of the Flash Memory for approximately 27 years, an application that stored constants many times each day would unnecessarily shorten the Flash Memory's lifetime. See Comments below.

- Executing CAL:TARE sets the Calibrating bit (bit 0) in Operation Status Group. Executing CAL:TARE? resets the bit.

- Because CAL:TARE also performs a *CAL? operation, completion of CAL:TARE may take many minutes to complete. The actual time it will take your HP E1422 to complete CAL:TARE depends on the mix of SCPs installed. CAL:TARE performs literally hundreds of measurements of the internal calibration sources for each channel and must allow 17 time constants of settling wait each time a filtered channel's calibrations source value is changed. The CAL:TARE procedure is internally very sophisticated and results in an extremely well calibrated module.

- Any output type channels in <ch_list> are ignored during CAL:TARE.
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:TARE?, CAL:STOR TARE, DIAG:CAL:TARE:OTD:MODE
- ***RST Condition:** Channel offsets are not affected by *RST.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

CAL:TARE *<ch_list>*	*to correct channel offsets*
CAL:TARE?	*to return the success flag from the CAL:TARE operation*
CAL:STORE TARE	*Optional depending on necessity of long term storage*

CALibration:TARE:RESet

CALibration:TARE:RESet resets the tare calibration constants to zero for all 64 channels. Executing CAL:TARE:RES affects the tare cal constants in RAM only. To reset the tare cal constants in Flash Memory, execute CAL:TARE:RES and then execute CAL:STORE TARE.

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

CAL:TARE:RESET	*to reset channel offsets*
CAL:STORE TARE	*Optional if necessary to reset tare cal constants in Flash Memory.*

CALibration:TARE?

CALibration:TARE? Returns a value to indicate the success of the last CAL:TARE operation. CAL:TARE? returns the value only after the CAL:TARE operation is complete.

- **Returned Value:**

Value	Meaning	Further Action
0	Cal OK	None
-1	Cal Error	Query the Error Queue (SYST:ERR?) See "Error Messages" on page 401 Also run *TST?
-2	No results available	Perform CAL:TARE before CAL:TARE?

The data type for this returned value is **int16**.

- Executing CAL:TARE sets the Calibrating bit (bit 0) in Operation Status Group.

Executing CAL:TARE? resets the bit.

- **Related Commands:** CAL:STOR TARE
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Command Sequence

CAL:TARE <*ch_list*>	*to correct channel offsets*
CAL:TARE?	*to return the success flag from the CAL:TARE operation*
CAL:STORE TARE	*Optional depending on necessity of long term storage*

CALibration:VALue:RESistance

CALibration:VALue:RESistance <*ref_ohms*> sends the just-measured value of the on-board reference resistor to the module for A/D calibration.

Parameters

Parameter Name	Parameter Type	Range of Value	Default Units
ref_ohms	numeric (float32)	7,500 ± 5%	ohms

Comments

- Use the CAL:CONF:RES command to configure the reference resistor for measurement at the Calibration Bus connector.
- A four-wire measurement of the resistor can be made with an external multimeter connected to the **HCAL**, **LCAL**, **HOHM**, and **LOHM** terminals on the Terminal Module, or the **V H**, **V L**, **Ω H**, and **Ω L** terminals on the Cal Bus connector when not using a terminal module.
- *ref_ohms* must be within 5% of the 7500Ω nominal reference resistor value or a -222 'Data out of range' error will be generated. If this error occurs, verify your external measurement equipment and run *TST? on your 1422A.
- *ref_ohms* may be specified in Kohm (kohm).
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:CONF:RES, CAL:STORE ADC
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

CAL:CONF:RES	
(*now measure ref resistor with external DMM*)	
CAL:VAL:RES <measured value>	*Send measured value to module*

CALibration:VALue:VOLTage

CALibration:VALue:VOLTage <*ref_volts*> sends the value of the just-measured

DC reference source to the module for A/D calibration.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ref_volts	numeric (float32)	must be within +10% or -50% of the "expected" source output value	volts

Comments

- The "expected" output values for the voltage reference source is:
  0.9 * Nominal Range Value for the .0625 through 4 volt ranges.
  10 volts for the 16 volt range.
- Use the CAL:CONF:VOLT command to configure the on-board voltage source for measurement by an external reference voltmeter via the Calibration Bus terminals.
- A measurement of the source voltage can be made with an external multimeter connected to the **HCAL**, and **LCAL** terminals on the Terminal Module, or the **V H**, **and V L** terminals on the Cal Bus connector when not using a terminal module.
- The *ref_volts* value given must be for the currently configured range and output (zero or full scale) as set by the previous **CAL:CONF:VOLT <*range*>, ZERO | FSCale** command.
- *ref_volts* must be within 4% of the actual reference voltage value as read after CAL:CONF:VOLT, or an error 3042 '0x400: DSP-DAC adjustment went to limit' will be generated. If the reading on your external reference voltmeter is in excess of 4% error from nominal voltage, verify your voltmeter and execute *TST? on the HP E1422A.
- *ref_volts* may be specified in millivolts (mv).
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:CONF:VOLT, CAL:STORE ADC
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Command Sequence

```
CAL:CONF:VOLTAGE  4,FSCALE
*OPC?                                Wait for operation to complete
enter statement
      (now measure voltage with external DMM)
CAL:VAL:VOLT  <measured value>       Send measured value to module
```

CALibration:ZERO?

CALibration:ZERO? corrects Analog to Digital converter offset for any drift since the last *CAL? or CAL:ZERO? command was executed. The offset calibration takes about 5 seconds and should be done as often as you control set up allows.

Comments

- The CAL:ZERO? command only corrects for A/D offset drift (zero). Use the *CAL? common command to perform on-line calibration of channels as well as A/D offset. *CAL? performs gain and offset correction of the A/D and each channel with an analog SCP installed (both input and output).

- **Returned Value:**

Value	Meaning	Further Action
0	Cal OK	None
-1	Cal Error	Query the Error Queue (SYST:ERR?) See “Error Messages” on page 401

  The data type for this returned value is **int16**.

- Executing this command **does not** alter the module’s programmed state (function, range etc.).

- **Related Commands:** *CAL?

- ***RST Condition:** A/D offset performed

- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

```
CAL:ZERO?
enter statement here                    returns 0 or -1
```

DIAGnostic

The DIAGnostic subsystem allows you to perform special operations that are not standard in the SCPI language. This includes checking the current revision of the Control Processor's firmware, and that it has been properly loaded into Flash Memory.

Subsystem Syntax

```
DIAGnostic
    :CALibration
        :SETup
            :MODE  0 | 1
            :MODE?
        :TARe
            [:OTD]
                :MODE  0 | 1
                :MODE?
    :CHECksum?
    :CONNect  <source>,<mode>,(@<ch_list>)
    :CUSTom
        :MXB <slope>,<offset>,(@<ch_list>)
        :PIECewise <table_range>,<table_block>,(@<ch_list>)
        :REFerence
            :TEMPerature
    :IEEE  1 | 0
    :IEEE?
    :INTerrupt
        [:LINe]  <intr_line>
        [:LINe]?
    :OTDetect
        [:STATe]  1 | 0 | ON | OFF,(@<ch_list>)
        [:STATe]?  (@<channel>)
    :QUERy
        :SCPREAD? <reg_addr>
    :REMote
        :USER
            :DATA  <user_data_block>,(@<ch_list>)
            :DATA?  (@<ch_list>)
    :TEST
        :REMote
            :NUMber?  <test_num>,<iterations>,(@<channel>)
            :SELFtest?  (@<channel>)
        :SELFtest?
    :VERSion?
```

DIAGnostic:CALibration:SETup[:MODE]

DIAGnostic:CALibration:SETup[:MODE] <*mode*> sets the type of calibration to use for analog output SCPs like the HP E1531 and HP E1532 when *CAL? or

CAL:SET are executed.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	boolean (uint 16)	0 | 1	volts

Comments

- When <*mode*> is set to 1 (the *RST Default) channels are calibrated using the Least Squares Fit method to provide the minimum error overall (over the entire output range). When <*mode*> is 0, channels are calibrated to provide the minimum error at their zero point. See your SCPs User's Manual for its accuracy specifications using each mode.
- **Related Commands:** *CAL?, CAL:SET, DIAG:CAL:SET:MODE?
- ***RST Condition:** DIAG:CAL:SET:MODE 1
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

set analog DAC SCP cal mode for best zero accuracy

DIAG:CAL:SET:MODE 0	*set mode for best zero cal*
*CAL?	*start channel calibration*

DIAGnostic:CALibration:SETup[:MODE]?

DIAGnostic:CALibration:SETup[:MODE]? returns the currently set calibration mode for analog output DAC SCPs.

Comments

- Returns a 1 when channels are calibrated using the Least Squares Fit method to provide the minimum error overall (over the entire output range). Returns a 0 when channels are calibrated to provide the minimum error at their zero point. See your SCPs User's Manual for its accuracy specifications using each mode. The data type is **int16**.
- **Related Commands:** DIAG:CAL:SET:MOD, *CAL?, CAL:SET
- ***RST Condition:** DIAG:CAL:SET:MODE 1
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

DIAGnostic:CALibration:TARE[:OTDetect]:MODE

DIAGnostic:CALibration:TARE[:OTDetect]:MODE <*mode*> sets whether Open Transducer Detect current will be turned off or left on (the default mode) during the CAL:TARE operation.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	boolean (uint 16)	0 | 1	volts

Comments

- When <*mode*> is set to 0 (the *RST Default), channels are tare calibrated with their OTD current off. When <*mode*> is 1, channels that have their OTD current on (DIAGnostic:OTDetect ON,(@<*ch_list*>)) are tare calibrated with their OTD current left on.
- By default (*RST) the CALibration:TARE? command will calibrate all channels with the OTD circuitry disabled. This is done for two reasons: first, most users do not leave OTD enabled while taking readings, and second, the CALibration:TARE? operation takes much longer with OTD enabled. However, for users who intend to take readings with OTD enabled, setting DIAG:CAL:TARE:OTD:MODE to 1, will force the CAL:TARE? command to perform calibration with OTD enabled on channels so specified by the user with the DIAG:OTD command.
- **Related Commands:** *CAL?, CAL:SET, DIAG:CAL:SET:MODE?
- ***RST Condition:** DIAG:CAL:TARE:MODE 0
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
configure OTD on during CAL:TARE
DIAG:CAL:TARE:MODE 1          set mode for OTD to stay on
CAL:TARE?                     start channel tare cal.
```

DIAGnostic:CALibration:TARE[:OTDetect]:MODE?

DIAGnostic:CALibration:TARE[:OTDetect]:MODE? returns the currently set mode for controlling Open Transducer Detect current while performing CAL:TARE? operation.

Comments

- Returns a 0 when OTD current will be turned off during CAL:TARE?. Returns 1 when OTD current will be left on during CAL:TARE? operation. The data type is **int16**.
- **Related Commands:** DIAG:CAL:TARE:MOD, DIAG:OTD, CAL:TARE?
- ***RST Condition:** DIAG:CAL:TARE:MODE 0
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

DIAGnostic:CHECksum?

DIAGnostic:CHECksum? performs a checksum operation on Flash Memory. A returned value of 1 indicates that Flash memory contents are correct. A returned value of 0 indicates that the Flash Memory is corrupted, or has been erased.

Comments

- **Returned Value:** Returns 1 or 0. The data type is **int16**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage DIAG:CHEC? *Checksum Flash Memory, return 1 for OK, 0 for corrupted*

DIAGnostic:CONNect

DIAGnostic:CONNect <*source*>,<*mode*>,(@<*ch_list*>) configures the HP E1529A to verify its measurement paths by measuring either the internal calibration source or an internal short for all 32 channels. You must define a matching scan list, trigger the instrument, and read the results from the FIFO or CVT.

Note The command DIAG:TEST:REMote:SELFtest? actually performs all of the verification functions this command provides and in addition includes filter and scanner tests.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
source	discrete (string)	NORM | SHORt | SOURce	none
mode	discrete (string)	ALL | ALT	none
ch_list	channel list (string)	10000 - 15700	none

Comments

- <*source*> specifies the source to measure. NORMal configures all inputs to measure user inputs. SHORT specifies the internal calibration short. SOURce specifies the internal 100mV calibration source.
- <*mode*>: ALL connects all channels to the specified <source>. ALT connects channels alternately to the SHORt or the SOURce. When <mode> is ALT, the <source> parameter specifies which source is connected to the first channel.For example, when <source> is SHORt, even channels are 0V, odd channels are .1V
- <*ch_list*> specifies which HP E1529A to configure. Specifying any channel on the an HP E1529A configures all channels on the unit.
- You must execute DIAG:CONN NORM,ALL,(@<ch_list>) to reset units for normal measurements.
- **Related Commands:** [SENSe:]DATA:FIFO?
- ***RST Condition:** DIAG:CONN ALL, NORM for all HP E1529A channels
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

DIAGnostic:CUSTom:MXB

DIAGnostic:CUSTom:MXB <*slope*>,<*offset*>,(@<*ch_list*>) sends the <slope> and <offset> parameters that allow the driver to calculate and download a custom linear Engineering Unit Conversion table to the HP E1422A. Use the "[SENSe:]FUNCtion:CUSTom" on page 303 to link this custom EU conversion with channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
slope	numeric (float32)	limit of float32	none
offset	numeric (float32)	limit of float32	none
ch_list	channel list (string)	100 - 163	none

Comments

- <*slope*> specifies the linear function's "slope": $(f_{outp1} - f_{outp0})/(V_{in1} - V_{in0})$
- <*offset*> specifies the conversion offset at zero input volts. This parameter is also commonly known as the "Y-intercept".
- <*ch_list*> specifies which channels may use this custom linear function.
- **Related Commands:** [SENSe:]FUNCtion:CUSTom (<*ch_list*>)
- ***RST Condition:** All custom EU tables erased
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

DIAGnostic:CUSTom:MXB

DIAGnostic:CUSTom:MXB <*slope*>,<*offset*>,(@<*ch_list*>) sends the <slope> and <offset> parameters that allow the driver to calculate and download a custom linear Engineering Unit Conversion table to the HP E1422A. Use the "[SENSe:]FUNCtion:CUSTom" on page 303 to link this custom EU conversion with channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
slope	numeric (float32)	limit of float32	none
offset	numeric (float32)	limit of float32	none
ch_list	channel list (string)	100 - 163	none

Comments

- <*slope*> specifies the linear function's "slope": $(f_{outp1} - f_{outp0})/(V_{in1} - V_{in0})$

- <*offset*> specifies the conversion offset at zero input volts. This parameter is also commonly known as the "Y-intercept".
- <*ch_list*> specifies which channels may use this custom linear function.
- **Related Commands:** [SENSe:]FUNCtion:CUSTom (<*ch_list*>)
- ***RST Condition:** All custom EU tables erased
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
DIAG:CUST:MXB 2.1,.19,(@10000:10131)    create table for chs 0000-0131
SENS:FUNC:CUST 1,1,(@10000:10131)       link custom EU with chs 0000-0131
```

DIAGnostic:CUSTom:PIECewise

DIAGnostic:CUSTom:PIECewise <*table_range*>,<*table_block*>, (@<*ch_list*>) downloads a custom piece wise Engineering Unit Conversion table (in <*table_block*>) to the HP E1422. Contact your Hewlett-Packard System Engineer for more information on Custom Engineering Unit Conversion for your application.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
table_range	numeric (float32)	.015625 | .03125 | .0625 | .125 | .25 | .5 | 1 | 2 | 4 | 8 | 16 | 32 | 64	volts
table_block	definite length block data	see comments	none
ch_list	channel list (string)	100 - 163	none

Comments

- <*table_block*> is a block of 1,024 bytes that define 512 16-bit values. SCPI requires that <*table_block*> include the definite length block data header. The VXIplug&play function *hpe1415_sendBlockUInt16(ViSession vi, ViString cmd_str, ViInt32 table[ ], ViInt32 size)* adds the header for you.
- <*table_range*> specifies the range of voltage that the table covers (from -<*table_range*> to +<*table_range*>).
- <*ch_list*> specifies which channels may use this custom EU table.
- **Related Commands:** [SENSe:]FUNCtion:CUSTom
- ***RST Condition:** All custom EU tables erased.
- **Use VXIplug&play function:** hpe1422_sendBlockUInt16(...)

Usage

```
program puts table constants into array table_block
DIAG:CUST:PIEC table_block,(@124:131)    send table for chs 24-31 to HP E1422
SENS:FUNC:CUST 1,1,(@124:131)            link custom EU with chs 24-31
```

INITiate then TRIGger module

DIAGnostic:CUSTom:REFerence:TEMPerature

DIAGnostic:CUSTom:REFerence:TEMPerature extracts the current Reference Temperature Register Contents, converts it to 32-bit floating point format and sends it to the FIFO. This command is used to verify that the reference temperature is as expected after measuring it using a custom reference temperature EU conversion table.

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

your program must have EU table values stored in *table_block*
download the new reference EU table
DIAG:CUST:PIECEWISE <*table_range*>,<*table_block*>,(@<*ch_list*>)
designate channel as reference
SENS:FUNC:CUST:REF <*range*>,(@<*ch_list*>)
set up scan list sequence (ch 0 in this case)
Now run the algorithm that uses the custom reference conversion table
dump reference temp register to FIFO
DIAG:CUST:REF:TEMP
read the diagnostic reference temperature value
SENS:DATA:FIFO?

DIAGnostic:IEEE

DIAGnostic:IEEE <*mode*> enables (1) or disables (0) IEEE-754 NAN (Not A Number) and ±INF value outputs. This command was created for the HP VEE platform.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	boolean (uint 16)	0 | 1	volts

Comments

- When <*mode*> is set to 1, the module can return ±INF and NAN values according to the IEEE-754 standard. When <*mode*> is set to 0, the module returns values as ±9.9E37 for INF and 9.91E37 for NAN.
- **Related Commands:** DIAG:IEEE?
- ***RST Condition:** DIAG:IEEE 1
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

Set IEEE mode

DIAG:IEEE 1 *INF values returned in IEEE standard*

DIAGnostic:IEEE?

DIAGnostic:IEEE? returns the currently set IEEE mode.

Comments

- The data type is **int16**.
- **Related Commands:** DIAG:IEEE
- ***RST Condition:** DIAG:IEEE 1
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

DIAGnostic:INTerrupt[:LINe]

DIAGnostic:INTerrupt[:LINe] <*intr_line*> sets the VXIbus interrupt line the module will use.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
intr_line	numeric (int16)	0 through 7	none

Comments

- **Related Commands:** DIAG:INT:LINE?
- **Power-on and *RST Condition:** DIAG:INT:LINE 1
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage DIAG:INT:LINE 5 *Module will interrupt on interrupt line 5*

DIAGnostic:INTerrupt[:LINe]?

DIAGnostic:INTerrupt[:LINe]? returns the VXIbus interrupt line that the module is set to use.

Comments

- **Returned Value:** Numeric 0 through 7. The data type is **int16**.
- **Related Commands:** DIAG:INT:LINE
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage DIAG:INT? *Enter statement will return 0 through 7*

DIAGnostic:OTDetect[:STATe]

DIAGnostic:OTDetect[:STATe] <*enable*>,(@<*ch_list*>) enables and disables the HP E1422's "Open Transducer Detection" capability (OTD). When Open Transducer

Detection is enabled, a very high impedance path connects all SCP channels to a voltage source greater than 16 volts. If an enabled channel has an open transducer, the input signal becomes the source voltage and the channel returns an input over-range value. The value returned is +9.91E+37 (ASCII).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	1 | 0 | ON | OFF	none
ch_list	channel list (string)	100 - 163	none

Comments

- Open Transducer Detection is enabled/disabled on a whole Signal Conditioning Plug-on basis. Selecting any channel on an SCP selects all channels on that SCP (8 channels per SCP).
- The DIAG:CAL:TARE:MODE <*mode*> command affects how OTD is controlled during the CAL:TARE? operation. When <*mode*> is set to 0 (the *RST Default), channels are tare calibrated with their OTD current off. When <*mode*> is 1, channels that have their OTD current on (DIAGnostic:OTDetect ON,(@<*ch_list*>)) are tare calibrated with their OTD current left on.
- **Related Commands:** DIAG:OTDETECT:STATE?, DIAG:CAL:TARE:MODE

Note ***RST Condition:** DIAG:OTDETECT OFF

If OTD is enabled when *CAL?, or CAL:TARE is executed, the module will disable OTD, wait 1 minute to allow channels to settle, perform the calibration, and then re-enable OTD.

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

DIAG:OTD ON,(@100:107,115:123)	*select OTD for the first and third SCP (complete channel lists for readability only)*
DIAG:OTD:STATE ON,(@100,115)	*same function as example above (only first channel of each SCP specified)*
DIAG:OTDETECT:STATE OFF,(@108)	*disable OTD for the 8 channels on the second SCP (only first channel of SCP specified)*

DIAGnostic:OTDetect[:STATe]?

DIAGnostic:OTDetect[:STATe]? (@<*channel*>) returns the current state of "Open Transducer Detection" for the SCP containing the specified *channel*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- *channel* must specify a single channel only.
- **Returned Value:** Returns 1 (enabled) or 0 (disabled). The data type is **int16**.
- **Related Commands:** DIAG:OTDETECT:STATE ON | OFF
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

DIAG:OTD:STATE? (@108)   *enter statement returns either a 1 or a 0*

DIAGnostic:QUERy:SCPREAD?

DIAGnostic:QUERy:SCPREAD? <*reg_addr*> returns data word from a Signal Conditioning Plug-on register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
reg_addr	numeric (int32)	0-65,535	none

Comments

- **NOTE:** This command **may not** be used while instrument is INITed.
- **Returned Value:** returns numeric register value. data type is **int32**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt32_Q(...)

Usage

DIAG:QUERY:SCPREAD? 258   *read Watchdog SCP's config/status register*
enter statement here   *return SCP ID value*

DIAGnostic:REMote:USER:DATA

DIAGnostic:REMote:USER:DATA <user_data_block>,(@<*channel*>)stores 894, 16-bit words of arbitrary user data to non-volatile flash memory. You can design your own format for the information you wish to store. For example, your data could define a 32 by 28 word array to store information about each channel.

Note A Remote Signal Conditioning Unit's Flash Memory has a finite lifetime of approximately ten thousand write cycles (unlimited read cycles). While executing DIAG:REM:USER:DATA once every day would not exceed the lifetime of the Flash Memory for approximately 27 years, an application that stored constants many times each day would unnecessarily shorten the

Flash Memory's lifetime.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
user_data_block	definite length block data (int16 array)	each element -32768 - 32767	none
channel	channel list (string)	10000 - 15731	none

Comments

- *channel* must specify a single channel only. The channel must be on an RSCU that supports the DIAG:REM:USER:DATA commands.
- DIAG:REM:USER:DATA sends to the RSCU a definite length block of 894 int16 values (1,792 bytes). The block must always be 894 words in length.
- ***RST Condition:** Stored values not changed by *RST
- **Use VXIplug&play function:** hpe1422_sendBlockInt16(...)

DIAGnostic:REMote:USER:DATA?

DIAGnostic:REMote:USER:DATA? (@<*channel*>) extracts 894, 16-bit words of arbitrary user data from non-volatile flash memory (stored with the DIAG:REM:USER:DATA command).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- *channel* must specify a single channel only. The channel must be on an RSCU that supports the DIAG:REM:USER:DATA commands.
- **Returned Value:** DIAG:REM:USER:DATA? returns an IEEE definite length data block which represents an array of 894, int16 values.
- **RST Condition:** Stored values not changed by *RST
- **Send with VXIplug&play Function:** hpe1422_cmdInt16Arr_Q(...)

DIAGnostic:TEST:REMote:NUMber?

DIAGnostic:TEST:REMote:NUMber? <*test_num*>,<*iterations*>,(@<*channel*>) executes a selected self-test number on a single Remote Signal Conditioning Unit connected through the HP E1539A SCP. See DIAG:TEST:REM:SELF? for details of each test.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
test_num	numeric	1 - 5	none
iterations	numeric	1 - 32767	none
channel	channel list (string)	10000 - 15731	none

Comments

- <*test_num*> specifies the test to perform. See "DIAGnostic:TEST:REMote:SELFtest?" on page 252 for explanations of test numbers.
- <*iterations*> specifies the number of times to perform a test.
- <*channel*> may contain only any single channel number on a Remote Signal Conditioning Unit. All channels on that RSCU will be tested.
- **Returned Value:**

Value	Meaning	Further Action
0	test passed	None
1	number of times test failed	Error information in FIFO buffer. See error codes below.
-1	Couldn't start remote self-test	Query the Error Queue (SYST:ERR?) See error messages starting on page 401

The data type for this returned value is **int16**.

- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

DIAGnostic:TEST:REMote:SELFtest?

DIAGnostic:TEST:REMote:SELFtest? (@<ch_list>) executes a self-test on a single Remote Signal Conditioning Unit connected through the HP E1539A SCP. An example of an RSCU is the HP E1529A Remote Strain Bridge Conditioning unit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- <*channel*> may contain only any single channel number on a Remote Signal Conditioning Unit. All channels on that RSCU will be tested.

- **Returned Value:**

Value	Meaning	Further Action
0	Self-test OK	None
≥1	Error during remote self-test	Test number of first failure. See comments for test information in FIFO.
-1	Couldn't start remote self-test	Query the Error Queue (SYST:ERR?) See error messages starting on page 401

The data type for this returned value is **int16**.

- Failure Information for +1 return: The FIFO buffer will contain pairs of values. The first value will be the test number that failed followed by the failing channel number. The following are descriptions of the various tests:
Test 1: This test alternates calibration source and short on all channels.
Expected values are less than ±45mV on channels 0,2,4,6,8,...30
and about 3.2 volts on channels 1,3,5,7,...31.
Test 2: This tests the calibration source setting on all channels.
Expected values are approximately 3.2 volts on all channels.
Test 3: This tests the calibration short setting on all channels.
Expected values are less than ±45mV on all channels.
Test 4: This tests a random channel list and wrap around of the list.
The channel list is channels 12, 7, 21, 14, and 10. The test supplies 8 triggers, so the expected final channel is 21. The voltages on those channels is expected to be: 3.2, 0.0, 0.0, 3.2, and 3.2 volts respectively. If a failure occurs, the channel number is reported -- NOTE that for the second pass (the wrap) of channels 12, 7, and 21, a failure is logged in the fifo by adding 32 to the channel number (i.e. if we were testing the E1529 at channel 10000, and the 7th trigger point had bad data, the failure would be logged as 10039).
Test 5: This tests the filter settings on each bank (of 8 channels).
The method of this test is to ensure that the approximate rise times increase as the filters are changed from 100 Hz to 10 Hz and then to 2 Hz. The list of possible error messages is shown below (NOTE that the 1xx prefix to the channel number denotes the first three digits that uniquely identify which E1529 is to be tested -- 100, 101, 108, 109, etc.):

1xx45 -- Channel 0: 10 Hz rise time not at least 2x that of 100 Hz.
1xx46 -- Channel 8: 10 Hz rise time not at least 2x that of 100 Hz.
1xx47 -- Channel 16: 10 Hz rise time not at least 2x that of 100 Hz.
1xx48 -- Channel 24: 10 Hz rise time not at least 2x that of 100 Hz.

1xx50 -- Channel 0: 2 Hz rise time not at least 5x that of 100 Hz.
1xx51 -- Channel 8: 2 Hz rise time not at least 5x that of 100 Hz.
1xx52 -- Channel 16: 2 Hz rise time not at least 5x that of 100 Hz.
1xx53 -- Channel 24: 2 Hz rise time not at least 5x that of 100 Hz.

1xx55 -- Channel 0: 2 Hz rise time not at least 2x that of 10 Hz.
1xx56 -- Channel 8: 2 Hz rise time not at least 2x that of 10 Hz.
1xx57 -- Channel 16: 2 Hz rise time not at least 2x that of 10 Hz.
1xx58 -- Channel 24: 2 Hz rise time not at least 2x that of 10 Hz.

The following errors are not likely to occur, but are possible:
1xx32 -- Channel 0 100 Hz rise time test took too long.
1xx33 -- Channel 8 100 Hz rise time test took too long.
1xx34 -- Channel 16 100 Hz rise time test took too long.
1xx35 -- Channel 24 100 Hz rise time test took too long.
1xx36 -- Channel 0 10 Hz rise time test took too long.
1xx37 -- Channel 8 10 Hz rise time test took too long.
1xx38 -- Channel 16 10 Hz rise time test took too long.
1xx39 -- Channel 24 10 Hz rise time test took too long.
1xx40 -- Channel 0 2 Hz rise time test took too long.
1xx41 -- Channel 8 2 Hz rise time test took too long.
1xx42 -- Channel 16 2 Hz rise time test took too long.
1xx43 -- Channel 24 2 Hz rise time test took too long.

- Failure Information for -1 return, Probable causes:

  a. Unable to communicate with HP E1529A (is cable connected?)
  b. Invalid channel number or multiple channels specified
  c. Not enough memory to allocate internal arrays to hold data
  d. HP E1422A is currently performing a calibration operation.
  e. HP E1422A is currently performing a measurement operation.

- **Related Commands:** *TST?, *CAL?, CAL:REMote?, SYST:ERR?

- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage DIAG:REM:TEST? (@10000:10900) *self-test 4 RSCs at chs 00, 01, 08, and 09*

DIAGnostic:VERSion?

DIAGnostic:VERSion? returns the version of the firmware currently loaded into Flash Memory. The version information includes manufacturer, model, serial number, firmware version and date.

Comments

- **Returned Value:** Examples of the response string format:
  HEWLETT-PACKARD,E1422,US34000478,A.04.00,Thu Aug 5 9:38:07 MDT 1994

- The data type is **string**.

- **Use VXIplug&play function:** hpe1422_revision_query(...)

Usage DIAG:VERS? *Returns version string as shown above*

FETCh?

Subsystem Syntax **FETCh?** returns readings stored in VME memory.

Comments

- This command is only available in systems using an HP E1405B or HP E1406A command module.
- FETCH? does not alter the readings stored in VME memory. Only the *RST or INIT… commands will clear the readings in VME memory.
- The format of readings returned is set using the FORMat[:DATA] command.
- **Returned Value:** REAL,32, REAL,64, and PACK,64, readings are returned in the IEEE-488.2-1987 Definite Length Arbitrary Block Data format. This data return format is explained in “Arbitrary Block Program and Response Data” on page 201. For REAL,32, readings are 4 bytes in length. For REAL 64, and PACK, 64, readings are 8 bytes in length.
- PACKed,64 returns the same values as REAL,64 except for Not-a-Number (NaN), IEEE +INF and IEEE -INF. The NaN, IEEE +INF and IEEE -INF values returned by PACKed,64 are in a form compatible with HP Workstation BASIC and HP BASIC/UX. Refer to the FORMat command for the actual values for NaN, +INF, and -INF.
- ASCii is the default format.
- ASCII readings are returned in the form ±1.234567E±123. For example 13.325 volts would be +1.3325000E+001. Each reading is followed by a comma (,). A line feed (LF) and End-Or-Identify (EOI) follow the last reading.
- **Related Commands:** MEMory Subsystem, FORMat[:DATA]
- ***RST Condition:** MEMORY:VME:ADDRESS 240000; MEMORY:VME:STATE OFF; MEMORY:VME:SIZE 0

Use Sequence

MEM:VME:ADDR #H300000	
MEM:VME:SIZE #H100000	*1M byte or 262144 readings*
MEM:VME:STAT ON	
°	
°*(set up E1422 for scanning)*	
°	
TRIG:SOUR IMM	*let unit trigger on INIT*
INIT	*program execution remains here until VME memory is full or the HP E1422 has stopped taking readings*
FORM REAL,64	*affects only the return of data*
FETCH?	

Note When using the MEM subsystem, the module must be triggered before executing the INIT command (as shown above) unless you are using an external trigger (EXT trigger). When using EXT trigger, the trigger can occur at any time.

FORMat

The FORMat subsystem provides commands to set and query the response data format of readings returned using the [SENSe:]DATA:FIFO:…? commands.

Subsystem Syntax

```
FORMat
    [:DATA]  <format>[,<size>]
    [:DATA]?
```

FORMat[:DATA]

FORMat[:DATA] <*format*>[,<*size*>] sets the format for data returned using the [SENSe:]DATA:FIFO:…?, [SENSe:]DATA:CVTable, and FETCh? commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
format	discrete (string)	REAL | ASCii | PACKed	none
size	numeric	for ASCii, 7 for REAL, 32 | 64 for PACKed, 64	none

Comments

- The REAL format is IEEE-754 Floating Point representation.
- REAL, 32 provides the highest data transfer performance since no format conversion step is placed between reading and returning the data. The default *size* for the REAL format is 32 bits. Also see DIAG:IEEE command.
- PACKed, 64 returns the same values as REAL, 64 except for Not-a-Number (NaN), IEEE +INF and IEEE -INF. The NaN, IEEE +INF and IEEE -INF values returned by PACKed,64 are in a form compatible with HP Workstation BASIC and HP BASIC/UX (see table on following page).
- REAL 32, REAL 64, and PACK 64, readings are returned in the IEEE-488.2-1987 Arbitrary Block Data format. The Block Data may be either Definite Length or Indefinite Length depending on the data query command executed. These data return formats are explained in “Arbitrary Block Program and Response Data” on page 201. For REAL 32, readings are 4 bytes in length (data type is **float32 array**). For REAL 64, and PACK, 64, readings are 8 bytes in length (data type is **float64 array**).
- ASCii is the default format. ASCII readings are returned in the form ±1.234567E±123. For example 13.325 volts would be +1.3325000E+001. Each reading is followed by a comma (,). A line feed (LF) and End-Or-Identify (EOI) follow the last reading (data type is **string array**).

Note *TST? leaves the instrument in its power-on reset state. This means that the ASC,7 data format is set even if you had it set to something else before executing *TST?. If you need to read the FIFO for test information, set the format after *TST? and before reading the FIFO.

- **Related Commands:** [SENSe:]DATA:FIFO:…?, [SENSe:]DATA:CVTable?, MEMory subsystem, and FETCh?, Also see how DIAG:IEEE can modify REAL,32 returned values.

- ***RST Condition:** ASCII, 7

- After *RST/Power-on, each channel location in the CVT contains the IEEE-754 value "Not-a-number" (NaN). Channel readings which are a positive overvoltage return IEEE +INF and a negative overvoltage return IEEE -INF. The NaN, +INF, and -INF values for each format are shown in the following table.

Format	IEEE Term	Value	Meaning
ASCii	+INF	+9.9E37	Positive Overload
	-INF	-9.9E37	Negative Overload
	NaN	+9.91E37	No Reading
REAL,32	+INF	$7F800000_{16}$	Positive Overload
	-INF	$FF800000_{16}$	Negative Overload
	NaN	$7FFFFFFF_{16}$	No Reading
REAL,64	+INF	$7FF000...00_{16}$	Positive Overload
	-INF	$FFF000..00_{16}$	Negative Overload
	NaN	$7FFFFF...FF_{16}$	No Reading
PACKed,64	+INF	47D2 9EAD 3677 $AF6F_{16}$ ($+9.0e37_{10}$)	Positive Overload
	-INF	C7D2 9EAD 3677 $AF6F_{16}$ ($-9.0e37_{10}$)	Negative Overload
	NaN	47D2 A37D CED4 6143_{16} ($+9.91e37_{10}$)	No Reading

Table 6-1. Data Formats

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

FORMAT REAL	*Set format to IEEE 32-bit Floating Point*
FORM REAL, 64	*Set format to IEEE 64-bit Floating Point*
FORMAT ASCII, 7	*Set format to 7-bit ASCII*

FORMat[:DATA]?

FORMat[:DATA]? returns the currently set response data format for readings.

Comments

- **Returned Value:** Returns REAL, +32 | REAL, +64 | PACK, +64 | ASC, +7. The data type is **string, int16**.
- **Related Commands:** FORMAT
- ***RST Condition:** ASCII, 7
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

FORMAT? *Returns REAL, +32 | REAL, +64 | PACK, +64 | ASC, +7*

INITiate

The INITiate command subsystem moves the HP E1422 from the Trigger Idle State to the Waiting For Trigger State. When initiated, the instrument is ready to receive one (:IMMediate) or more (depending on TRIG:COUNT) trigger events. On each trigger, the module will perform one control cycle which includes reading analog and digital input channels (Input Phase), executing all defined algorithms (Calculate Phase), and updating output channels (Output Phase). See the TRIGger subsystem to specify the trigger source and count.

Subsystem Syntax

```
INITiate
    [:IMMediate]
```

INITiate[:IMMediate]

INITiate[:IMMediate] changes the trigger system from the Idle state to the Wait For Trigger state. When triggered, one or more (depending on TRIGger:COUNt) trigger cycles occur and the instrument returns to the Trigger Idle state.

Comments

- INIT:IMM clears the FIFO and Current Value Table.
- If a trigger event is received before the instrument is Initiated, a -211 "Trigger ignored" error is generated.
- If another trigger event is received before the instrument has completed the current trigger cycle (measurement scan), the Questionable Data Status bit 9 is set and a +3012 "Trigger too fast" error is generated.
- Sending INIT while the system is still in the Wait for Trigger state (already INITiated) will cause an error -213,"Init ignored".
- Sending the ABORt command send the trigger system to the Trigger Idle state when the current input-calculate-output cycle is completed.
- If updates are pending, they are made prior to beginning the Input phase.
- **When Accepted:** Not while INITiated
- **Related Commands:** ABORt, CONFigure, TRIGger
- ***RST Condition:** Trigger system is in the Idle state.
- **Use VXIplug&play function:** hpe1422_initImm(...)

Usage

INIT	*Both versions same function*
INITIATE:IMMEDIATE	

INPut

The INPut subsystem controls configuration of programmable *input* Signal Conditioning Plug-Ons (SCPs).

Subsystem Syntax

INPut
:FILTer
[:LPASs]
:FREQuency <*cutoff_freq*>,(@<*ch_list*>)
:FREQuency? (@<*channel*>)
[:STATe] 1 | 0 | ON | OFF,(@<*channel*>)
[:STATe]? (@<*channel*>)
:GAIN <*chan_gain*>,(@<*ch_list*>)
:GAIN? (@<*channel*>)
:LOW <*wvolt_type*>,(@<*ch_list*>)
:LOW? (@<*channel*>)
:POLarity NORMal | INVerted,(@<*ch_list*>)
:POLarity? (@<*channel*>)

INPut:FILTer[:LPASs]:FREQuency

INPut:FILTer[:LPASs]:FREQuency <*cutoff_freq*>,(@<*ch_list*>) sets the cutoff frequency of the filter on the specified channels.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
cutoff_freq	numeric (float32) (string)	see comment | MIN | MAX	Hz
ch_list	channel list (string)	100 - 15731	none

Comments

- *cutoff_freq* may be specified in killoHertz (khz). A programmable Filter in either an SCP or a Remote Signal Conditioning unit (RSC) has a choice of several discrete cutoff frequencies. The cutoff frequency set will be the one closest to the value specified by *cutoff_freq*.
- Sending MAX for the *cutoff_freq* selects the SCP or RSC's highest cutoff frequency. Sending MIN for the *cutoff_freq* selects the SCP or RSC's lowest cutoff frequency. To disable filtering (the "pass through" mode), execute the INP:FILT:STATE OFF command.
- Sending a value greater than the SCP's highest cutoff frequency or less than the SCP's lowest cutoff frequency generates a -222 "Data out of range" error.
- **When Accepted:** Not while INITiated
- **Related Commands:** INP:FILT:FREQ?, INP:FILT:STAT ON | OFF

- ***RST Condition:** generally set to MIN. The HP E1529A is set to 10Hz.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

INP:FILT:FREQ 100,(@100:119)	*Set cutoff frequency of 100 Hz for first 20 channels*
INPUT:FILTER:FREQ 2,(@15622)	*Set cutoff frequency of 2 Hz for RSC channel 5622*

INPut:FILTer[:LPASs]:FREQuency?

INPut:FILTer[:LPASs]:FREQuency? (@<*channel*>) returns the cutoff frequency currently set for *channel*. Non-programmable SCP channels may be queried to determine their fixed cutoff frequency. If the channel is not on an input SCP, the query will return zero.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- *channel* must specify a single channel only.
- This command is for programmable filter SCPs only.
- **Returned Value:** Numeric value of Hz as set by the INP:FILT:FREQ command. The data type is **float32**.
- **When Accepted:** Not while INITiated
- **Related Commands:** INP:FILT:LPAS:FREQ, INP:FILT:STATE
- ***RST Condition:** generally set to MIN. The HP E1529A is set to 10Hz.
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

INPUT:FILTER:LPASS:FREQUENCY? (@155)	*Check cutoff freq on channel 55*
INP:FILT:FREQ? (@10024)	*Check cutoff freq on RSC channel 0024*

INPut:FILTer[:LPASs][:STATe]

INPut:FILTer[:LPASs][:STATe] <*enable*>,(@<*ch_list*>) enables or disables a programmable filter SCP or RSC channel. When disabled (*enable*=OFF), these channels are in their "pass through" mode and provide no filtering. When re-enabled (*enable*=ON), the SCP channel reverts to its previously programmed setting.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	1 | 0 | ON | OFF	none
ch_list	channel list (string)	100 - 15731	none

Comments

- If the SCP has not yet been programmed, ON enables the SCP's default cutoff frequency.
- **When Accepted:** Not while INITiated
- ***RST Condition:** ON
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

INPUT:FILTER:STATE ON,(@115,117)	*Channels 115 and 117 return to previously set (or default) cutoff frequency*
INP:FILT OFF,(@10000:1131)	*Set RSC channels 0000-0131 to "pass-through" state*

INPut:FILTer[:LPASs][:STATe]?

INPut:FILTer[LPASs][:STATe]? (@<*channel*>) returns the currently set state of filtering for the specified channel. If the channel is not on an input SCP or RSC, the query will return zero.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- **Returned Value:** Numeric value either 0 (off or "pass-through") or 1 (on). The data type is **int16**.
- *channel* must specify a single channel only.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

INPUT:FILTER:LPASS:STATE? (@115)	*Enter statement returns either 0 or 1*
INP:FILT? (@12424)	*check filter cut-off on RSC channel 2424*

INPut:GAIN

INPut:GAIN <*gain*>,(@<*ch_list*>) sets the channel gain on programmable amplifier SCP or RSCU.

Note An important thing to understand about input amplifier SCPs and RSCUs is that given a fixed input value at a channel, changes in channel gain do not change the value returned from that channel. The DSP chip (Digital Signal Processor) keeps track of SCP gain and A/D range amplifier settings, and "calculates" a value that reflects the signal level at the input terminal. The only time this in not true is when the SCP gain chosen would cause the output of the SCP amplifier to be too great for the selected A/D range. As an example; with SCP gain set to 64, an input signal greater than ±0.25 volts would cause an over-range reading even with the A/D set to its 16 volt range.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
gain	numeric (float32) discrete (string)	see comment | MIN | MAX	none
ch_list	channel list (string)	100 - 163	none

Comments

- A programmable amplifier SCPor RSC has a choice of several discrete gain settings. The gain set will be the one closest to the value specified by *gain*. Refer to your SCP manual for specific information on the SCP you are programming. Sending MAX will program the highest gain available with the SCP installed. Sending MIN will program the lowest gain.

- Sending a value for *gain* that is greater than the highest or less than the lowest setting allowable for the SCP will generate a -222 "Data out of range" error.

- **When Accepted:** Not while INITiated

- **Related Commands:** INP:GAIN?

- ***RST Condition:** gain set to MIN

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
INP:GAIN  8,(@100:119)        Set gain of 8 for first 20 channels
INPUT:GAIN  64,(@155)         Set gain of 64 for SCP channel 55
```

INPut:GAIN?

INPut:GAIN? (@<*channel*>) returns the gain currently set for *channel*. If the channel is not on an input SCP, the query will return zero.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- *channel* must specify a single channel only.
- If the channel specified does not have a programmable amplifier, INP:GAIN? will return the nominal as-designed gain for that channel.
- **Returned Value:** Numeric value as set by the INP:GAIN command. The data type is **float32.**
- **When Accepted:** Not while INITiated
- **Related Commands:** INP:GAIN
- ***RST Condition:** gain set to 1
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

INPUT:GAIN? (@105)	*Check gain on channel 5*
INP:GAIN? (@100)	*Check gain on channel 0*

INPut:LOW

INPut:LOW <*wvolt_type*>,(@<*ch_list*>) controls the connection of input LO at a Strain Bridge SCP channel specified by <*ch_list*>. LO can be connected to the Wagner Voltage tap for quarter or half bridge configurations, or disconnected for full bridges. Note the HP E1529A's Wagner Voltage connection is only controlled by the command "[SENSe:]STRain:BRIDge[:TYPE]" on page 316.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
wvolt_type	discrete (string)	FLOat | WVOLtage	none
ch_list	channel list (string)	100 - 163	none

Comments

- **Related Commands:** INP:LOW?
- ***RST Condition:** INP:LOW FLOAT (all Option 21 channels)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

INP:LOW WVOL (@100:103,116:119)	*connect LO of channels 0 through 3 and 16 through 19 to Wagner Ground.*

INPut:LOW?

INPut:LOW? (@<*channel*>) returns the LO input configuration for the channel specified by <*channel*>. This command is for strain SCPs only, not for HP E1529A.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- *channel* must specify a single channel only.
- **Returned Value:** Returns FLO or WV. The data type is **string**.
- **Related Commands:** INP:LOW
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

INP:LOW? (@103) — *enter statement will return either FLO or WV for channel 3*

INPut:POLarity

INPut:POLarity <*mode*>,<*ch_list*> sets logical input polarity on a digital SCP channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	discrete (string)	NORMal | INVerted	none
ch_list	string	100 - 163	none

Comments

- If the channels specified are on an SCP that doesn't support this function, an error will be generated. See your SCP's User's Manual to determine its capabilities.
- **Related Commands:** for output sense; SOURce:PULSe:POLarity
- ***RST Condition:** INP:POL NORM for all digital SCP channels.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

INP:POL INV,(@140:143) — *invert first 4 channels on SCP at SCP position 5. Channels 40 through 43*

INPut:POLarity?

INPut:POLarity? <*channel*> returns the logical input polarity on a digital SCP channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- <*channel*> must specify a single channel.
- If the channel specified is on an SCP that doesn't support this function, an error will be generated. See your SCP's User's Manual to determine its capabilities.
- **Returned Value:** returns "NORM" or "INV". The type is **string**.
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

MEASure

The MEASure susbystem provides convenient setup-and-execution for some pre-measurement strain operations.

Subsystem Syntax

```
MEASure
    :VOLTage
        :EXCitation  (@<ch_list>)
        :UNSTrained  (@<ch_list>)
```

MEASure:VOLTage:EXCitation?

MEASure:VOLTage:EXCitation? (@<*ch_list*>) This command automatically configures the HP E1422A to measure the bridge excitation voltage at each channel in <*ch_list*> and starts a measurement scan. 32 measurements are averaged for each channel, and the averaged values are stored internally for later use by the strain Engineering Unit Conversion process. The average of each channel's reading is also sent to the FIFO buffer in case you want them for your own conversion process. The command returns a single value which is the number of readings sent to the FIFO.

Note The maximum excitation voltage the HP E1422A can sense through the HP E1529A's excitation sense path is 16 volts (±8VDC centered about the Gnd terminal). If you supply higher excitation voltage through the HP E1529A, don't connect the excitation sense terminals.

Note that this command executes a measurement scan without executing any algorithms that might be defined.

The sequence of individual commands to approximate this operation is

TRIGger :COUNt 1	*one time through scan list*
ROUTe:SEQuence:DEFine (@<ch_list>)	*input the list of channels to measure*
SENSe:FUNCtion:VOLTage [<range>,](@<ch_list>)	*set measurement function to volts*
SENSe:STRain:EXCitation:STATE ON,(@<ch_list>)	*turn on excitation supplies*
SENSe:STRain:CONNect EXCite,(@<ch_list>)	*connect channel sense to excitation supply*
INIT	*start measurement scan*
SENSe:DATA:FIFO:COUNT?	*query for number of readings in FIFO*
enter statement here to return FIFO reading <count>	
SENSe:DATA:FIFO:PART? <count>	*read excitation values from the FIFO*
enter statement to for block data from FIFO	

next the excitation voltage values acquired above must be sent back to the HP E1422 by executing the following command once for each channel in <ch_list> above:

SENSe:STRain:EXCitation <voltage_value>,(@channel)

Notes

1. Unlike the MEAS:VOLT:EXC? command, the individual command sequence above cannot keep defined algorithms from running at INIT. Since algorithms can place values into the FIFO buffer, you will have to determine which FIFO values are the excitation voltages.
2. Remember that the MEAS:VOLT:EXC? command also provides the average of 32 measurements for each excitation value.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	channel list (string)	10000 - 15731	none

Comments

- This command is only for use on channels measured with the HP E1529A. If executed on channels connected to other strain SCPs, a 3007 "Invalid signal conditioning plug-on" error message will be generated
- This comand executes a measurement scan without running defined algorithms. This is to keep algorithms from placing values in the FIFO buffer.
- The measurement sample interval is 392µS
- Filter settings and states are not changed by this command.
- After completing the measurements, the instrument is re-configured to the same settings that existed before the command was executed.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENSe:STRain:EXCitation, MEAS:VOLT:UNST?
- ***RST Condition:** channel excitation voltage values are not affected by *RST. However, *RST changes the function for all analog input channels to Voltage. When you change a strain channel back to the strain function with a SENS:FUNC:STRAIN... command, the excitation voltage values for these channel will still be in effect. Of course loss of power will cause the excitation values to be lost.
- **Returned Value:** numeric, number of channel values in FIFO. The type is **int16**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

MEAS:VOLT:EXC? (@10000:10931)	*determine excitation voltage on 128 strain channels through HP E1539As in SCP positons 0 and 1*
SENS:DATA:FIFO:RESET	*As in most cases, since the values have been sent to the strain EU conversion, we won't need to see the individual excitation values.*

MEASure:VOLTage:UNSTrained?

MEASure:VOLTage:UNSTrained? (@<*ch_list*>) This command automatically configures the HP E1422A to measure the bridge output voltage at each channel in <ch_list> and initiates a measurement scan. 32 measurements are averaged for each channel, and the averaged values are stored for later use by the strain Engineering Units conversion process on these channels. The strain bridges must be unstrained during this time. The average of each channel's reading is also sent to the FIFO buffer in case you want to view them. The command returns a single value which is the number of readings sent to the FIFO.

Note that this command executes a measurement scan without executing any algorithms that might be defined.

The sequence of individual commands to approximate this operation is

TRIGger :COUNt 1	*one time through scan list*
ROUTe:SEQuence:DEFine (@<ch_list>)	*input the list of channels to measure*
SENSe:FUNCtion:VOLTage [<range>,](@<ch_list>)	*set measurement function to volts*
SENSe:STRain:EXCitation:STATE ON,(@<ch_list>)	*turn on excitation supplies*
SENSe:STRain:CONNect BRIDge,(@<ch_list>)	*connect channel sense to bridge output*
INIT	*start measurement scan*
SENSe:DATA:FIFO:COUNT?	*query for number of readings in FIFO*
enter statement here to return FIFO reading <count>	
SENSe:DATA:FIFO:PART? <count>	*read unstrained voltage readings from the FIFO buffer*
enter statement to for block data from FIFO	

next the unstrained voltage values acquired above must be sent back to the HP E1422 by executing the following command once for each channel in <ch_list> above:

SENSe:STRain:UNSTrained <voltage_value>,(@channel)

Notes

1. Unlike the MEAS:VOLT:UNST? command, the individual command sequence above cannot keep defined algorithms from running at INIT. Since algorithms can place values into the FIFO buffer, you will have to determine which FIFO values are the excitation voltages.
2. Remember that the MEAS:VOLT:UNST? command also provides the average of 32 measurements for each excitation value.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	channel list (string)	10000 - 15731	none

Comments

- This command is only for use on channels measured with the HP E1529A. If executed on channels connected to other strain SCPs, a 3007 "Invalid signal conditioning plug-on" error message will be generated
- This comand executes a measurement scan without running defined algorithms.

This is to keep algorithms from placing values in the FIFO buffer.

- The measurement sample interval is 392µS
- Filter settings and states are not changed by this command.
- Note also that shunt resistor source and state are left as currently programmed.
- After completing the measurements, the instrument is re-configured to the same settings that existed before the command was executed.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENSe:STRain:EXCitation, MEAS:VOLT:UNST?
- ***RST Condition:** channel unstrained values are not affected by *RST. However, *RST changes the function for all analog input channels to Voltage. When you change a strain channel back to the strain function with a SENS:FUNC:STRAIN... command, the unstrained values for these channel will still be in effect. Of course loss of power will cause the unstrained values to be lost.
- **Returned Value:** numeric, number of channel values in FIFO. The type is **int16**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

MEAS:VOLT:UNST? (@10000:10931)	*determine unstrained voltage on 128 strain channels through HP E1539As in SCP positons 0 and 1*
SENS:DATA:FIFO:RESET	*As in most cases, since the values have been sent to the strain EU conversion, we won't need to see the individual unstrained values.*

MEMory

The MEMory subsystem allows using VME memory as an additional reading storage buffer.

Subsystem Syntax

```
MEMory
    :VME
        :ADDRess  <A24_address>
        :ADDRess?
        :SIZE  <mem_size>
        :SIZE?
        :STATe  1 | 0 | ON | OFF
        :STATe?
```

Note This subsystem is only available in systems using an HP E1405B or HP E1406A command module.

Use Sequence

*RST	
MEM:VME:ADDR #H300000	
MEM:VME:SIZE #H100000	*1M byte or 262144 readings*
MEM:VME:STAT ON	
*	
*(set up E1422 for scanning)	
*	
TRIG:SOUR IMM	*let unit trigger on INIT*
INIT	
*OPC?	*program execution remains here until VME memory is full or the HP E1422 has stopped taking readings*
FORM REAL,64	*affects only the return of data*
FETCH?	*return data from VME memory*

Note When using the MEM subsystem, the module must be triggered before executing the INIT command (as shown above) unless you are using an external trigger (EXT trigger). When using EXT trigger, the trigger can occur at any time.

MEMory:VME:ADDRess

MEMory:VME:ADDRess <*A24_address*> sets the A24 address of the VME memory card to be used as additional reading storage.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
A24_address	numeric	valid A24 address	none

Comments

- This command is only available in systems using an HP E1405B or HP E1406A command module.
- The default (if MEM:VME:ADDR not executed) is 240000_{16}.
- *A24_address* may be specified in decimal, hex (#H), octal (#Q), or binary (#B).
- **Related Commands:** MEMory subsystem, FORMat, and FETCH?
- ***RST Condition:** VME memory address starts at 200000_{16}. When using an HP E1405/6 command module, the first HP E1422 occupies 200000_{16} - $23FFFF_{16}$.

Usage MEM:VME:ADDR #H400000 — *Set the address for the VME memory card to be used as reading storage*

MEMory:VME:ADDRess?

MEMory:VME:ADDRess? returns the address specified for the VME memory card used for reading storage.

Comments

- **Returned Value:** numeric.
- This command is only available in systems using an HP E1405B or HP E1406A command module.
- **Related Commands:** MEMory subsystem, , FORMat, and FETCH?

Usage MEM:VME:ADDR? — *Returns the address of the VME memory card.*

MEMory:VME:SIZE

MEMory:VME:SIZE <*mem_size*> Specifies the number of bytes of VME memory to allocate for additional reading storage.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mem_size	numeric	to limit of available VME memory	none

Comments

- This command is only available in systems using an HP E1405B or HP E1406A command module.

- *mem_size* may be specified in decimal, hex (#H), octal (#Q), or binary(#B).
- *mem_size* should be a multiple of four (4) to accommodate 32 bit readings.
- **Related Commands:** MEMory subsystem, FORMAT, and FETCH?
- ***RST Condition:** MEM:VME:SIZE 0

Usage MEM:VME:SIZE 32768 — *Allocate 32 Kbytes of VME memory to reading storage (8192 readings)*

MEMory:VME:SIZE?

MEMory:VME:SIZE? returns the amount (in bytes) of VME memory allocated to reading storage.

Comments

- This command is only available in systems using an HP E1405B or HP E1406A command module.
- **Returned Value:** Numeric.
- **Related Commands:** MEMory subsystem, and FETCH?

Usage MEM:VME:SIZE? — *Returns the number of bytes allocated to reading storage.*

MEMory:VME:STATe

MEMory:VME:STATe <*enable*> enables or disables use of the VME memory card as additional reading storage.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	1 | 0 | ON | OFF	none

Comments

- This command is only available in systems using an HP E1405B or HP E1406A command module.
- When the VME memory card is enabled, the INIT command does not terminate until data acquisition stops or VME memory is full.
- **Related Commands:** Memory subsystem, and FETCH?
- ***RST Condition:** MEM:VME:STAT OFF

Usage

MEMORY:VME:STATE ON — *enable VME card as reading storage*
MEM:VME:STAT 0 — *Disable VME card as reading storage*

MEMory:VME:STATe?

MEMory:VME:STATe? returned value of 0 indicates that VME reading storage is disabled. Returned value of 1 indicates VME memory is enabled.

Comments

- This command is only available in systems using an HP E1405B or HP E1406A command module.
- **Returned Value:** Numeric 1 or 0. data type **uint16**.
- **Related Commands:** MEMory subsystem, and FETCH?

Usage MEM:VME:STAT? *Returns 1 for enabled, 0 for disabled*

OUTPut

The OUTPut subsystem is involved in programming source SCPs as well as controlling the state of VXIbus TTLTRG lines 0 through 7.

Subsystem Syntax

```
OUTPut
    :CURRent
        :AMPLitude  <amplitude>,(@<ch_list>)
        :AMPLitude?  (@<channel>)
        [:STATe]  1 | 0 | ON | OFF,(@<ch_list>)
        [:STATe]?  (@<channel>)
    :POLarity  NORMal | INVerted,(@<ch_list>)
    :POLarity?  (@<channel>)
    :SHUNt  1 | 0 | ON | OFF,(@<ch_list>)
    :SHUNt?  (@<channel>)
        :SOURce  INT | EXT,(@<ch_list>)
        :SOURce?  (@<channel>)
    :SHUNt?  (@<channel>)
    :TTLTrg
        :SOURce  TRIGger | FTRigger | SCPlugon | LIMit
        :SOURce?
    :TTLTrg<n>
        [:STATe]  1 | 0 | ON | OFF
        [:STATe]?
    :TYPE  PASSive | ACTive,(@<ch_list>)
    :TYPE?  (@<channel>)
    :VOLTage
        :AMPLitude <amplitude>,(@<ch_list>)
        :AMPLitude? (@<channel>)
```

OUTPut:CURRent:AMPLitude

OUTPut:CURRent:AMPLitude <*amplitude*>,(@<*ch_list*>) sets the HP E1505 Current Source SCP channels specified by *ch_list* to either 488 µA, or 30 µA. This current is typically used for four-wire resistance and resistance temperature measurements.

Note This command does not set current amplitude on SCPs like the HP E1532 Current Output SCP.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
amplitude	numeric (float32)	MIN | 30E-6 | MAX | 488E-6	ADC
ch_list	channel list (string)	100 - 163	none

Comments

- Select 488E-6 (or MAX) for measuring resistances of less than 8000 Ohms. Select 30E-6 (or MIN) for resistances of 8000 Ohms and above. ***amplitude*** may be specified in µA (ua).
- For resistance temperature measurements ([SENSe:]FUNCtion:TEMPerature) the Current Source SCP must be set as follows:

MAX (488µA)	for RTD,85 | 92 and THER,2250
MIN (30µA)	for THER,5000 | 10000

- When *CAL? is executed, the current sources are calibrated on the range selected at that time.
- **When Accepted:** Not while INITiated
- **Related Commands:** *CAL?, OUTP:CURR:AMPL?
- ***RST Condition:** MIN
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

OUTP:CURR:AMPL 488ua,(@116:123)	*Set Current Source SCP at channels 16 through 23 to 488 µA*
OUTP:CURR:AMPL 30E-6,(@105)	*Set Current Source SCP at channel 5 to 30 µA*

OUTPut:CURRent:AMPLitude?

OUTPut:CURRent:AMPLitude? (@<*channel*>) returns the range setting of the Current Source SCP channel specified by *channel*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- *channel* must specify a single channel only.
- If *channel* specifies an SCP which is not a Current Source, a +3007, "Invalid signal conditioning plug-on" error is generated.

- **Returned Value:** Numeric value of amplitude set. The data type is **float32**.
- **Related Commands:** OUTP:CURR:AMPL
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage OUTP:CURR:AMPLITUDE? (@163) *Check SCP current set for channel 63 (returns +3.0E-5 or +4.88E-4)*

OUTPut:CURRent[:STATe]

OUTPut:CURRent[:STATe] <*enable*>,(@<*ch_list*>) enables or disables current source on channels specified in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	1 | 0 | ON | OFF	none
ch_list	channel list (string)	100 - 163	none

Comments

- OUTP:CURR:STAT does not affect a channel's amplitude setting. A channel that has been disabled, when re-enabled sources the same current set by the previous OUTP:CURR:AMPL command.
- OUTP:CURR:STAT is most commonly used to turn off excitation current to four-wire resistance (and resistance temperature device) circuits during execution of CAL:TARE for those channels.
- **When Accepted:** Not while INITiated
- **Related Commands:** OUTP:CURR:AMPL, CAL:TARE
- ***RST Condition:** OUTP:CURR OFF (all channels)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage OUTP:CURR OFF,(@100,108) *turn off current source channels 0 and 8*

OUTPut:CURRent[:STATe]?

OUTPut:CURRent[:STATe]? (@<*channel*>) returns the state of the Current Source SCP channel specified by <*channel*>. If the channel is not on an HP E1505 Current Source SCP, the query will return zero.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- *channel* must specify a single channel only.
- **Returned Value:** returns 1 for enabled, 0 for disabled. data type is **uint16**.
- **Related Commands:** OUTP:CURR:STATE, OUTP:CURR:AMPL
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

```
OUTP:CURR?  (@108)              query for state of Current SCP channel 8
execute enter statement here    enter query value, either 1 or 0
```

OUTPut:POLarity

OUTPut:POLarity <*select*>,(@<ch_list>) sets the polarity on digital output channels in <ch_list>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
select	discrete (string)	NORMal | INVerted	none
ch_list	string	100 - 163	none

Comments

- If the channels specified do not support this function, an error will be generated.
- Related Commands: INPut:POLarity, OUTPut:POLarity?
- ***RST Condition:** OUTP:POL NORM for all digital channels
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
OUTP:POL  INV,(@144)            invert output logic sense on channel 44
```

OUTPut:POLarity?

OUTPut:POLarity? (@<*channel*>) returns the polarity on the digital output channel in <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- *Channel* must specify a single channel
- **Returned Value:** returns one of NORM or INV. The type is **string**.
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

OUTPut:SHUNt

OUTPut:SHUNt <*enable*>,(@<*ch_list*>) adds shunt resistance to one leg of bridge on Strain Bridge Completion SCPs and the HP E1529A Remote Strain Bridge unit. This can be used for diagnostic purposes and characterization of bridge response.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	0 | 1 | ON | OFF	none
ch_list	channel list (string)	100 - 15731	none

Comments

- If *ch_list* specifies a non strain SCP, a 3007 "Invalid signal conditioning plug-on" error is generated.
- Only one channel on any one HP E1529A can be specified in <ch_list>. This is because a single resistor is used to shunt each of an HP E1529As 32 channels. The <ch_list> may specify one channel on each of several HP E1529As.
- **When Accepted:** Not while INITiated
- **Related Commands:** [SENSe:]FUNCtion:STRain…, [SENSe:]STRain…
- ***RST Condition:** OUTP:SHUNT 0 on all Strain SCP and RSC channels
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
OUTP:SHUNT  1,(@116:119)
```
add shunt resistance at channels 16 through 19

OUTPut:SHUNt?

OUTPut:SHUNt? (@<*channel*>) returns the status of the shunt resistance on the specified Strain SCP or RSC channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- channel must specify a single channel only.

- If *channel* specifies a non strain SCP or RSC, a 3007 "Invalid signal conditioning plug-on" error is generated.
- **Returned Value:** Returns 1 or 0. The data type is **uint16**.
- **Related Commands:** OUTP:SHUNT
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

OUTPUT:SHUNt? (@116)	*Check status of shunt resistance on channel 16*
OUTPUT:SHUNt? (@10124)	*Check status of shunt resistance on HP E1529A channel 0124*

OUTPut:SHUNt:SOURce

OUTPut:SHUNt:SOURce <*select*>,(@<*ch_list*>) selects the source of the bridge shunt resistance for a HP E1529A Remote Strain Bridge Conditioning unit. The HP E1529A has an internal shunt resistor, and also supports an external user supplied resistor.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
select	discrete (string)	INTernal | EXTernal	none
ch_list	channel list (string)	10000 - 15731	none

Comments

- If *ch_list* specifies a non HP E1529A strain SCP, a 3007 "Invalid signal conditioning plug-on" error is generated.
- Only one channel on each HP E1529A needs to be specified since a single resistor is used for all channels in the module.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENSe:STRain:SHUNted, SENSe:FUNCtion:STRain…, [SENSe:]STRain…
- ***RST Condition:** OUTP:SHUNT:SOURCE INT on all HP E1529A channels
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

OUTP:SHUNT:SOUR EXT,(@10000,10800)	*select user supplied shunt resistor on HP E1529s connected to channels 0,1,8,and 9*

OUTPut:SHUNt:SOURce?

OUTPut:SHUNt:SOURce? (@<*channel*>) returns the source of the shunt resistance on the specified HP E1529A Strain channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- channel must specify a single channel only, and since there is a single shunt resistor for all channels on an HP E1529A, it can be any channel on the HP E1529A.
- If *channel* specifies a non HP E1529A channel, a 3007 "Invalid signal conditioning plug-on" error is generated.
- **Returned Value:** Returns "INT" or "EXT". The data type is **string**.
- **Related Commands:** OUTP:SHUNT:SOUR
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

OUTPUT:SHUNT:SOURCE? (@11600) — *Check source of shunt resistance on HP E1529A connected to channel 16*

OUTPut:TTLTrg:SOURce

OUTPut:TTLTrg:SOURce <*trig_source*> selects the internal source of the trigger event that will operate the VXIbus TTLTRG lines.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
trig_source	discrete (string)	ALGorithm | TRIGger | FTRigger | SCPlugon	none

Comments

- The following table explains the possible choices.

ALGorithm	Generated by the Algorithm Language function "interrupt()"
FTRigger	Generated on the First Trigger of a multiple "counted scan" (set by TRIG:COUNT <*trig_count*>)
SCPlugon	Generated by a Signal Conditioning Plug-on (SCP). Do not use this when Sample-and-Hold SCPs are installed.
TRIGger	Generated every time a scan is triggered (see TRIG:SOUR <*trig_source*>)

- FTRigger (First TRigger) is used to generate a single TTLTRG output when repeated triggers are being used to make multiple executions of the enabled algorithms. The TTLTRG line will go low (asserted) at the first trigger event and stay low through subsequent triggers until the trigger count (as set by TRIG:COUNT) is exhausted. At this point the TTLTRG line will return to its high state (de-asserted). This feature can be used to signal when the HP E1422 has started running its control algorithms.
- **Related Commands:** OUTP:TTLT<n>[:STATE], OUTP:TTLT:SOUR?, TRIG:SOUR, TRIG:COUNT
- ***RST Condition:** OUTP:TTLT:SOUR TRIG
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage OUTP:TTLT:SOUR TRIG — *toggle TTLTRG line every time module is triggered (use to trigger other HP E1422s)*

OUTPut:TTLTrg:SOURce?

OUTPut:TTLTrg:SOURce? returns the current setting for the TTLTRG line source.

Comments

- **Returned Value:** Discrete, one of; TRIG, FTR, or SCP. The data type is **string**.
- **Related Commands:** OUTP:TTLT:SOUR
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage OUTP:TTLT:SOUR? — *enter statement will return on of FTR, SCP, or TRIG*

OUTPut:TTLTrg<n>[:STATe]

OUTPut:TTLTrg<n>:STATe <*ttltrg_cntrl*> specifies which VXIbus TTLTRG line is enabled to source a trigger signal when the module is triggered. TTLTrg<n> can specify line 0 through 7. For example, …:TTLTRG4, or TTLT4 for VXIbus TTLTRG line 4.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ttltrg_cntrl	boolean (uint16)	1 | 0 | ON | OFF	none

Comments

- Only one VXIbus TTLTRG line can be enabled simultaneously.
- **When Accepted:** Not while INITiated
- **Related Commands:** ABORT, INIT…, TRIG…

- ***RST Condition:** OUTPut:TTLTrg<0 through 7> OFF
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

OUTP:TTLT2 ON	*Enable TTLTRG2 line to source a trigger*
OUTPUT:TTLTRG7:STATE ON	*Enable TTLTRG7 line to source a trigger*

OUTPut:TTLTrg<n>[:STATe]?

OUTPut:TTLTrg<n>[:STATe]? returns the current state for TTLTRG line <n>.

Comments

- **Returned Value:** Returns 1 or 0. The data type is **int16**.
- **Related Commands:** OUTP:TTLT<n>
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

OUTP:TTLT2?	*See if TTLTRG2 line is enabled (returns 1 or 0)*
OUTPUT:TTLTRG7:STATE?	*See if TTLTRG7 line is enabled*

OUTPut:TYPE

OUTPut:TYPE <*select*>,(@<*ch_list*>) sets the output drive characteristic for digital SCPs with programmable channels.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
select	discrete (string)	PASSive | ACTive	seconds
ch_list	string	100 - 163	none

Comments

- If the channels specified are on an SCP that doesn't support this function an error will be generated. See your SCP's User's Manual to determine its capabilities.
- PASSive configures the digital channel/bit to be passive (resistor) pull-up to allow you to wire-or more than one output together.
- ACTive configures the digital channel/bit to both source and sink current.
- **Related Commands:** SOURce:PULSe:POLarity, OUTPut:TYPE?
- ***RST Condition:** OUTP:TYPE ACTIVE (for TTL compatibility)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

OUTP:TYPE PASS,(@140:143)	*make channels 40 to 43 passive pull-up*

OUTPut:TYPE?

OUTPut:TYPE? <*channel*> returns the output drive characteristic for a digital channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- *Channel* must specify a single channel.
- If the channel specified is not on a digital SCP, an error will be generated.
- **Returned Value:** returns PASS or ACT. The type is **string**.
- ***RST Condition:** returns ACT
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

OUTPut:VOLTage:AMPLitude

OUTPut:VOLTage:AMPLitude <*amplitude*>,(@<*ch_list*>) sets the excitation voltage on programmable Strain Bridge Completion SCPs pointed to by <*ch_list*> (the HP E1511 for example).

Note This command is not used to set output voltage on SCPs like the HP E1531 Voltage Output SCP.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
amplitude	numeric (float32)	MIN | 0 |1 | 2 | 5 | 10 | MAX	none
ch_list	channel list (string)	100 - 163	none

Comments

- To turn off excitation voltage (when using external voltage source) program *amplitude* to 0.
- **Related Commands:** OUTP:VOLT:AMPL?
- ***RST Condition:** MIN (0) for HP E1511A
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

OUTP:VOLT:AMPL 5,(@116:119) — *set excitation voltage for channels 16 through 19*

OUTPut:VOLTage:AMPLitude?

OUTPut:VOLTage:AMPLitude? (@<*channel*>) returns the current setting of excitation voltage for the channel specified by <*channel*>. If the channel is not on an HP E1511 SCP, the query will return zero.

Comments

- *channel* must specify a single channel only.
- **Returned Value:** Numeric, one of 0, 1, 2 ,5, or 10 for HP E1511A SCP, 3.9 for non programmable HP E1506/07 SCPs. data type is **float32**.
- **Related Commands:** OUTP:VOLT:AMPL
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

OUTP:VOLT:AMPL? (@103)	*returns current setting of excitation voltage for channel 3*

ROUTe

The ROUTe subsystem provides a method to define the sequence of channels in the HP E1422A's Analog Input scan list. Note that any analog input channels specified in an algorithm definition also affect the contents of this scan list. Queries are provided to determine the overall channel list definition including analog output channels as well as digital input and output channels.

Subsystem Syntax

```
ROUTe
    :SEQuence
        :DEFine  (@<ch_list>)
        :DEFine?
        :POINts?
```

ROUTe:SEQuence:DEFine

ROUTe:SEQuence:DEFine (@<*ch_list*>) adds channels to the analog input scan list. Channels specified with ROUT:SEQ:DEF will be scanned each time the HP E1422 receives a scan trigger. By default, the readings taken will be sent to the Current Value Table (CVT) and FIFO buffer. A special form of channel specifier allows changing the default data destinations (see comments below). Any algorithms you define can also add channels to the analog input scan list.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	channel list (string)	100 - 15731	none

Comments

- <*ch_list*> must contain analog input channels only.
- Channels specified by ROUT:SEQ:DEF are combined with any channels referenced in defined algorithms. Duplicate channel references are discarded. No matter how many times a channel is referenced, it is only measured once per trigger and the same value is seen in storage and by algorithms.

Note Certain analog input SCPs display higher than normal offset and noise figures if their channels are scanned just before channels on a Remote Signal Conditioning Unit. To avoid any such interraction, you should order your scan list so all remote channels (5-digit channel numbers) appear before any on-board channels (3-digit channel numbers)

- **Controlling Data Destination:**The relative form of the SCPI Channel List syntax is used to control the destination of data from channels in the scan list. For a discussion of the syntax See “Channel List (Relative Form)” on page 200 The value of the "Data Destination" digit controls the destination of the values read from the specified channels in the following manner:

Data Destination	Effect on Reading
1	Reading sent to Current Value Table (CVT)
2	Reading sent to FIFO Buffer
3	Reading sent to CVT and FIFO
0	Reading not recorded (neither CVT or FIFO)

Example Channel lists (same applies to On-board channels):

ROUT:SEQ:DEF (@1(10000:10931)) *1st 128 remote Chs sent to CVT*
ROUT:SEQ:DEF (@2(10000:10931)) *1st 128 remote Chs sent toFIFO buffer*
ROUT:SEQ:DEF (@3(10000:10931)) *1st 128 remote Chs sent to CVT & FIFO*
ROUT:SEQ:DEF (@0(10000:10931)) *1st 128 remote Chs discarded*

- **Relationship between Channel number and CVT location:** There is a fixed relationship between each possible channel number and the Current Value Table location that the channel reading is sent to.

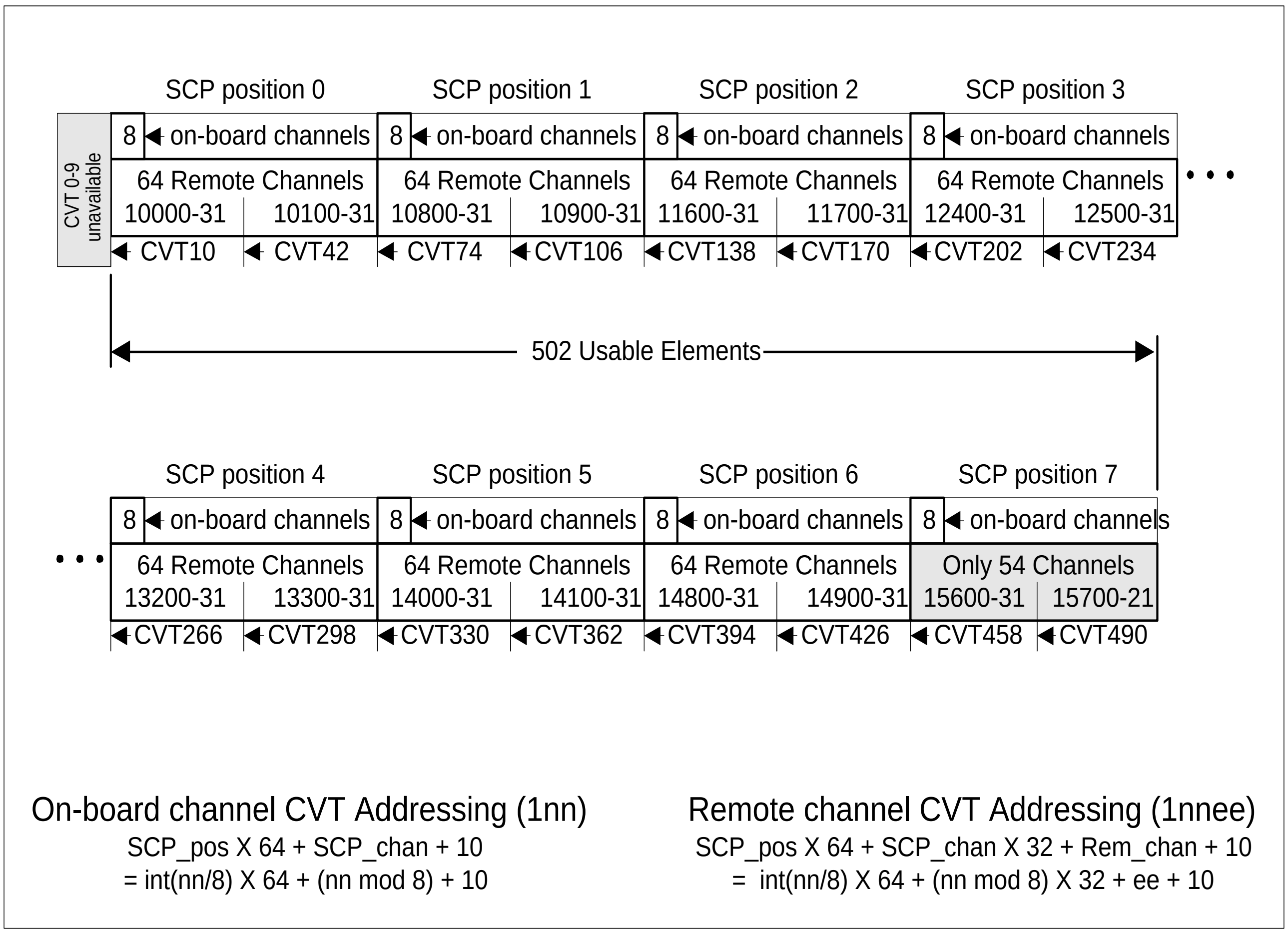


Figure 6-4. Channel Number vs. CVT Element

- Notice that since there are only 502 CVT elements available for up to 512 possible remote channel specifiers, these last 10 channels (15722-15731) must not be sent to the CVT or an error will be generated. Since the default data destination is to BOTH the FIFO and CVT (Data Destination 3), any reference in ROUT:SEQ:DEF to the last 10 remote channels must force the data destination to FIFO only.

- ***RST Condition:** To supply the necessary time delay before Digital inputs are read, the analog input (AIN) scan list contains two entries for channel 0 (100).This minimum delay is maintained by replacing these default channels as others are defined by algorithms or ROUT:SEQ:DEF. The three other lists contain no channels.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage *define mix of Remote and on-board channels. First 128 Remote channels default to FIFO and CVT, next 64 Remote channels are directed to FIFO only, and last 8 On-board channels default to FIFO and CVT.*

ROUT:SEQ:DEF (@10000:10931,2(11600:11731),124:131)

ROUTe:SEQuence:DEFine?

ROUTe:SEQuence:DEFine? <*type*> returns the sequence of channels defined in the scan list.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	(string)	AIN | AOUT | DEST | DIN | DOUT	none

Comments

- The channel list contents and sequence are determined primarily by channel references in the ROUT:SEQ:DEF command and in any algorithms currently defined. The SENS:REF:CHANNELS, and SENS:CHAN:SETTLING commands also effect the scan list contents.
- The <*type*> parameter selects which channel list will be queried:

  "AIN" selects the Analog Input channel list (this is the Scan List).
  "AOUT" selects the Analog Output channel list.
  "DIN" selects the Digital Input channel list.
  "DOUT" selects the Digital Output channel list.

  "DEST" does not requesting the contents of a channel list type, rather it requests the Data Destination number for each channel in the "AIN" channel list.

- **Returned Value:** Data type is an **int16 array**). Use ROUT:SEQ:POINTS? to determine how many values will be returned in the array. Can also be returned as type **string**, see later comment regarding VXIplug&play function.

  When the <*type*> parameter is "AIN", "AOUT", "DIN", or "DOUT", each value returned represents a channel number.

When the <*type*> parameter is "DEST", each value returned is the data destination code for a channel in the "AIN" list. The destination codes are"
1 = Reading will be sent to CVT only
2 = Reading will be sent to FIFO only
3 = Reading will be sent to CVT and FIFO
0 = Reading will be taken but not recorded (neither CVT or FIFO)
-1 = Channel is referenced in algorithm. Algorithm specifies destination.

- ***RST Condition:** To supply the necessary time delay before Digital inputs are read, the analog input (AIN) scan list contains two entries for channel 0 (100).This minimum delay is maintained by replacing these default channels as others are defined by algorithms or ROUT:SEQ:DEF. The three other lists contain no channels.

- **Send with VXIplug&play Function:** hpe1422_cmdString_q(...) to return a string which is a comma-separated-list of values

Usage ROUT:SEQ:DEF? AIN *query for analog input (Scan List) sequence*

ROUTe:SEQuence:POINts?

ROUTe:SEQuence:POINts? <*type*> returns the number of channels defined in the selected channel list types.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	(string)	AIN | AOUT | DIN | DOUT	none

Comments

- The channel list contents and sequence are determined by channel references in the algorithms currently defined.

- The <*type*> parameter selects which channel list will be queried:

  "AIN" selects the Analog Input list.
  "AOUT" selects the Analog Output list.
  "DIN" selects the Digital Input list.
  "DOUT" selects the Digital Output list.

  Note: "DEST" is not one of the choices because it is not a channel list type.

- **Returned Value:** Numeric. The C_SCPI type is **int16**.

- ***RST Condition:** The Analog Input list returns +8, the others return +0.

- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage ROUT:SEQ:POINTS? AIN *query for analog input channel count*

SAMPle

The SAMPle subsystem provides commands to set and query the interval between channel measurements (pacing).

Subsystem Syntax

SAMPle
    :TIMer <*interval*>
    :TIMer?

SAMPle:TIMer

SAMPle:TIMer <*interval*> sets the time interval between channel measurements. It is used to provide additional channel settling time. See “Settling Characteristics” on page 149

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
interval	numeric (float32) (string)	4.0E-5 to 16.3825E-3 | MIN | MAX	seconds

Comments

- The minimum *interval* is 40 µ seconds. The resolution for *interval* is 2.5 µsecond.
- If the Sample Timer interval multiplied by the number of channels in the specified Scan List is longer than the Trigger Timer interval, at run time a "Trigger too fast" error will be generated.
- the SAMP:TIMER interval can change the effect of the SENS:CHAN:SETTLING command. SENS:CHAN:SETT specifies the number of times a channel measurement should be repeated for channels defined in an algorithm. The total settling time per channel then is (SAMP:TIMER <*interval*>) X (<*chan_repeats*> from SENS:CHAN:SETT)
- **When Accepted:** Not while INITiated
- **Related Commands:** SENSE:CHAN:SETTLING, SAMP:TIMER?
- ***RST Condition:** Sample Timer for all Channel Lists set to 4.0E-5 seconds.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SAMPLE:TIMER 150E-6   *Pace measurements at 150µsecond intervals channel to channel*

SAMPle:TIMer?

SAMPle:TIMer? returns the sample timer interval.

Comments

- **Returned Value:** Numeric. The data type is **float32**.
- **Related Commands:** SAMP:TIMER
- ***RST Condition:** Sample Timer set to 4.0E-5 seconds.
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage SAMPLE:TIMER? *Check the interval between channel measurements*

[SENSe]

The SENSe subsystem controls conversion of the sensed electrical signal to a value in Engineering Units (EU) like volts, Ohms, temperature. Sense commands allow you to configure and extract data from the A/D-EU conversion portion of the instrument (see “INPut” subsystem on page 261 for input signal conditioning.

Subsystem Syntax

```
[SENSe:]
    :CHANnel
        :SETTling <settle_time>,(@<ch_list>)
        :SETTling? (@<channel>)
    DATA
        :CVTable? (@<element_list>)
            :RESet
        :FIFO
            [:ALL]?
            :COUNt?
                :HALF?
            :HALF?
            :MODE BLOCk | OVERwrite
            :MODE?
            :PART? <n_values>
            :RESet
    FREQuency:APERture <gate time>,<ch_list>
    FREQuency:APERture? <channel>
    FUNCtion
        :CONDition (@<ch_list>)
        :CUSTom [<range>,](@<ch_list>)
            :REFerence [<range>,](@<ch_list>)
            :TC <type>,[<range>,](@<ch_list>)
        :FREQuency (@<ch_list>)
        :RESistance <excite_current>,[<range>,](@<ch_list>)
        :STRain
            :FBENding [<range>,](@<ch_list>)
            :FBPoisson [<range>,](@<ch_list>)
            :FPOisson [<range>,](@<ch_list>)
            :HBENding [<range>,](@<ch_list>)
            :HPOisson [<range>,](@<ch_list>)
            [:QUARter] [<range>,](@<ch_list>)
            :Q120 [<range>,](@<ch_list>)
            :Q350 [<range>,](@<ch_list>)
            :USER [<range>,](@<ch_list>)
        :TEMPerature
<sensor_type>,<sub_type>,[<range>,](@<ch_list>)
        :TOTalize (@<ch_list>)
        :VOLTage[:DC] [<range>,](@<ch_list>)
    REFerence <sensor_type>, [<sub_type>,](@<ch_list>)
        :CHANnels (@<ref_channel>),(@<ch_list>)
        :TEMPerature <degrees_celsius>
```

```
STRain
    :BRIDge
        :TYPE FBEN | HBEN | Q120 | Q350 | USER,(@<ch_list>)
        :TYPE? (@<channel>)
CONNect  BRIDge | EXCitation,(@<ch_list>)
CONNect?  (@<channel>)
    :EXCitation <excite_v>,(@<ch_list>)
        :STATe  ON | OFF,(@<ch_list>)
        :STATe?  (@<channel>)
    :EXCitation?  (@<channel>)
    :GFACtor  <gage_factor>,(@<ch_list>)
    :GFACtor?  (@<channel>)
    :POISson  <poisson_ratio>,(@<ch_list>)
    :POISson?  (@<channel>)
    :UNSTrained <unstrained_v>,(@<ch_list>)
    :UNSTrained?  (@<channel>)
TOTalize:RESet:MODE  INIT | TRIGger,(@<ch_list>)
TOTalize:RESet:MODE?  (@<channel>)
```

[SENSe:]CHANnel:SETTling

[SENSe:]CHANnel:SETTling <*num_samples*>,<*ch_list*> specifies the number of measurement samples to make on channels in <*ch_list*> that are also referenced in currently defined algorithms. SENS:CHAN:SETTLING is used to provide additional settling time only to selected channels that might need it. See "Settling Characteristics" on page 149

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
settle_time	numeric (int16)	1 to 64	none
ch_list	string	100 - 15731	none

Comments

- SENS:CHAN:SETTLING causes each channel specified in <*ch_list*> that is also referenced in an algorithm to appear <*num_samples*> times in the analog input Scan List. Channels that do not appear in any SENS:CHAN:SETT command will be entered into the scan list only once when referenced in an algorithm.

- Since the scan list is limited to 512 entries, an error will be generated if the number of channels referenced in ROUT:SEQ:DEF, and algorithms plus the additional entries from any SENS:CHAN:SETTLING command exceeds 512. In addition, The scan list for any RSC is limited to 32 channels so if all channels are already specified in the scan list, no further references are allowed with the SENS:CHAN:SETT command for that RSC.

- The SAMPLE:TIMER command can change the effect of the SENS:CHAN:SETTLING command since SAMPLE:TIMER changes the amount of time for each measurement sample.

- **When Accepted:** Not while INITiated
- **Related Commands:** [SENSe:]CHANnel:SETTling?, SAMPLE:TIMER
- ***RST Condition:** SENS:CHAN:SETTLING 1,(@100:163)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SENS:CHAN:SETT 4,(@144,156) *settle channels 44 and 56 for 4 measurement periods*

[SENSe:]CHANnel:SETTling?

[SENSe:]CHANnel:SETTling? <*channel*> returns the current number of samples to make on <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 15731	none

Comments

- <*channel*> must specify a single channel.
- Related Commands: SENS:CHAN:SETT, SAMP:TIMER?
- ***RST Condition:** will return 1 for all channels.
- **Returned Value:** returns numeric number of samples, The type is **int16**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

[SENSe:]DATA:CVTable?

[SENSe:]DATA:CVTable? (@<*element_list*>) returns from the Current Value Table the most recent values stored by algorithms.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
element_list	channel list	10 - 511	none

Comments

- [SENSe:]DATA:CVTable? (@<*element_list*>) allows you to "view" the latest values from algorithms and/or analog scans.
- The Current Value Table is an area in memory that can contain as many as 502 32-bit floating point values. Algorithms can copy any of their variable values into these CVT elements while they execute. The algorithm statements to put data into the CVT are:

writecvt(<expr>, <element_number>), and
writeboth(<expr>, <element_number>).

There is a fixed relationship between channel number and CVT element for reading values from channels placed in the Scan List with ROUT:SEQ:DEF. When you are mixing Scan List data acquisition with algorithm data storage, be careful not to overwrite Scan List generated values with algorithm generated values. See “ROUTe:SEQuence:DEFine” on page 287 for controlling CVT entries from the analog scan list.

- Elements 0 through 9 are not accessible.
- The format of values returned is set using the FORMat[:DATA] command
- **Returned Value:** ASCII values are returned in the form ±1.234567E±123. For example 13.325 volts would be +1.3325000E+001. Each value is followed by a comma (,). A line feed (LF) and End-Or-Identify (EOI) follow the last value. The data type is a **string array**.

  REAL 32, REAL 64, and PACK 64, values are returned in the IEEE-488.2-1987 Definite Length Arbitrary Block Data format. This data return format is explained in “Arbitrary Block Program and Response Data” on page 201. For REAL 32, each value is 4 bytes in length (the data type is a **float32 array**). For REAL 64 and PACK 64, each value is 8 bytes in length (the data type is a **float64 array**).

Note After *RST/Power-on, each element in the CVT contains the IEEE-754 value "Not-a-number" (NaN). Elements specified in the DATA:CVT? command that have not been written to be an algorithm will return the value 9.91E37.

- ***RST Condition:** All elements of CVT contains IEEE-754 "Not a Number".
- **Use VXIplug&play function:** hpe1422_readCVT_Q(...)

Usage

SENS:DATA:CVT? (@10:511)	*Return all CVT values (502)*
SENS:DATA:CVT? (@30:38)	*Return 9 values*

[SENSe:]DATA:CVTable:RESet

[SENSe:]DATA:CVTable:RESet sets all 64 Current Value Table entries to the IEEE-754 "Not-a-number".

Comments

- The value of NaN is +9.910000E+037 (ASCII).
- Executing DATA:CVT:RES while the module is INITiated will generate an error 3000, "Illegal while initiated".

- **When Accepted:** Not while INITiated
- **Related Commands:** SENSE:DATA:CVT?
- ***RST Condition:** SENSE:DATA:CVT:RESET
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SENSE:DATA:CVT:RESET *Clear the Current Value Table*

[SENSe:]DATA:FIFO[:ALL]?

[SENSe:]DATA:FIFO[:ALL]? returns all values remaining in the FIFO buffer until all measurements are complete or until the number of values returned exceeds FIFO buffer size (65,024).

Comments

- DATA:FIFO? may be used to acquire all values (even while they are being made) into a single large buffer, or can be used after one or more DATA:FIFO:HALF? commands to return the remaining values from the FIFO.
- The format of values returned is set using the FORMat[:DATA] command.
- **Returned Value:** ASCII values are returned in the form ±1.234567E±123. For example 13.325 volts would be +1.3325000E+001. Each value is followed by a comma (,). A line feed (LF) and End-Or-Identify (EOI) follow the last value. The data type is a **string array**.

  REAL 32, REAL 64, and PACK 64, values are returned in the IEEE-488.2-1987 Indefinite Length Arbitrary Block Data format. This data return format is explained in “Arbitrary Block Program and Response Data” on page 201. For REAL 32, each value is 4 bytes in length (the data type is a **float32 array**). For REAL 64 and PACK 64, each value is 8 bytes in length (the data type is a **float64 array**).

Note Values which are a positive overvoltage return IEEE +INF and a negative overvoltage return IEEE -INF (see Table 6-1 on page 258 for actual values for each data format).

- **Related Commands:** SENSE:DATA:FIFO:HALF?, ROUT:SEQ:DEFine
- ***RST Condition:** FIFO is empty
- **Use VXIplug&play function:** hpe1422_readFifo_Q(...)

Usage DATA:FIFO? *return all FIFO values until measurements complete and FIFO empty*

Command Sequence

set up scan list/algorithms and trigger
SENSE:DATA:FIFO:ALL?
now execute read statement — *read statement does not complete until triggered measurements are complete and FIFO is empty*

[SENSe:]DATA:FIFO:COUNt?

[SENSe:]DATA:FIFO:COUNt? returns the number of values currently in the FIFO buffer.

Comments

- DATA:FIFO:COUNT? is used to determine the number of values to acquire with the DATA:FIFO:PART? command.
- **Returned Value:** Numeric 0 through 65,024. The data type is **int32**.
- **Related Commands:** DATA:FIFO:PART?
- ***RST Condition:** FIFO empty
- **Use VXIplug&play function:** hpe1422_sensDataFifoCoun_Q(...)
- **Send with VXIplug&play Function:** hpe1422_cmdInt32_Q(...)

Usage

DATA:FIFO:COUNT? — *Check the number of values in the FIFO buffer*

[SENSe:]DATA:FIFO:COUNt:HALF?

[SENSe:]DATA:FIFO:COUNt:HALF? returns a 1 if the FIFO is at least half full (contains at least 32,768 values), or 0 if FIFO is less than half-full.

Comments

- DATA:FIFO:COUNT:HALF? is used as a fast method to poll the FIFO for the half-full condition.
- **Returned Value:** Numeric 1 or 0. The data type is **int16**.
- **Related Commands:** DATA:FIFO:HALF?
- ***RST Condition:** FIFO empty
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Command Sequence

DATA:FIFO:COUNT:HALF? — *poll FIFO for half-full status*
DATA:FIFO:HALF? — *returns 32768 values*

[SENSe:]DATA:FIFO:HALF?

[SENSe:]DATA:FIFO:HALF? returns 32,768 values if the FIFO buffer is at least half-full. This command provides a fast means of acquiring blocks of values from the buffer.

Comments

- For acquiring data from continuous scans, your application needs to execute a DATA:FIFO:HALF? command and a read statement often enough to keep up with the rate that values are being sent to the FIFO.
- Use the DATA:FIFO:ALL? command to acquire the values remaining in the FIFO buffer after the ABORT command has stopped execution.
- The format of values returned is set using the FORMat[:DATA] command.
- **Returned Value:** ASCII values are returned in the form ±1.234567E±123. For example 13.325 volts would be +1.3325000E+001. Each value is followed by a comma (,). A line feed (LF) and End-Or-Identify (EOI) follow the last value. The data type is a **string array**.

  REAL 32, REAL 64, and PACK 64, values are returned in the IEEE-488.2-1987 Definite Length Arbitrary Block Data format. This data return format is explained in “Arbitrary Block Program and Response Data” on page 201. For REAL 32, each value is 4 bytes in length (the data type is a **float32 array**). For REAL 64 and PACK 64, each value is 8 bytes in length (the data type is a **float64 array**).

Note Values which are a positive overvoltage return IEEE +INF and a negative overvoltage return IEEE -INF (see Table 6-1 on page 258 for actual values for each data format).

- **Related Commands:** DATA:FIFO:COUNT:HALF?
- ***RST Condition:** FIFO buffer is empty
- **Send with VXIplug&play Function:** hpe1422_readFifoFast_Q(...)

Command Sequence

DATA:FIFO:COUNT:HALF?	*poll FIFO for half-full status*
DATA:FIFO:HALF?	*returns 32768 values*

[SENSe:]DATA:FIFO:MODE

[SENSe:]DATA:FIFO:MODE <*mode*> sets the mode of operation for the FIFO buffer.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	discrete (string)	BLOCk | OVERwrite	none

Comments

- In BLOCk(ing) mode, if the FIFO becomes full and measurements are still being made, the new values are discarded.
- OVERwrite mode is used record the latest 65,024 values. The module must be halted (ABORT sent) before attempting to read the FIFO. In OVERwrite Mode, if the FIFO becomes full and measurements are still being made, new values overwrite the oldest values.
- In both modes Error 3021, "FIFO Overflow" is generated to let you know that measurements have been lost.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENSE:DATA:FIFO:MODE?, SENSE:DATA:FIFO:ALL?, SENSE:DATA:FIFO:HALF?, SENSE:DATA:FIFO:PART?, SENSE:DATA:FIFO:COUNT?
- ***RST Condition:** SENSE:DATA:FIFO:MODE BLOCk
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENSE:DATA:FIFO:MODE OVERWRITE	*Set FIFO to overwrite mode*
DATA:FIFO:MODE BLOCK	*Set FIFO to block mode*

[SENSe:]DATA:FIFO:MODE?

[SENSe:]DATA:FIFO:MODE? returns the currently set FIFO mode.

Comments

- **Returned Value:** String value either BLOCK or OVERWRITE. The data type is **string**.
- **Related Commands:** SENSE:DATA:FIFO:MODE
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

DATA:FIFO:MODE?	*Enter statement returns either BLOCK or OVERWRITE*

[SENSe:]DATA:FIFO:PART?

[SENSe:]DATA:FIFO:PART? <*n_values*> returns *n_values* from the FIFO buffer.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n_values	numeric (int32)	1 - 2,147,483,647	none

Comments

- Use the DATA:FIFO:COUNT? command to determine the number of values in the FIFO buffer.

- The format of values returned is set using the FORMat[:DATA] command.
- **Returned Value:** ASCII values are returned in the form ±1.234567E±123. For example 13.325 volts would be +1.3325000E+001. Each value is followed by a comma (,). A line feed (LF) and End-Or-Identify (EOI) follow the last value. The data type is a **string array**.

  REAL 32, REAL 64, and PACK 64, values are returned in the IEEE-488.2-1987 Definite Length Arbitrary Block Data format. This data return format is explained in “Arbitrary Block Program and Response Data” on page 201. For REAL 32, each value is 4 bytes in length (the data type is a **float32 array**). For REAL 64 and PACK 64, each value is 8 bytes in length (the data type is a **float64 array**).

Note Values which are a positive overvoltage return IEEE +INF and a negative overvoltage return IEEE -INF (see Table 6-1 on page 258 for actual values for each data format).

- **Related Commands:** DATA:FIFO:COUNT?
- ***RST Condition:** FIFO buffer empty
- **Use VXIplug&play function:** hpe1422_readFifoFast_Q(...)

Usage DATA:FIFO:PART? 256 *return 256 values from FIFO*

[SENSe:]DATA:FIFO:RESet

[SENSe:]DATA:FIFO:RESet clears the FIFO of values. The FIFO counter is reset to 0.

Comments

- **When Accepted:** Not while INITiated
- **Related Commands:** SENSE:DATA:FIFO…
- ***RST Condition:** SENSE:DATA:FIFO:RESET
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SENSE:DATA:FIFO:RESET *Clear the FIFO*

[SENSe:]FREQuency:APERture

[SENSe:]FREQuency:APERture <*gate_time*>,<*ch_list*> sets the gate time for frequency measurement. The gate time is the time period that the SCP will allow for counting signal transitions in order to calculate frequency.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
gate_time	numeric (float32)	.001 to 1 (.001 resolution)	seconds
ch_list	string	100 - 163	none

Comments

- If the channels specified are on an SCP that doesn't support this function, an error will be generated. See your SCP's User's Manual for its capabilities.
- Related Commands: SENSe:FUNCtion:FREQuency
- ***RST Condition:** .001 sec
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENS:FREQ:APER .01,(@144) *set channel 44 aperture to 10msec*

[SENSe:]FREQuency:APERture?

[SENSe:]FREQuency:APERture? <*channel*> returns the frequency counting gate time.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- If the channel specified is on an SCP that doesn't support this function, an error will be generated. See your SCP's User's Manual for its capabilities.
- Related Commands: SENSe:FREQuency:APERture
- **Returned Value:** returns numeric gate time in seconds, The type is **float32**.
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

[SENSe:]FUNCtion:CONDition

[SENSe:]FUNCtion:CONDition <*ch_list*> sets the SENSe function to input the digital state for channels in <*ch_list*>. Also configures digital SCP channels as inputs (this is the *RST condition for all digital I/O channels).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	string	100 - 163	none

Comments

- The HP E1533 SCP senses 8 digital bits on each channel specified by this command. The HP E1534 SCP senses 1 digital bit on each channel specified by this command.
- If the channels specified are not on a digital SCP, an error will be generated.
- Use the INPut:POLarity command to set input logical sense.
- **Related Commands:** INPut:POLarity
- ***RST Condition:** SENS:FUNC:COND and INP:POL NORM for all digital SCP channels.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage To set second 8-bits of HP E1533 at SCP position 4, and upper 4-bits of HP E1534 at SCP position 5 to digital inputs send:

```
SENS:FUNC:COND  (@133,144:147)
```

[SENSe:]FUNCtion:CUSTom

[SENSe:]FUNCtion:CUSTom [<*range*>,](@<*ch_list*>) links channels with the custom Engineering Unit Conversion table loaded with the DIAG:CUST:MXB or DIAG:CUST:PIECE commands. Contact your Hewlett-Packard System Engineer for more information on Custom Piecewise Engineering Unit Conversion for your application.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see first comment	VDC
ch_list	channel list (string)	100 - 15731	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 142
- <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 causes an error -222 "Data out of range". Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If you are using amplifier SCPs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input

out-of-range condition will exist.

- If an A/D reading is greater than the <*table_range*> specified with DIAG:CUSTOM:PIEC, an overrange condition will occur.
- If no custom table has been loaded for the channels specified with SENS:FUNC:CUST, an error will be generated when an INIT command is given.
- **When Accepted:** Not while INITiated
- **Related Commands:** DIAG:CUST:…
- ***RST Condition:** all custom EU tables erased
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

program must put table constants into array table_block
DIAG:CUST:MXB slope,offset,(@116:123) *send table to HP E1422 for chs 16-23*
SENS:FUNC:CUST 1,(@116:123) *link custom EU with chs 16-23*
INITiate then TRIGger module

[SENSe:]FUNCtion:CUSTom:REFerence

[SENSe:]FUNCtion:CUSTom:REFerence [<*range*>,](@<*ch_list*>) links channels with the custom Engineering Unit Conversion table loaded with the DIAG:CUST:PIECE command. Measurements from a channel linked with SENS:FUNC:CUST:REF will result in a temperature that is sent to the Reference Temperature Register. This command is used to measure the temperature of an isothermal reference panel using custom characterized RTDs or thermistors.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see comments	VDC
ch_list	channel list (string)	100 - 163	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 142
- The <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If you are using amplifier SCPs, you should set them first and keep their settings

in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.

- The *CAL? command calibrates temperature channels based on Sense Amplifier SCP setup at the time of execution. If SCP settings are changed, those channels are no longer calibrated. *CAL? must be executed again.
- **Related Commands:** DIAG:CUST:PIEC, SENS:FUNC:TEMP, SENS:FUNC:CUST:TC, *CAL?
- ***RST Condition:** all custom EU tables erased
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

program must put table constants into array table_block
DIAG:CUST:PIEC 1,table_block,(@108) *send characterized reference transducer table for use by channel 8*
SENS:FUNC:CUST:REF .25,(@108) *link custom ref temp EU with ch 8*
include this channel in a scan list with thermocouple channels (REF channel first)
INITiate then TRIGger module

[SENSe:]FUNCtion:CUSTom:TCouple

[SENSe:]FUNCtion:CUSTom:TCouple <*type*>,[<*range*>,](@<*ch_list*>) links channels with the custom Engineering Unit Conversion table loaded with the DIAG:CUST:PIECE command. The table is assumed to be for a thermocouple and the <*type*> parameter will specify the built-in compensation voltage table to be used for reference junction temperature compensation. SENS:FUNC:CUST:TC allows you to use an EU table that is custom matched to thermocouple wire you have characterized. Contact your Hewlett-Packard System Engineer for more information on Custom Piecewise Engineering Unit Conversion for your application.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	discrete (string)	E | EEXT | J | K | N | R | S | T	none
range	numeric (float32)	see comments	VDC
ch_list	channel list (string)	100 - 163	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 142.
- The <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO

selects auto range. The default range (no range parameter specified) is auto range.

- If you are using amplifier SCPs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.

- The *sub_type* EEXTended applies to E type thermocouples at 800°C and above.

- The *CAL? command calibrates temperature channels based on Sense Amplifier SCP setup at the time of execution. If SCP settings are changed, those channels are no longer calibrated. *CAL? must be executed again.

- **Related Commands:** DIAG:CUST:PIEC, *CAL?,SENS:REF, and SENS:REF:TEMP

- ***RST Condition:** all custom EU tables erased

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
program must put table constants into array table_block
DIAG:CUST:PIEC 1,table_block,(@100:107)  send characterized thermocouple table
                                         for use by channels 0-7
SENS:FUNC:CUST:TC N,.25,(@100:107)       link custom thermocouple EU with chs
                                         0-7, use reference temperature
                                         compensation for N type wire.
SENSE:REF  RTD,92,(@120)                 designate a channel to measure the
                                         reference junction temperature
include these channels in a scan list (REF channel first)
INITiate then TRIGger module
```

[SENSe:]FUNCtion:FREQuency

[SENSe:]FUNCtion:FREQuency <*ch_list*> sets the SENSe function to frequency for channels in <*ch_list*>. Also configures the channels specified as digital inputs.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	string	100 - 163	none

Comments

- If the channels specified are on an SCP that doesn't support this function, an error will be generated. See your SCP's User's Manual for its capabilities.

- Use the SENSe:FREQuency:APERture command to set the gate time for the frequency measurement.

- **Related commands:** SENS:FREQ:APER

- ***RST Condition:** SENS:FUNC:COND and INP:POL NORM for all digital SCP channels

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SENS:FUNC:FREQ (@144) *set channel 44's to frequency*

[SENSe:]FUNCtion:RESistance

[SENSe:]FUNCtion:RESistance <*excite_current*>,[<*range*>,](@<*ch_list*>) links the EU conversion type for resistance and range with the channels specified by *ch_list*.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
excite_current	discrete(string)	30E-6 | 488E-6 | MIN | MAX	Amps
range	numeric (float32)	see first comment	VDC
ch_list	channel list (string)	100 - 163	none

Comments

- The <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If you are using amplifier SCPs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.

- Resistance measurements require the use of Current Source Signal Conditioning Plug-Ons.

- The *excite_current* parameter (excitation current) does <u>not</u> control the current applied to the channel to be measured. The *excite_current* parameter only passes the setting of the SCP supplying current to channel to be measured. The current must have already been set using the OUTPUT:CURRENT:AMPL command. The choices for *excite_current* are 30E-6 (or MIN) and 488E-6 (or MAX). *excite_current* may be specified in milliamps (ma) and microamps (ua).

- The *CAL? command calibrates resistance channels based on Current Source SCP and Sense Amplifier SCP setup at the time of execution. If SCP settings are changed, those channels are no longer calibrated. *CAL? must be executed again.

- See “Linking Input Channels to EU Conversion” on page 102
- **When Accepted:** Not while INITiated
- **Related Commands:** OUTP:CURR, *CAL?
- ***RST Condition:** SENSE:FUNC:VOLT (@100:163)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage FUNC:RES 30ua,(@100,105,107) *Set channels 0, 5, and 7 to convert voltage to resistance assuming current source set to 30 µA use auto-range (default)*

[SENSe:]FUNCtion:STRain:FBENding
[SENSe:]FUNCtion:STRain:FBPoisson
[SENSe:]FUNCtion:STRain:FPOisson
[SENSe:]FUNCtion:STRain:HBENding
[SENSe:]FUNCtion:STRain:HPOisson
[SENSe:]FUNCtion:STRain[:QUARter]
[SENSe:]FUNCtion:STRain:Q120
[SENSe:]FUNCtion:STRain:Q350
[SENSe:]FUNCtion:STRain:USER

[SENSe:]FUNCtion:STRain:FBENding [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:FBPoisson [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:FPOisson [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:HBENding [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:HPOisson [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain[:QUARter] [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:Q120 [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:Q350 [<*range*>,](@<*ch_list*>)
[SENSe:]FUNCtion:STRain:USER [<*range*>,](@<*ch_list*>)

A Note on Syntax: Although the strain function is comprised of nine separate SCPI commands, their syntax and function is so similar they are discussed in a single reference entry.

[SENSe:]FUNCtion:STRain:<bridge_type> [<range>,](@<ch_list>) links the strain EU conversion with the channels specified by ch_list to measure the strain bridge output. See “Linking Input Channels to EU Conversion” on page 102

Note When the SENS:FUNC:STR:<bridge_type> command is used with HP E1529A channels, the bridge configuration switches for those channels are set to actually configure the bridge type specified. There is no need to send the configuration only SENSe:STRain:BRIDge:TYPE command for HP E1529A channels that use the SENSe:FUNCtion:STRain:<bridge_type> command.

Some of the SENS:STR:FUNC:<bridge_type> commands are used for both strain bridge completion SCPs and the HP E1529A while some are exclusive to one or the other.

The following table relates the command syntax to bridge type. See your Strain SCP user's manual for bridge schematics and field wiring information.

Command	Bridge Type	Strain SCP and HP E1529 Usage
:FBENding	Full Bending Bridge	Both HP E1529A and SCPs
:FBPoisson	Full Bending Poisson Bridge	SCPs only
:FPOisson	Full Poisson Bridge	SCPs only
:HBENding	Half Bending Bridge	Both HP E1529A and SCPs
:HPOisson	Half Poisson Bridge	SCPs only
[:QUARter]	Quarter Bridge (default)	Both HP E1529A and SCPs. For HP E1529A, selects Q350
:Q120	Quarter Bridge 120Ω	HP E1529A only
:Q350	Quarter Bridge 350Ω	HP E1529A only
:USER	Quarter Bridge with user installed resistor	HP E1529A only

Note Because of the number of possible strain gage configurations, the driver must generate any Strain EU conversion tables and download them to the instrument when INITiate is executed. This can cause the time to complete the first INIT command to exceed 1 minute on some platforms, notably the HP E1405A/E1406A. Subsequent INITs (with no other configuration changes) do not need to regenerate EU tables and execute much faster.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see comments	VDC
ch_list	channel list (string)	100 - 15731	none

Comments

- Strain measurements require the use of Bridge Completion Signal Conditioning Plug-Ons or a Remote Strain Bridge Conditioning Unit.
- Bridge Completion SCPs and RSCs provide the strain measurement bridges and their excitation voltage sources. *ch_list* specifies the voltage sensing channels that are to measure the bridge outputs. Measuring channels on a Bridge Completion SCP only returns that SCP's excitation source voltage.
- The <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC,

1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If you are using amplifier SCPs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.
- The HP E1529A has a fixed gain of 32. Keep this in mind when you set <range>.
- The channel calibration command (*CAL?) calibrates the excitation voltage source on each Bridge Completion SCP.
- **When Accepted:** Not while INITiated
- **Related Commands:** *CAL?, [SENSE:]STRAIN…
- ***RST Condition:** SENSE:FUNC:VOLT 0,(@100:163)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
SENS:FUNC:STRAIN:QUAR  1,(@100,105,107)   quarter bridge conversion for
                                          channels 0, 5, and 7
FUNC:STRAIN:HBEN  1,(@10800:10931)        full bridge conversion for E1529A
                                          channels 0800 to 0931 (64 channels)
```

[SENSe:]FUNCtion:TEMPerature

[SENSe:]FUNCtion:TEMPerature <*type*>,<*sub_type*>,[<*range*>,](@<*ch_list*>) links channels to an EU conversion for temperature based on the sensor specified in *type* and *sub_type*. **Not for sensing thermocouple reference temperature (for that, use the SENS:REF <*type*>,<*sub_type*>,(@<*channel*>) command)**.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	discrete (string)	RTD | THERmistor | TCouple	none
sub_type	numeric (float32) numeric (float32) discrete (string)	for RTD use 85 | 92 for THER use 2250 | 5000 | 10000 for TC use CUSTom | E | EEXT | J | K | N | R | S | T	none Ohms none
range	numeric (float32)	see comments	VDC
ch_list	channel list (string)	100 - 163	none

Comments

- Resistance temperature measurements (RTDs and THERmistors) require the use of Current Source Signal Conditioning Plug-Ons. The following table shows the Current Source setting that must be used for the following RTDs and Thermistors:

MAX (488µA)	for RTD and THER,2250
MIN (30µA)	for THER,5000 and THER,10000

- The <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If you are using amplifier SCPs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.

- The ***sub_type*** **parameter:** values of 85 and 92 differentiate between 100 Ohm (@ 0°C) RTDs with temperature coefficients of 0.00385 and and 0.00392 Ohm/Ohm/°C respectively. The *sub_type* values of 2250, 5000, and 10000 refer to thermistors that match the Omega 44000 series temperature response curve. These 44000 series thermistors are selected to match the curve within 0.1 or 0.2°C. For thermistors sub_type may be specified in Kohms (kohm).

  The *sub_type* EEXTended applies to E type thermocouples at 800°C and above.

  CUSTom is pre-defined as Type K, with no reference junction compensation (reference junction assumed to be at 0 °C).

- The *CAL? command calibrates temperature channels based on Current Source SCP and Sense Amplifier SCP setup at the time of execution. If SCP settings are changed, those channels are no longer calibrated. *CAL? must be executed again.

- See “Linking Input Channels to EU Conversion” on page 102

- **When Accepted:** Not while INITiated

- **Related Commands:** *CAL?, OUTP:CURR (for RTDs and Thermistors), SENS:REF, and SENS:REF:TEMP (for Thermocouples)

- ***RST Condition:** SENSE:FUNC:VOLT AUTO,(@100:163)

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

Link two channels to the K type thermocouple temperature conversion
SENS:FUNC:TEMP TCOUPLE,K,(@101,102)
Link channel 0 to measure reference temperature using 5K thermistor
SENS:REF THER,5000,(@100)

[SENSe:]FUNCtion:TOTalize

[SENSe:]FUNCtion:TOTalize <*ch_list*> sets the SENSe function to TOTalize for channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	string	100 - 163	none

Comments

- The totalize function counts rising edges of digital transitions at Frequency/Totalize SCP channels. The counter is 24 bits wide and can count up to 16,777,215.
- The SENS:TOT:RESET:MODE command controls which events will reset the counter.
- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- **Related Commands:** SENS:TOT:RESET:MODE, INPUT:POLARITY
- ***RST Condition:** SENS:FUNC:COND and INP:POL NORM for all digital SCP channels.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SENS:FUNC:TOT (@134) *channel 34 is a totalizer*

[SENSe:]FUNCtion:VOLTage[:DC]

[SENSe:]FUNCtion:VOLTage[:DC] [<*range*>,](@<*ch_list*>) links the specified channels to return DC voltage.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see comments	VDC
ch_list	channel list (string)	100 - 15731	none

Comments

- The <range> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If you are using amplifier SCPs or RSCs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.
- The *CAL? command calibrates channels based on Sense Amplifier SCP setup at the time of execution. If SCP settings are changed, those channels are no longer calibrated. *CAL? must be executed again.
- See “Linking Input Channels to EU Conversion” on page 102
- **When Accepted:** Not while INITiated
- **Related Commands:** *CAL?, INPUT:GAIN…
- ***RST Condition:** SENSE:FUNC:VOLT AUTO,(@100:163)
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage FUNC:VOLT (@140:163) *Channels 40 - 63 measure voltage in auto-range (defaulted)*

[SENSe:]REFerence

[SENSe:]REFerence <*type*>,<*sub_type*>,[<*range*>,](@<*ch_list*>) links channel in <*ch_list*> to the reference junction temperature EU conversion based on *type* and *sub_type*. When scanned, the resultant value is stored in the Reference Temperature Register, and by default the FIFO and CVT. This is a resistance temperature measurement and uses the on-board 122 µA current source.

Note The reference junction temperature value generated by scanning the reference channel is stored in the Reference Temperature Register. This reference temperature is used to compensate all subsequent thermocouple measurements until the register is overwritten by another reference measurement or by specifying a constant reference temperature with the SENSE:REF:TEMP command. If used, the reference junction channel must be scanned before any thermocouple channels. Use the SENSE:REF:CHANNELS command to place the reference measuring channel into the scan list ahead of the thermocouple measuring channels.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	discrete (string)	THERmistor | RTD | CUSTom	none
sub_type	numeric (float32) numeric (float32)	for THER use 5000 for RTD use 85 | 92 for CUSTom use 1	Ohm none none
range	numeric (float32)	see comments	VDC
ch_list	channel list (string)	100 - 15731	none

Comments

- See “Linking Input Channels to EU Conversion” on page 102
- The <*range*> parameter: The HP E1422 has five ranges: .0625VDC, .25VDC, 1VDC, 4VDC, and 16VDC. To select a range, simply specify the range value (for example, 4 selects the 4VDC range). If you specify a value larger than one of the first four ranges, the HP E1422 selects the next higher range (for example, 4.1 selects the 16VDC range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (.0625VDC). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If you are using amplifier SCPs or RSCs, you should set them first and keep their settings in mind when specifying a range setting. For instance, if your expected signal voltage is to be approximately .1VDC and the amplifier SCP for that channel has a gain of 8, you must set <*range*> no lower than 1VDC or an input out-of-range condition will exist.
- The <*type*> parameter specifies the sensor type that will be used to determine the temperature of the isothermal reference panel. <*type*> CUSTom is pre-defined as Type E with 0°C reference junction temp and is not re-defineable.
- For <*type*> THERmistor, the **<***sub_type*> parameter may be specified in ohms or kohm.
- The *CAL? command calibrates resistance channels based on Current Source SCP and Sense Amplifier SCP setup at the time of execution. If SCP settings are changed, those channels are no longer calibrated. *CAL? must be executed again.
- **Related Commands:** SENSE:FUNC:TEMP
- ***RST Condition:** Reference temperature is 0 °C
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENSE:REF RTD,92,(@120)	*sense the reference temperature on channel 20 using an RTD*
SENSE:REF THR,5000,(@13231)	*sense the reference temperature on RSC channel 3231 using a 5K thermistor*

[SENSe:]REFerence:CHANnels

[SENSe:]REFerence:CHANnels (@<*ref_channel*>),(@<*ch_list*>) causes channel specified by <*ref_channel*> to appear in the scan list just before the channel(s) specified by <*ch_list*>. This command is used to include the thermocouple reference temperature channel in the scan list before other thermocouple channels are measured.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ref_channel	channel list (string)	100 - 15731	none
ch_list	channel list (string)	100 - 15731	none

Comments

- Use SENS:FUNC:TEMP to configure channels to measure thermocouples. Then use SENS:REF to configure one or more channels to measure an isothermal reference temperature. Now use SENS:REF:CHAN to group the reference channel with its thermocouple measurement channels in the scan list.
- If thermocouple measurements are made through more than one isothermal reference panel, you will set up a reference channel for each. Execute the SENS:REF:CHAN command for each reference/measurement channel group.
- **Related commands:** SENS:FUNC:TEMP, SENS:REF
- ***RST Condition:** Scan List contains no channel references.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
SENS:FUNC:TEMP  TC,E,.0625,(@108:115)    E type TCs on channels 8 through 15
SENS:REF  THER,5000,1,(@106)             Reference ch is thermistor at channel 6
SENS:REF  RTD,85,.25,(@107)              Reference ch is RTD at channel 7
SENS:REF:CHAN (@106),(@108:111)          Thermistor measured before chs 8 - 11
SENS:REF:CHAN (@107),(@112:115)          RTD measured before chs 12 - 15
```

[SENSe:]REFerence:TEMPerature

[SENSe:]REFerence:TEMPerature <*degrees_c*> stores a fixed reference junction temperature in the Reference Temperature Register. Use when the thermocouple reference junction is kept at a controlled temperature.

Note

This reference temperature is used to compensate all subsequent thermocouple measurements until the register is overwritten by another SENSE:REF:TEMP value or by scanning a channel linked with the SENSE:REFERENCE command. If used, SENS:REF:TEMP must be executed before scanning any thermocouple channels.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
degrees_c	numeric (float32)	-126 to +126	none

Comments

- This command is used to specify to the HP E1422 the temperature of a controlled temperature thermocouple reference junction.
- **When Accepted:** Not while INITiated
- **Related Commands:** FUNC:TEMP TC…
- ***RST Condition:** Reference temperature is 0 °C
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENSE:REF:TEMP 40 — *subsequent thermocouple conversion will assume compensation junction at 40 degrees C*

[SENSe:]STRain:BRIDge[:TYPE]

[SENSe:]STRain:BRIDge[:TYPE] <select>,(@<*ch_list*>) sets the HP E1529A's bridge configuration switches for channels specified by <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
select	discrete (string)	FBENding | HBENding | Q120 | Q350 | USER	none
ch_list	channel list (string)	10000 - 15731	none

Comments

- For a discription of the effects of <*select*> see "HP E1529A Bridge Configurations" on page 67
- **Related Commands:** SENSE:FUNC:…
- ***RST Condition:** SENS:STR:BRIDG[:TYPE] FBEN for all HP E1529A channels
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENS:STRAIN:BRID Q120,(@1000:10031) — *configure strain RSC unit channels 00-31 connected to on-board channel 00 to 120 Ohm quarter bridge*

[SENSe:]STRain:BRIDge:[TYPE]?

[SENSe:]STRain:BRIDge[:TYPE]? (@<*channel*>) returns the HP E1529A's bridge configuration for channel specified by <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- <channel> must be a single channel only.
- **Returned Value:** one of "FBEN" | "HBEN" | "Q120" | "Q350" | "USER". The data type is **string**.
- **Related Commands:** SENSE:STR:BRID[:TYPE]
- ***RST Condition:** SENS:STR:BRID:TYPE FBEN for all HP E1529A channels
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

SENS:STRAIN:BRID (@10022) *check strain RSC unit channel 22 bridge configuration connected to on-board channel 00*

[SENSe:]STRain:CONNect

[SENSe:]STRain:CONNect <select>,(@<*ch_list*>) connects the HP E1529A channels specified by <*ch_list*> to sense either the strain bridge output or the bridge excitation supply. Only one channel for each HP E1529A needs to be specified in <*ch_list*> and all channels on that unit will configure as specified in <*select*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
select	discrete (string)	BRIDge | EXCitation	none
ch_list	channel list (string)	10000 - 15731	none

Comments

- **Related Commands:** SENSE:STRAIN:…, SENSE:FUNC:STRAIN…
- ***RST Condition:** SENS:STR:CONN BRIDGE for all HP E1529A channels
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENS:STRAIN:CONN EXC,(@10000:10031) *configure strain RSC unit channels 00-31 connected to on-board channel 00 to measure excitation voltages*

[SENSe:]STRain:CONNect?

[SENSe:]STRain:CONNect? (@<*channel*>) returns the measurement connection state for the single HP E1529A channel specified by <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- <*channel*> must specify a single channel only.
- **Returned Value:** one of "BRID" or "EXC". The data type is **string**.
- **Related Commands:** SENSE:STR:CONN, SENSE:STRAIN:…, SENSE:FUNC:STRAIN…
- ***RST Condition:** SENS:STR:CONN is BRIDGE for all HP E1529A channels
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage

SENS:STRAIN:CONN? (@10022) *check the measurement connection for strain RSC unit channel 22 connected to on-board channel 00*

[SENSe:]STRain:EXCitation

[SENSe:]STRain:EXCitation <*excite_v*>,(@<*ch_list*>) specifies the excitation voltage value to be used in the strain EU conversion for the channels specified by <*ch_list*>. The value used is usually measured at each strain bridge's excitation point. For the HP E1529A, the MEAS:VOLT:EXCitation command will make the measurements and automatically send the value to each measured channel's EU conversion.This command does not control the output voltage of any source.

Note The maximum excitation voltage the HP E1422A can sense through the HP E1529A's excitation sense path is 16 volts (±8VDC centered about the Gnd terminal). If you supply higher excitation voltage through the HP E1529A, don't connect the excitation sense terminals.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
excite_v	numeric (float32)	.01 - 99	volts
ch_list	channel list (string)	100 - 15731	none

Comments

- <*ch_list*> must specify the channel used to sense the bridge voltage, **not** the channel position on a Bridge Completion SCP.

- **Related Commands:** SENSE:STRAIN:…, SENSE:FUNC:STRAIN…, MEAS:VOLT:EXCitation
- ***RST Condition:** 3.9V
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

STRAIN:EXC Meas_excV,(@107)	*set measured excitation voltage for channel 7*
STRAIN:EXC Meas_excV,(@10022)	*set excitation voltage for E1529A channel 0022*

[SENSe:]STRain:EXCitation?

[SENSe:]STRain:EXCitation? (@<*channel*>) returns the excitation voltage value currently set for the sense channel specified by <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- <*channel*> must specify a single channel only.
- **Returned Value:** Numeric value of excitation voltage. The data type is **float32**.
- **Related Commands:** SENS:STRAIN:EXCitation, MEAS:VOLT:EXCitation
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

STRAIN:EXC? (@107)	*query excitation voltage for channel 7*
enter statement here	*returns the excitation voltage set by STR:EXC*

[SENSe:]STRain:EXCitation:STATe

[SENSe:]STRain:EXCitation:STATe <enable>,(@<*ch_list*>) connects or disconnects all four excitation supply ports on an HP E1529A. Each supply port powers a block of eight strain channels. Only one channel for each HP E1529A needs to be specified in <*ch_list*> to turn on all four of its excitation supplies. The first channel number for each possible HP E1529A is:

10000, 10100, 10800, 10900, 11600, 11700, 12400, 12500,
13200, 13300, 14000, 14100, 14800, 14900, 15600, 15700

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	ON | OFF	none
ch_list	channel list (string)	10000 - 15731	none

Comments

- **Related Commands:** SENSE:STRAIN:…, SENSE:FUNC:STRAIN…
- ***RST Condition:** OFF
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

STRAIN:EXC:STAT ON,(@10000:10016) — *turn on excitation supplies to channels 0-23 on HP E1529 connected to on-board channel 00*

[SENSe:]STRain:EXCitation:STATe?

[SENSe:]STRain:EXCitation:STATe? (@<*channel*>) returns the state of the HP E1529A excitation supply port referenced in <*ch_list*>. Each HP E1529A excitation port supplies a block of eight channels. The first HP E1529A channel number in each block is 00, 08, 16, and 24.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	1000 - 15731	none

Comments

- **Related Commands:** SENSE:STRAIN:EXC:STAT
- **Returned Value:** Numeric, 0 or 1. Type is **uint16**.
- ***RST Condition:** OFF
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

STRAIN:EXC:STAT? (@10016) — *check state of excitation supply port to cHannel 16 on HP E1529 connected to on-board channel 00*

[SENSe:]STRain:GFACtor

[SENSe:]STRain:GFACtor <*gage_factor*>,(@<*ch_list*>) specifies the gage factor to be used to convert strain bridge readings for the channels specified by <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
gage_factor	numeric (float32)	1 - 5	none
ch_list	channel list (string)	100 - 15731	none

Comments

- <*ch_list*> must specify the SCP or RSC channel used to sense the bridge voltage, **not** the channel position on a Bridge Completion SCP.
- **Related Commands:** SENSE:STRAIN:GFAC?, SENSE:FUNC:STRAIN…
- ***RST Condition:** Gage factor is 2
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENS:STRAIN:GFAC 3,(@100:107)	*set gage factor for channels 0 through 7*
SENS:STRAIN:GFAC 2.2,(@10000:10931)	*set gage factor for HP E1529A channels 0000 through 0931 (128 channels)*

[SENSe:]STRain:GFACtor?

[SENSe:]STRain:GFACtor? (@<*channel*>) returns the gage factor currently set for the sense channel specified by <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- **Returned Value:** Numeric value of gage factor. The data type is **float32**.
- <*channel*> must specify a single channel only.
- **Related Commands:** STRAIN:GFACTOR
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

STRAIN:GFAC? (@107)	*query gage factor for channel 7*
enter statement here	*returns the gage factor set by STR:GFAC*

[SENSe:]STRain:POISson

[SENSe:]STRain:POISson <*poisson_ratio*>,(@<*ch_list*>) sets the Poisson ratio to be used for EU conversion of values measured on sense channels specified by <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
poisson_ratio	numeric (float32)	.1 - .5	none
ch_list	channel list (string)	100 - 163	none

Comments

- <*ch_list*> must specify channels used to sense strain bridge output, **not** channel positions on a Bridge Completion SCP.
- **Related Commands:** FUNC:STRAIN…, STRAIN:POISson?
- ***RST Condition:** Poisson ratio is .3
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

STRAIN:POISSON .5,(@124:131) — *set Poisson ratio for sense channels 24 through 31*

[SENSe:]STRain:POISson?

[SENSe:]STRain:POISson? (@<*channel*>) returns the Poisson ratio currently set for the sense channel specified by <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- **Returned Value:** numeric value of the Poisson ratio. The data type is **float32**.
- <*channel*> must specify a single channel only.
- **Related Commands:** FUNC:STRAIN…, STRAIN:POISSON
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

STRAIN:POISSON? (@131) — *query for the Poisson ratio specified for sense channel 31*

enter statement here — *enter the Poisson ratio value*

[SENSe:]STRain:UNSTrained

[SENSe:]STRain:UNSTrained <*unstrained_v*>,(@<*ch_list*>) specifies the unstrained voltage value to be used to convert strain bridge readings for the channels specified by <*ch_list*>. The HP E1529A can use the MEAS:VOLT:UNSTrained command which automatically measures the unstrained bridge values and sends each value to the channels' EU conversion. This command does not control the output voltage of any source.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unstrained_v	numeric (float32)	-16 through +16	volts
ch_list	channel list (string)	100 - 15731	none

Comments

- Use a voltage measurement of the unstrained strain bridge sense channel to determine the correct value for *unstrained_v*.
- <*ch_list*> must specify the channel used to sense the bridge voltage, **not** the channel position on a Bridge Completion SCP.
- **Related Commands:** SENSE:STRAIN:UNST?, SENSE:FUNC:STRAIN…, MEAS:VOLT:UNSTRained
- ***RST Condition:** Unstrained voltage is zero
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage STRAIN:UNST .024,(@100) *set unstrained voltage for channel 0*

[SENSe:]STRain:UNSTrained?

[SENSe:]STRain:UNSTrained? (@<*channel*>) returns the unstrained voltage value currently used for EU conversion for the sense channel specified by <*channel*>. This command does not make a measurement.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- **Returned Value:** Numeric value of unstrained voltage. The data type is **float32**.
- <*channel*> must specify a single channel only.
- **Related Commands:** STRAIN:UNST
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

STRAIN:UNST? (@107)	*query unstrained voltage for channel 7*
enter statement here	*returns the unstrained voltage set by STR:UNST*

[SENSe:]TOTalize:RESet:MODE

[SENSe:]TOTalize:RESet:MODE <*select*>,<*ch_list*> sets the mode for resetting totalizer channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
select	discrete (string)	INIT | TRIGger	seconds
ch_list	string	100 - 163	none

Comments

- In the INIT mode the total is reset only when the INITiate command is executed. In the TRIGger mode the total is reset every time a new scan is triggered.
- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- **Related Commands:** SENS:FUNC:TOT, INPUT:POLARITY
- ***RST Condition:** SENS:TOT:RESET:MODE INIT
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

SENS:TOT:RESET:MODE TRIG,(@134)   *totalizer at channel 34 resets at each trigger event*

[SENSe:]TOTalize:RESet:MODE?

[SENSe:]TOTalize:RESet:MODE? <*channel*> returns the reset mode for the totalizer channel in <channel>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- *Channel* must specify a single channel.
- If the channel specified is not on a frequency/totalize SCP, an error will be generated.
- **Returned Value:** returns INIT or TRIG. The type is **string**.
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

SOURce

The SOURce command subsystem allows configuring output SCPs as well as linking channels to output functions.

Subsystem Syntax

```
SOURce
    :FM
        :STATe  1 | 0 | ON | OFF,(@<ch_list>)
        :STATe?  (@<channel>)
    :FUNCtion
        [:SHAPe]
            :CONDition  (@<ch_list>)
            :PULSe  (@<ch_list>)
            :SQUare  (@<ch_list>)
    :PULM
        :STATe  1 | 0 | ON | OFF,(@<ch_list>)
        :STATe?  (@<channel>)
    :PULSe
        :PERiod  <period>,(@<ch_list>)
        :PERiod?  (@<channel>)
        :WIDTh  <pulse_width>,(@<ch_list>)
        :WIDTh?  (@<channel>)
```

SOURce:FM[:STATe]

SOURce:FM[:STATe] <*enable*>,(@<*ch_list*>) enables the Frequency Modulated mode for a PULSe channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	1 | 0 | ON | OFF	none
ch_list	string	100 - 163	none

Comments

- This command is coupled with the SOURce:PULM:STATE command. If the FM state is ON then the PULM state is OFF. If the PULM state is ON then the FM state is OFF. If both the FM and the PULM states are OFF then the PULSe channel is in the single pulse mode.
- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- Use SOURce:FUNCtion[:SHAPe]:SQUare to set FM pulse train to 50% duty cycle. Use SOURce:PULSe:PERiod to set the period
- ***RST Condition:** SOUR:FM:STATE OFF, SOUR:PULM:STATE OFF,

SENS:FUNC:COND and INP:POL for all digital SCP channels

- **Related Commands:** SOUR:PULM[:STATe], SOUR:PULS:POLarity, SOUR:PULS:PERiod, SOUR:FUNC[:SHAPe]:SQUare

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage The variable frequency control for this channel is provided by the algorithm language. When the algorithm executes an assignment statement to this channel, the value assigned will be the frequency setting. For example:

O143 = 2000 /* set channel 43 to 2KHz */

SOURce:FM:STATe?

SOURce:FM:STATe? (@<*channel*>) returns the frequency modulated mode state for a PULSe channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- *Channel* must specify a single channel.

- If the channel specified is not on a Frequency/Totalize SCP, an error will be generated.

- **Returned Value:** returns 1 (ON) or 0 (OFF). The type is **uint16**.

- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

SOURce:FUNCtion[:SHAPe]:CONDition

SOURce:FUNCtion[:SHAPe]:CONDition (@<*ch_list*>) sets the SOURce function to output digital patterns to bits in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	string	100 - 163	none

Comments

- The HP E1533 SCP sources 8 digital bits on the channel specified by this command. The HP E1534 SCP can source 1 digital bit on each of the the channels specified by this command.

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

SOURce:FUNCtion[:SHAPe]:PULSe

SOURce:FUNCtion[:SHAPe]:PULSe (@<*ch_list*>) sets the SOURce function to PULSe for the channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	string	100 - 163	none

Comments

- This PULSe channel function is further defined by the SOURce:FM:STATe and SOURce:PULM:STATe commands. If the FM state is enabled then the frequency modulated mode is active. If the PULM state is enabled then the pulse width modulated mode is active. If both the FM and the PULM states are disabled then the PULSe channel is in the single pulse mode.
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

SOURce:FUNCtion[:SHAPe]:SQUare

SOURce:FUNCtion[:SHAPe]:SQUare (@<*ch_list*>) sets the SOURce function to output a square wave (50% duty cycle) on the channels in <ch_list>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ch_list	string	100 - 163	none

Comments

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

The frequency control for these channels is provided by the algorithm language function:.

```
O143 = 2000 /* set channel 43 to 2KHz */
```

SOURce:PULM[:STATe]

SOURce:PULM[:STATe] <*enable*>,(@<*ch_list*>) enable the pulse width modulated mode for the PULSe channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable	boolean (uint16)	1 | 0 | ON | OFF	none
ch_list	string	100 - 163	none

Comments

- This command is coupled with the SOURce:FM command. If the FM state is enabled then the PULM state is disabled. If the PULM state is enabled then the FM state is disabled. If both the FM and the PULM states are disabled then the PULSe channel is in the single pulse mode.
- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- ***RST Condition:** SOUR:PULM:STATE OFF
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

SOURce:PULM:STATe?

SOURce:PULM[:STATe]? (@<*channel*>) returns the pulse width modulated mode state for the PULSe channel in <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments *Channel* must specify a single channel.

- **Returned Value:** returns 1 (on) or 0 (off). The type is **int16**.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

SOURce:PULSe:PERiod

SOURce:PULSe:PERiod <*period*>,(@<*ch_list*>) sets the fixed pulse period value on a pulse width modulated pulse channel. This sets the frequency (1/period) of the pulse-width-modulated pulse train.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
period	numeric (float32)	25E-6 to 7.8125E-3 (resolution 0.238µsec)	seconds
ch_list	string	100 - 163	none

Comments

- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- ***RST Condition:** SOUR:FM:STATE OFF and SOUR:PULM:STATE OFF
- **Related Commands:** SOUR:PULM:STATE, SOUR:PULS:POLarity

- The variable pulse-width control for this channel is provided by the algorithm language. When the algorithm executes an assignment statement to this channel, the value assigned will be the pulse-width setting. For example:

```
O140 = .0025 /* set channel 43 pulse-width to 2.5 msec */
```

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SOUR:PULS:PER .005,(@140) *set PWM pulse train to 200 Hz on channel 40*

SOURce:PULSe:PERiod?

SOURce:PULSe:PERiod? (@<*channel*>) returns the fixed pulse period value on the pulse width modulated pulse channel in <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- **Returned Value:** numeric period. The type is **float32**.
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

SOURce:PULSe:WIDTh

SOURce:PULSe:WIDTh <*pulse_width*>,(@<*ch_list*>) sets the fixed pulse width value on the frequency modulated pulse channels in <*ch_list*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
pulse_width	numeric (float32)	7.87E-6 to 7.8125E-3 (238.4E-9 resolution)	seconds
ch_list	string	100 - 163	none

Comments

- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- ***RST Condition:** SOUR:FM:STATE OFF and SOUR:PULM:STATE OFF
- **Related Commands:** SOUR:PULM:STATE, SOUR:PULS:POLarity

- The variable frequency control for this channel is provided by the algorithm language. When the algorithm executes an assignment statement to this channel, the value assigned will be the frequency setting. For example:

```
O143 = 2000 /* set channel 43 to 2KHz */
```

- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage SOUR:PULS:WIDTH 2.50E-3,(@143) *set fixed pulse width of 2.5 msec on channel 43*

SOURce:PULSe:WIDTh?

SOURce:PULSe:WIDTh? (@<*ch_list*>) returns the fixed pulse width value on a frequency modulated pulse channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	string	100 - 163	none

Comments

- *Channel* must specify a single channel.
- If the channels specified are not on a Frequency/Totalize SCP, an error will be generated.
- **Returned Value:** returns the numeric pulse width. The type is **float32**.
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

STATus

The STATus subsystem communicates with the SCPI defined Operation and Questionable Data status register sets. Each is comprised of a Condition register, a set of Positive and Negative Transition Filter registers, an Event register, and an Enable register. Condition registers allow you to view the current real-time states of their status signal inputs (signal states are not latched). The Positive and Negative Transition Filter registers allow you to control the polarity of change from the Condition registers that will set Event register bits. Event registers contain latched representations of signal transition events from their Condition register. Querying an Event register reads and then clears its contents, making it ready to record further event transitions from its Condition register. Enable registers are used to select which signals from an Event register will be logically ORed together to form a summary bit in the Status Byte Summary register. Setting a bit to one in an Enable register enables the corresponding bit from its Event register.

Note For a complete discussion See "Using the Status System" on page 134

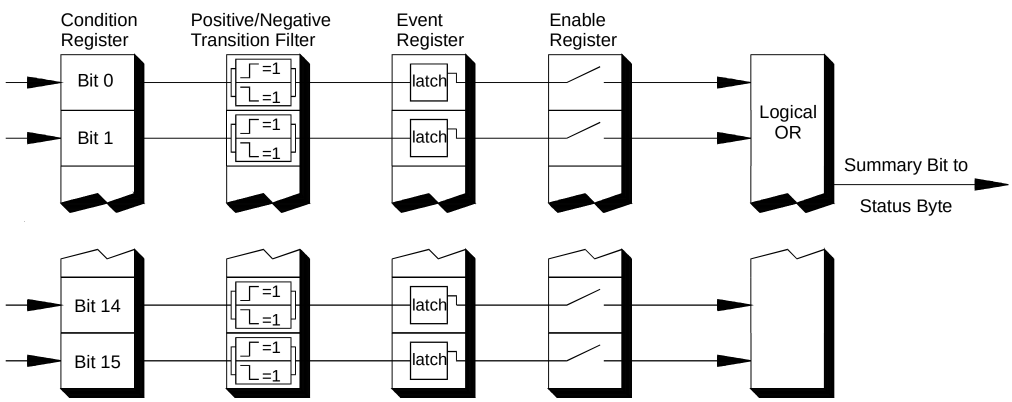


Figure 6-5. General Status Register Organization

Initializing the Status System

The following table shows the effect of Power-on, *RST, *CLS and STATus:PRESet on the status system register settings.

	SCPI Transition Filters	SCPI Enable Registers	SCPI Event Registers	IEEE 488.2 Registers ESE and SRE	IEEE 488.2 Registers SESR and STB
Power-on	preset	preset	clear	clear	clear
*RST	none	none	none	none	none
*CLS	none	none	clear	none	clear
STAT:PRESET	preset	preset	none	none	none

Subsystem Syntax

```
STATus
    :OPERation
        :CONDition?
        :ENABle  <enable_mask>
        :ENABle?
        [:EVENt]?
        :NTRansition <transition_mask>
        :NTRansition?
        :PTRansition <transition_mask>
        :PTRansition?
    :PRESet
    :QUEStionable
        :CONDition?
        :ENABle  <enable_mask>
        :ENABle?
        [:EVENt]?
        :NTRansition <transition_mask>
        :NTRansition?
        :PTRansition <transition_mask>
        :PTRansition?
```

The Status system contains four status groups

- Operation Status Group
- Questionable Data Group
- Standard Event Group
- Status Byte Group

This SCPI STATus subsystem communicates with the first two groups while IEEE-488.2 Common Commands (documented later in this chapter) communicate with Standard Event and Status Byte Groups.

Weighted Bit Values

Register queries are returned using decimal weighted bit values. Enable registers can be set using decimal, hex, octal, or binary. The following table can be used to help set Enable registers using decimal, and decode register queries.

Status System Decimal Weighted Bit Values

bit#	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
value	always 0	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1

The Operation Status Group

The Operation Status Group indicates the current operating state of the HP E1422. The bit assignments are:

Bit #	dec value	hex value	Bit Name	Description
0	1	0001_{16}	Calibrating	Set by CAL:TARE, and CAL:SETup. Cleared by CAL:TARE?, and CAL:SETup?. Set while *CAL? executes and reset when *CAL? completes. Set by CAL:CONFIG:VOLT or CAL:CONFIG:RES, cleared by CAL:VAL:VOLT or CAL:VAL:RES.
1-3				Not used
4	16	0010_{16}	Measuring	Set when instrument INITiated. Cleared when instrument returns to Trigger Idle State.
5-7				Not used
8	256	0100_{16}	Scan Complete	Set when each pass through a Scan List completed (may not indicate all measurements have been taken when TRIG:COUNT >1).
9	512	0200_{16}	SCP Trigger	An SCP has sourced a trigger event (future HP 1422 SCPs)
10	1024	0400_{16}	FIFO Half Full	The FIFO contains at least 32,768 readings
11	2048	0800_{16}	Algorithm Interrupted	The *interrupt()* function was called in an algorithm
12-15				Not used

STATus:OPERation:CONDition?

STATus:OPERation:CONDition? returns the decimal weighted value of the bits set in the Condition register.

Comments

- The Condition register reflects the real-time state of the status signals. The signals are not latched; therfore past events are not retained in this register (see STAT:OPER:EVENT?).

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** *CAL?, CAL:ZERO, INITiate[:IMMediate], STAT:OPER:EVENT?, STAT:OPER:ENABLE, STAT:OPER:ENABLE?
- ***RST Condition:** No Change
- **Use VXIplug&play function:** hpe1422_operCond_Q(...)

Usage STATUS:OPERATION:CONDITION? *Enter statement will return value from condition register*

STATus:OPERation:ENABle

STATus:OPERation:ENABle <*enable_mask*> sets bits in the Enable register that will enable corresponding bits from the Event register to set the Operation summary bit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable_mask	numeric (uint16)	0-32767	none

Comments

- Enable_mask may be sent as decimal, hex (#H), octal (#Q), or binary (#B).
- VXI Interrupts: When Operation Status Group bits 4, 8, 9, 10, or 11 are enabled, VXI card interrupts will occur as follows:

  When the event corresponding to bit 4 occurs <u>and then is cleared</u>, the card will generate a VXI interrupt. When the event corresponding to bit 8, 9, 10, or 11 occurs, the card will generate a VXI interrupt.

  NOTE: In C-SCPI, the C-SCPI overlap mode must be on for VXIbus interrupts to occur.
- **Related Commands:** *STB?, SPOLL, STAT:OPER:COND?, STAT:OPER:EVENT?, STAT:OPER:ENABLE?
- Cleared By: STAT:PRESet and power-on.
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage STAT:OPER:ENABLE 1 *Set bit 0 in the Operation Enable register*

STATus:OPERation:ENABle?

STATus:OPERation:ENABle? returns the value of bits set in the Operation Enable register.

Comments

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** *STB?, SPOLL, STAT:OPER:COND?, STAT:OPER:EVENT?, STAT:OPER:ENABLE
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

STAT:OPER:ENABLE?	*Enter statement returns current value of bits set in the Operation Enable register*

STATus:OPERation[:EVENt]?

STATus:OPERation[:EVENt]? returns the decimal weighted value of the bits set in the Event register.

Comments

- When using the Operation Event register to cause SRQ interrupts, STAT:OPER:EVENT? must be executed after an SRQ to re-enable future interrupts.
- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** *STB?, SPOLL, STAT:OPER:COND?, STAT:OPER:ENABLE, STAT:OPER:ENABLE?
- Cleared By: *CLS, power-on, and by reading the register.
- ***RST Condition:** No change
- **Use VXIplug&play function:** hpe1422_operEvent_Q(...)

Usage

STAT:OPER:EVENT?	*Enter statement will return the value of bits set in the Operation Event register*
STAT:OPER?	*Same as above*

STATus:OPERation:NTRansition

STATus:OPERation:NTRansition <*transition_mask*> sets bits in the Negative Transition Filter (NTF) register. When a bit in the NTF register is set to one, the corresponding bit in the Condition register must change from a one to a zero in order to set the corresponding bit in the Event register. When a bit in the NTF register is zero, a negative transition of the Condition register bit will not change the Event register bit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
transition_mask	numeric (uint16)	0-32767	none

Comments

- *transition_mask* may be sent as decimal, hex (#H), octal (#Q), or binary (#B).
- If both the STAT:OPER:PTR and STAT:OPER:NTR registers have a corresponding bit set to one, any transition, positive or negative will set the corresponding bit in the Event register.
- If neither the STAT:OPER:PTR or STAT:OPER:NTR registers have a corresponding bit set to one, transitions from the Condition register will have no effect on the Event register.
- **Related Commands:** STAT:OPER:NTR?, STAT:OPER:PTR
- Cleared By: STAT:PRESet and power-on.
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

```
STAT:OPER:NTR  16
```

When "Measuring" bit goes false, set bit 4 in Status Operation Event register.

STATus:OPERation:NTRansition?

STATus:OPERation:NTRansition? returns the value of bits set in the Negative Transition Filter (NTF) register.

Comments

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** STAT:OPER:NTR
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

```
STAT:OPER:NTR?
```

Enter statement returns current value of bits set in the NTF register

STATus:OPERation:PTRansition

STATus:OPERation:PTRansition <*transition_mask*> sets bits in the Positive

Transition Filter (PTF) register. When a bit in the PTF register is set to one, the corresponding bit in the Condition register must change from a zero to a one in order to set the corresponding bit in the Event register. When a bit in the PTF register is zero, a positive transition of the Condition register bit will not change the Event register bit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
transition_mask	numeric (uint16)	0-32767	none

Comments

- *transition_mask* may be sent as decimal, hex (#H), octal (#Q), or binary (#B).
- If both the STAT:OPER:PTR and STAT:OPER:NTR registers have a corresponding bit set to one, any transition, positive or negative will set the corresponding bit in the Event register.
- If neither the STAT:OPER:PTR or STAT:OPER:NTR registers have a corresponding bit set to one, transitions from the Condition register will have no effect on the Event register.
- **Related Commands:** STAT:OPER:PTR?, STAT:OPER:NTR
- Set to all ones by: STAT:PRESet and power-on.
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

STAT:OPER:PTR 16 — *When "Measuring" bit goes true, set bit 4 in Status Operation Event register.*

STATus:OPERation:PTRansition?

STATus:OPERation:PTRansition? returns the value of bits set in the Positive Transition Filter (PTF) register.

Comments

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** STAT:OPER:PTR
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

STAT:OPER:PTR? — *Enter statement returns current value of bits set in the PTF register*

STATus:PRESet

STATus:PRESet sets the Operation Status Enable and Questionable Data Enable registers to 0. After executing this command, none of the events in the Operation Event or Questionable Event registers will be reported as a summary bit in either the Status Byte Group or Standard Event Status Group. STATus:PRESet does not clear either of the Event registers.

Comments

- **Related Commands:** *STB?, SPOLL, STAT:OPER:ENABLE, STAT:OPER:ENABLE?, STAT:QUES:ENABLE, STAT:QUES:ENABLE?
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage STAT:PRESET *Clear both of the Enable registers*

The Questionable Data Group

The Questionable Data Group indicates when errors are causing lost or questionable data. The bit assignments are:

Bit #	dec value	hex value	Bit Name	Description
0-7				Not used
8	256	0100_{16}	Calibration Lost	At *RST or Power-on Control Processor has found a checksum error in the Calibration Constants. Read error(s) with SYST:ERR? and re-calibrate area(s) that lost constants.
9	512	0200_{16}	Trigger Too Fast	Scan not complete when another trigger event received.
10	1024	0400_{16}	FIFO Overflowed	Attempt to store more than 65,024 readings in FIFO.
11	2048	0800_{16}	Over voltage Detected on Input	If the input protection jumper has not been cut, the input relays have been opened and *RST is required to reset the module. Overvoltage will also generate an error.
12	4096	1000_{16}	VME Memory Overflow	The number of readings taken exceeds VME memory space.
13	8192	2000_{16}	Setup Changed	Channel Calibration in doubt because SCP setup may have changed since last *CAL? or CAL:SETup command. (*RST always sets this bit).
14-15				Not used

STATus:QUEStionable:CONDition?

STATus:QUEStionable:CONDition? returns the decimal weighted value of the bits set in the Condition register.

Comments

- The Condition register reflects the real-time state of the status signals. The signals are not latched; therfore past events are not retained in this register (see STAT:QUES:EVENT?).
- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** CAL:VALUE:RESISTANCE, CAL:VALUE:VOLTAGE, STAT:QUES:EVENT?, STAT:QUES:ENABLE, STAT:QUES:ENABLE?
- ***RST Condition:** Bit 13, "Setup Changed" is set to 1
- **Use VXIplug&play function:** hpe1422_quesCond_Q(...)

Usage STATUS:QUESTIONABLE:CONDITION? *Enter statement will return value from condition register*

STATus:QUEStionable:ENABle

STATus:QUEStionable:ENABle <*enable_mask*> sets bits in the Enable register that will enable corresponding bits from the Event register to set the Questionable summary bit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
enable_mask	numeric (uint16)	0-32767	none

Comments

- *Enable_mask* may be sent as decimal, hex (#H), octal (#Q), or binary (#B).
- VXI Interrupts: When bits 9, 10, or 11 are enabled and C-SCPI overlap mode is on (or if you are using non-compiled SCPI), VXI card interrupts will be enabled. When the event corresponding to bit 9, 10, or 11 occurs, the card will generate a VXI interrupt.
- **Related Commands:** *STB?, SPOLL, STAT:QUES:COND?, STAT:QUES:EVENT?, STAT:QUES:ENABLE?
- Cleared By: STAT:PRESet and power-on.
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage STAT:QUES:ENABLE 128 *Set bit 7 in the Questionable Enable register*

STATus:QUEStionable:ENABle?

STATus:QUEStionable:ENABle? returns the value of bits set in the Questionable Enable register.

Comments

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** *STB?, SPOLL, STAT:QUES:COND?, STAT:QUES:EVENT?, STAT:QUES:ENABLE
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

STAT:QUES:ENABLE?	*Enter statement returns current value of bits set in the Questionable Enable register*

STATus:QUEStionable[:EVENt]?

STATus:QUEStionable[:EVENt]? returns the decimal weighted value of the bits set in the Event register.

Comments

- When using the Questionable Event register to cause SRQ interrupts, STAT:QUES:EVENT? must be executed after an SRQ to re-enable future interrupts.
- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- Cleared By: *CLS, power-on, and by reading the register.
- **Related Commands:** *STB?, SPOLL, STAT:QUES:COND?, STAT:QUES:ENABLE, STAT:QUES:ENABLE?
- **Use VXIplug&play function:** hpe1422_quesEvent_Q(...)

Usage

STAT:QUES:EVENT?	*Enter statement will return the value of bits set in the Questionable Event register*
STAT:QUES?	*Same as above*

STATus:QUEStionable:NTRansition

STATus:QUEStionable:NTRansition *<transition_mask>* sets bits in the Negative Transition Filter (NTF) register. When a bit in the NTF register is set to one, the corresponding bit in the Condition register must change from a one to a zero in order to set the corresponding bit in the Event register. When a bit in the NTF register is zero, a negative transition of the Condition register bit will not change the Event register bit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
transition_mask	numeric (uint16)	0-32767	none

Comments

- *transition_mask* may be sent as decimal, hex (#H), octal (#Q), or binary (#B).
- If both the STAT:QUES:PTR and STAT:QUES:NTR registers have a corresponding bit set to one, any transition, positive or negative will set the corresponding bit in the Event register.
- If neither the STAT:QUES:PTR or STAT:QUES:NTR registers have a corresponding bit set to one, transitions from the Condition register will have no effect on the Event register.
- **Related Commands:** STAT:QUES:NTR?, STAT:QUES:PTR
- Cleared By: STAT:PRESet and power-on.
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage STAT:QUES:NTR 1024 — *When "FIFO Overflowed" bit goes false, set bit 10 in Status Questionable Event register.*

STATus:QUEStionable:NTRansition?

STATus:QUEStionable:NTRansition? returns the value of bits set in the Negative Transition Filter (NTF) register.

Comments

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** STAT:QUES:NTR
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage STAT:QUES:NTR? — *Enter statement returns current value of bits set in the NTF register*

STATus:QUEStionable:PTRansition

STATus:QUEStionable:PTRansition <*transition_mask*> sets bits in the Positive Transition Filter (PTF) register. When a bit in the PTF register is set to one, the corresponding bit in the Condition register must change from a zero to a one in order to set the corresponding bit in the Event register. When a bit in the PTF register is zero, a positive transition of the Condition register bit will not change the Event register bit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
transition_mask	numeric (uint16)	0-32767	none

Comments

- *transition_mask* may be sent as decimal, hex (#H), octal (#Q), or binary (#B).
- If both the STAT:QUES:PTR and STAT:QUES:NTR registers have a corresponding bit set to one, any transition, positive or negative will set the corresponding bit in the Event register.
- If neither the STAT:QUES:PTR or STAT:QUES:NTR registers have a corresponding bit set to one, transitions from the Condition register will have no effect on the Event register.
- **Related Commands:** STAT:QUES:PTR?, STAT:QUES:NTR
- Set to all ones by: STAT:PRESet and power-on.
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

STAT:QUES:PTR 1024 — *When "FIFO Overflowed" bit goes true, set bit 10 in Status Operation Event register.*

STATus:QUEStionable:PTRansition?

STATus:QUEStionable:PTRansition? returns the value of bits set in the Positive Transition Filter (PTF) register.

Comments

- **Returned Value:** Decimal weighted sum of all set bits. The data type is **uint16**.
- **Related Commands:** STAT:QUES:PTR
- ***RST Condition:** No change
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Usage

STAT:OPER:PTR? — *Enter statement returns current value of bits set in the PTF register*

SYSTem

The SYSTem subsystem is used to query for error messages, types of Signal Conditioning Plug-ons (SCPs), and the SCPI version currently implemented.

Subsystem Syntax

SYSTem
  :CTYPe? (@<*channel*>)
  :ERRor?
  :VERSion?

SYSTem:CTYPe?

SYSTem:CTYPe? (@<*channel*>) returns the identification of the Signal Conditioning Plug-On installed at the specified channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- *channel* must specify a single channel only.
- **Returned Value:** An example of the response string format is:
  HEWLETT-PACKARD,E1422 Option <*option number and description*> SCP,0,0

  The data type is **string**. For specific response string, refer to the appropriate SCP manual. If <*channel*> specifies a position where no SCP is installed, the module returns the response string:
  0,No SCP at this Address,0,0
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage SYST:CTYPE? (@100) *return SCP type install at channel 0*

SYSTem:ERRor?

SYSTem:ERRor? returns the latest error entered into the Error Queue.

Comments

- SYST:ERR? returns one error message from the Error Queue (returned error is removed from queue). To return all errors in the queue, repeatedly execute SYST:ERR? until the error message string = +0, "No error"
- **Returned Value:** Errors are returned in the form:
  ±<error number>, "<*error message string*>"
- **RST Condition:** Error Queue is empty.

- **Use VXIplug&play function:** hpe1422_error_query(...)

Usage SYST:ERR? *returns the next error message from the Error Queue*

SYSTem:VERSion?

SYSTem:VERSion? returns the version of SCPI this instrument complies with.

Comments

- **Returned Value:** String "1990". The data type is **string**.
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage SYST:VER? *Returns "1990"*

TRIGger

The TRIGger command subsystem controls the behavior of the trigger system once it is initiated (see INITiate command subsystem).

Figure 6-6 shows the overall Trigger System model. The shaded area shows the ARM subsystem portion.

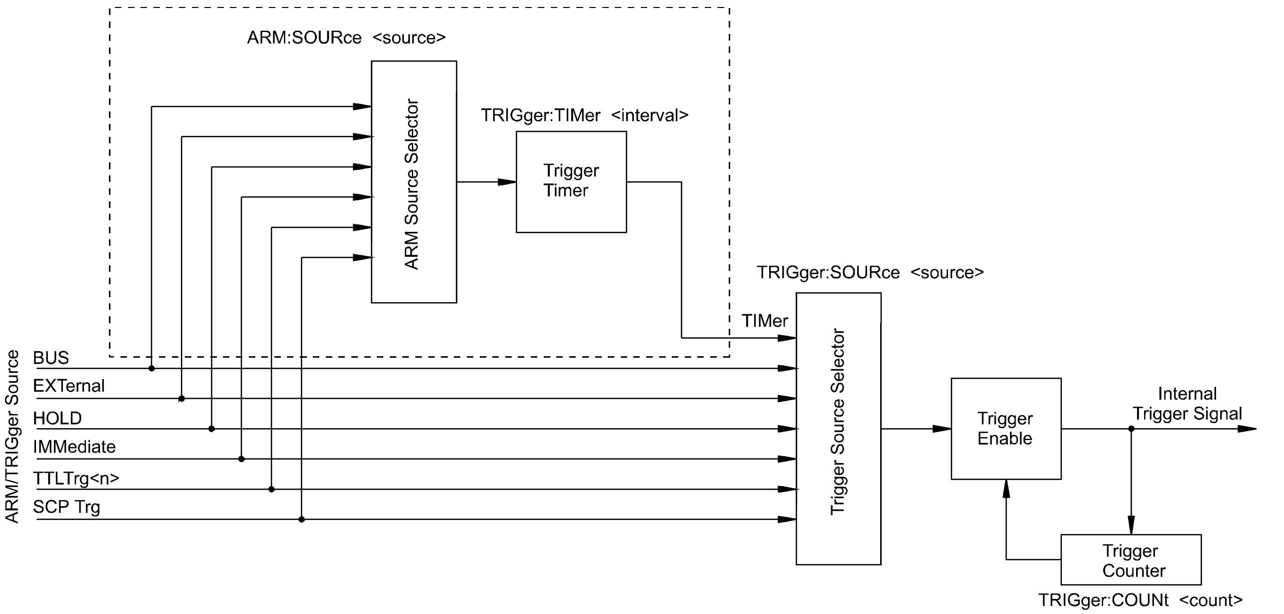


Figure 6-6. Logical Trigger Model

Caution **Algorithms execute at most once per trigger event. Should trigger events cease (external trigger source stops) or are ignored (TRIGger:COUNt reached), algorithms execution will stop. In this case control outputs are left at the last value set by the algorithms. Depending on the process, this uncontrolled situation could even be dangerous. Make certain that you have put your process into a safe state before you halt (stop triggering) execution of a controlling algorithm.**

The HP E1535 Watchdog Timer SCP was specifically developed to automatically signal that an algorithm has stopped controlling a process. Use of the Watchdog Timer is recommended for critical processes.

Event Sequence

Figure 6-7 shows how the module responds to various trigger/arm configurations.

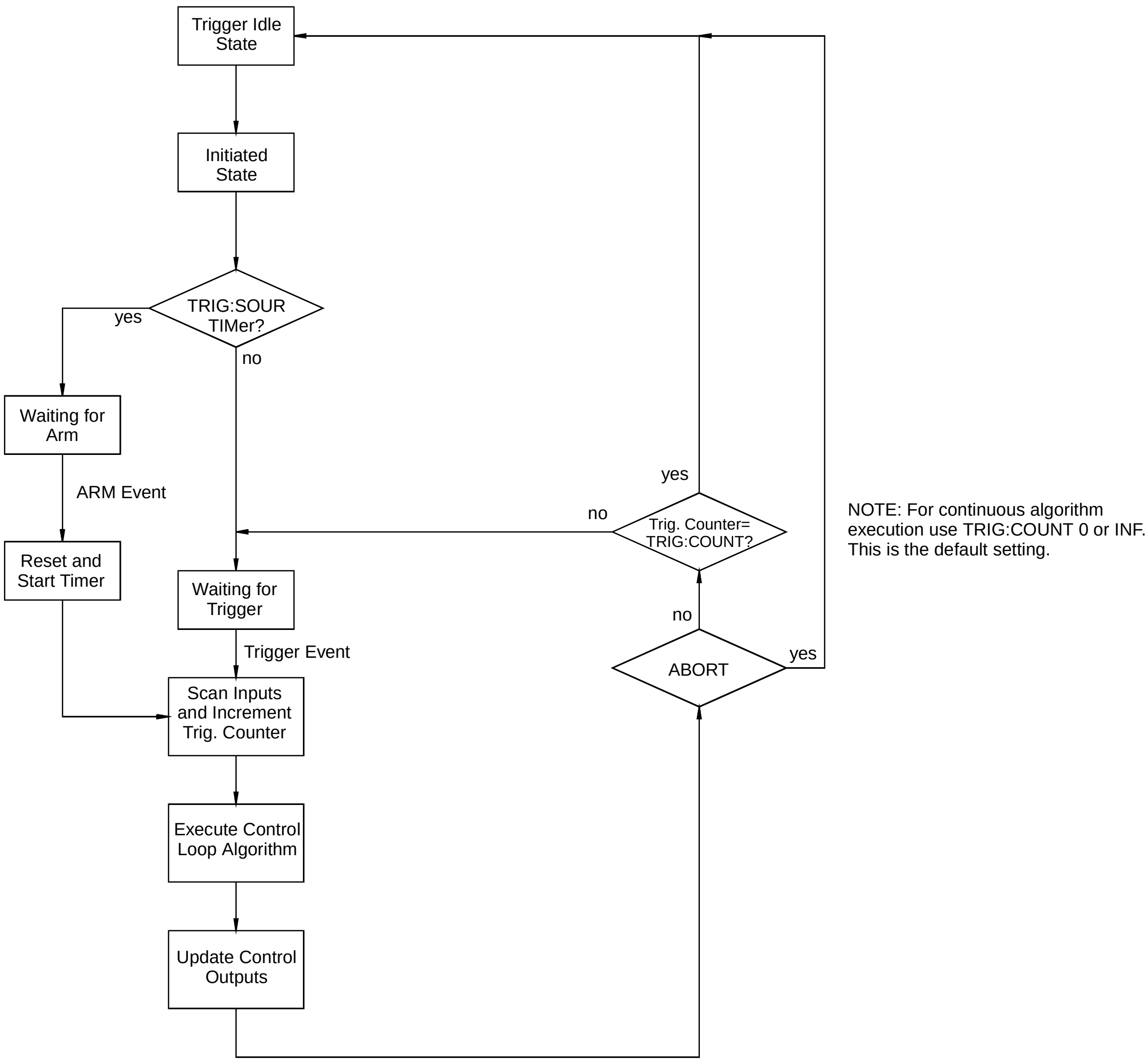


Figure 6-7. Trigger/Scan Sequence Diagram

Subsystem Syntax

TRIGger
    :COUNt *<trig_count>*
    :COUNt?
    [:IMMediate]
    :SOURce BUS | EXTernal | HOLD | SCP | IMMediate | TIMer | TTLTrg<n>
    :SOURce?
    :TIMer
        [:PERiod] *<trig_interval>*
        [:PERiod]?

TRIGger:COUNt

TRIGger:COUNt <*trig_count*> sets the number of times the module can be triggered before it returns to the Trigger Idle State. The default count is 1. Note that this default was chosen to make testing data aquisition scan list easier (only one scan list woth of data in FIFO per trigger). For algorithm operation, you will probably want to change the count to INFinite to accepts continuous triggers. See Figure 6-7 on page 346

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
Trig_count	numeric (uint16) (string)	0 to 65535 | INF	none

Comments

- When *trig_count* is set to 0 or INF, the trigger counter is disabled. Once INITiated the module will return to the Waiting For Trigger State after each trigger event. The ABORT (preferred) and *RST commands will return the module to the Trigger Idle State. ABORT is preferred since *RST also returns other module configurations to their default settings.
- The default count is 0
- **Related Commands:** TRIG:COUNT?
- ***RST Condition:** TRIG:COUNT 0
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

TRIG:COUNT 10	*Set the module to make 10 passes all enabled algorithms.*
TRIG:COUNT 0	*Set the module to accept unlimited triggers (the default)*

TRIGger:COUNt?

TRIGger:COUNt? returns the currently set trigger count.

Comments

- If TRIG:COUNT? returns 0, the trigger counter is disabled and the module will accept an unlimited number of trigger events.
- **Returned Value:** Numeric 0 through 65,535. The data type is **int32**.
- **Related Commands:** TRIG:COUNT
- ***RST Condition:** TRIG:COUNT? returns 0
- **Send with VXIplug&play Function:** hpe1422_cmdInt32_Q(...)

Usage TRIG:COUNT? — *Query for trigger count setting*
enter statement — *Returns the TRIG:COUNT setting*

TRIGger[:IMMediate]

TRIGger[:IMMediate] causes one trigger when the module is set to the TRIG:SOUR BUS or TRIG:SOUR HOLD mode.

Comments

- This command is equivalent to the *TRG common command or the IEEE-488.2 "GET" bus command.
- **Related Commands:** TRIG:SOURCE
- **Use VXIplug&play function:** hpe1422_trigImm(...)

Usage TRIG:IMM — *Use TRIGGER to start a measurement scan*

TRIGger:SOURce

TRIGger:SOURce ***<trig_source>*** configures the trigger system to respond to the trigger event.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
trig_source	discrete (string)	BUS | EXT | HOLD | IMM | SCP | TIM | TTLTrg<n>	none

Comments

- The following table explains the possible choices.

BUS	TRIGger[:IMMediate], *TRG, GET (for HP-IB)
EXTernal	“TRG” signal on terminal module
HOLD	TRIGger[:IMMediate]
IMMediate	The trigger event is always satisfied.
SCP	SCP Trigger Bus (future HP or SCP Breadboard)
TIMer	The internal trigger timer
TTLTrg<n>	The VXIbus TTLTRG lines (n=0 through 7)

Note The ARM system only exists while TRIG:SOUR is TIMer. When TRIG:SOUR is not TIMer, SCPI compatibility requires that ARM:SOUR be IMM or an Error

-221,"Settings conflict" will be generated.

- While TRIG:SOUR is IMM, you need only INITiate the trigger system to start a measurement scan.
- **When Accepted: Before INIT only**.
- **Related Commands:** ABORt, INITiate, *TRG
- ***RST Condition:** TRIG:SOUR TIMER
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage TRIG:SOUR EXT *Hardware trigger input at Connector Module*

TRIGger:SOURce?

TRIGger:SOURce? returns the current trigger source configuration.

- **Returned Value:** Discrete; one of BUS, EXT, HOLD, IMM, SCP, TIM, or TTLT0 through TTLT7. The data type is **string**. See the TRIG:SOUR command for more response data information.
- **Send with VXIplug&play Function:** hpe1422_cmdString_Q(...)

Usage TRIG:SOUR? *ask HP E1422 to return trigger source configuration*

TRIGger:TIMer[:PERiod]

TRIGger:TIMer[:PERiod] <*trig_interval*> sets the interval between scan triggers. Used with the TRIG:SOUR TIMER trigger mode.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
trig_interval	numeric (float32) (string)	100E-6 to 6.5536 | MIN | MAX	seconds

Comments

- In order for the TRIG:TIMER to start it must be Armed. For information on timer arming see the ARM subsystem in this command reference.
- The default interval is 10E-3 seconds. ***interval*** may be specified in seconds, milliseconds (ms), or microseconds (us). For example; .0016, 1.6ms or 1600us. The resolution for *interval* is 100 µ second.

- TRIG:TIMer periods shorter than the value returned from the ALG[:EXPL]:TIME? command may result in "Trigger too fast"errors.
- **When Accepted: Before INIT only**.
- **Related Commands:** TRIG:SOUR TIMER, ARM:SOUR, ARM:IMM, INIT, TRIG:SOUR?, ALG:EXPL:TIME?
- ***RST Condition:** TRIG:TIM 1.0E-3
- **Send with VXIplug&play Function:** hpe1422_cmd(...)

Usage

TRIG:TIMER 1.0E-1	*Set the module to scan inputs and execute all algorithms every 100 mS*
TRIG:TIMER 1	*Set the module to scan inputs and execute all algorithms every second*

TRIGger:TIMer[:PERiod]?

TRIGger:TIMer[:PERiod]? returns the currently set Trigger Timer interval.

Comments

- **Returned Value:** Numeric 1 through 6.5536. The data type is **float32**.
- **Related Commands:** TRIG:TIMER
- ***RST Condition:** 1.0E-4
- **Send with VXIplug&play Function:** hpe1422_cmdReal64_Q(...)

Usage

TRIG:TIMER?	*Query trig timer*
enter statement	*Returns the timer setting*

IEEE-488.2 Common Command Reference

*CAL?

***CAL?** Calibration command. The calibration command causes the Channel Calibration function to be performed for every module channel. The Channel Calibration function includes calibration of A/D Offset, and Gain and Offset for all 64 channels. This calibration is accomplished using internal calibration references. The *CAL? command causes the module to calibrate A/D offset and gain, and all channel offsets. This may take many minutes to complete. The actual time it will take your HP E1422 to complete *CAL? depends on the mix of SCPs installed. *CAL performs literally hundreds of measurements of the internal calibration sources for each channel and must allow 17 time constants of settling wait each time a filtered channel's calibrations source value is changed. The *CAL procedure is internally very sophisticated and results in an extremely well calibrated module.

To perform Channel Calibration on multiple HP E1422s, use CAL:SETup/CAL:SETup?.

Note that the scope of the *CAL? and CAL:SETup commands is limited to the HP E1422A and the SCPs it contains. They do not calibrate Remote Signal Conditioning Units like the HP E1529A. You must use CAL:REMote? in addition to *CAL?/CAL:SETup for RSC units.

- **Returned Value:**

Value	Meaning	Further Action
0	Cal OK	None
-1	Cal Error	Query the Error Queue (SYST:ERR?) See "Error Messages" on page 401

The data type for this returned value is **int16**.

- **When Accepted:** Not while INITiated
- **Related Commands:** CALibration:SETup, CALibration:SETup?, CALibration:STORe ADC
- CAL:STOR ADC stores the calibration constants for *CAL? and CAL:SETup into non-volatile memory.
- Executing this command **does not** alter the module's programmed state (function, range, etc.). It does however clear STAT:QUES:COND? register bit 13.
- **Send with VXIplug&play Function:** hpe1422_cmdInt16_Q(...)

Note If Open Transducer Detect (OTD) is enabled when *CAL? is executed, the module will disable OTD, wait 1 minute to allow channels to settle, perform the calibration, and then re-enable OTD. If your program turns off OTD before executing *CAL?, it should also wait 1 minute for settling.

*CLS

***CLS** Clear Status Command. The *CLS command clears all status event registers (Standard Event Status Event Register, Standard Operation Status Event Register, Questionable Data Event Register) and the instrument's error queue. This clears the corresponding summary bits (bits 3, 5, & 7) in the Status Byte Register. *CLS does not affect the enable bits in any of the status register groups. (The SCPI command STATus:PRESet *does* clear the Operation Status Enable and Questionable Data Enable registers.) *CLS disables the Operation Complete function (*OPC command) and the Operation Complete Query function (*OPC? command).

Send with VXIplug&play Function: hpe1422_cmd(...)

*DMC

DMC** ***<name>*,*<cmd_data> Define Macro Command. Assigns one, or a sequence of commands to a named macro.

The command sequence may be composed of SCPI and/or Common commands.

<*name*> may be the same as a SCPI command, but may not be the same as a Common command. When a SCPI named macro is executed, the macro rather than the SCPI command is executed. To regain the function of the SCPI command, execute *EMC 0 command.

<*cmd_data*> is sent as *arbitrary block program data* (see "Arbitrary Block Program and Response Data" on page 201).

*EMC

EMC *<enable> Enable Macro Command. When <enable> is non-zero, macros are enabled. When <*enable*> is zero, macros are disabled.

*EMC?

***EMC?** Enable Macro query. Returns either 1 (macros are enabled), or 0 (macros are disabled).

*ESE

***ESE <*mask*>** Standard Event Status Enable Register Command. Enables one or more events in the Standard Event Status Register to be reported in bit 5 (the Standard Event Status Summary Bit) of the Status Byte Register. You enable an event by specifying its decimal weight for <*mask*>. To enable more than one event (bit), specify the sum of the decimal weights. The data type for <*mask*> is **int16**.

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Weighted Value	128	64	32	16	8	4	2	1
Event	power-On	User Request	Command Error	Execution Error	Device Dependent Error	Query Error	Request Control	Operation Complete

Send with VXIplug&play Function: hpe1422_cmd(...)

*ESE?

***ESE?** Standard Event Status Enable Query. Returns the weighted sum of all enabled (unmasked) bits in the Standard Event Status Register. The data type for this returned value is **int16**.

*ESR?

***ESR?** Standard Event Status Register Query. Returns the weighted sum of all set bits in the Standard Event Status Register. After reading the register, *ESR? clears the register. The events recorded in the Standard Event Status Register are independent of whether or not those events are enabled with the *ESE command to set the Standard Event Summary Bit in the Status Byte Register. The Standard Event bits are described in the *ESE command. The data type for this returned value is **int16**.

*GMC?

***GMC? <*name*>** Get Macro query. Returns arbitrary block response data which contains the command or command sequence defined for <*name*>. For more information see “Arbitrary Block Program and Response Data” on page 201.

*IDN?

***IDN?** Identity. Returns the device identity. The response consists of the following four fields (fields are separated by commas):

- Manufacturer
- Model Number
- Serial Number (returns 0 if not available)
- Driver Revision (returns 0 if not available)

*IDN? returns the following response strings depending on model and options:
HEWLETT-PACKARD,E1422A,<*serial number*>,<*revision number*>

- The data type for this returned value is **string**.

Note The revision will vary with the revision of the driver software installed in your system. This is the only indication of which version of the driver is installed.

*LMC?

***LMC?** Learn Macros query. Returns a quoted string name for each currently defined macro. If more than one macro is defined, the strings are separated by commas (,). If no macro is defined, *LMC? returns a null string.

*OPC

***OPC** Operation Complete. Causes an instrument to set bit 0 (Operation Complete Message) in the Standard Event Status Register when all pending operations invoked by SCPI commands have been completed. By enabling this bit to be reflected in the Status Byte Register (*ESE 1 command), you can ensure synchronization between the instrument and an external computer or between multiple instruments.

Note Do not use *OPC to determine when the CAL:SETUP or CAL:TARE commands have completed. Instead, use their query forms CAL:SETUP? or CAL:TARE?.

Send with VXIplug&play Function: hpe1422_cmd(...)

*OPC?

***OPC?** Operation Complete Query. Causes an instrument to place a 1 into the instrument's output queue when all pending instrument operations invoked by SCPI commands are finished. By requiring your computer to read this response before continuing program execution, you can ensure synchronization between one or more instruments and the computer. The data type for this returned value is **int16**.

Note Do not use *OPC? to determine when the CAL:SETUP or CAL:TARE commands have completed. Instead, use their query forms CAL:SETUP? or CAL:TARE?.

If an algorithm is running contiuously, then *OPC? will never return (will "hang"). In this case, send a device clear, then *RST or ABORT to stop the algorithm. *OPC?

must be used with care when the HP E1422A is INITiated.

*PMC

***PMC** Purge Macros Command. Purges all currently defined macros.

*RMC

***RMC <*name*>** Remove individual Macro Command. Removes the named macro command.

*RST

***RST** Reset Command. Resets the HP E1422 as follows:

- Erases all algorithms
- All elements in the Input Channel Buffer (I100 - I163) set to zero.
- All elements in the Output Channel Buffer (O100-O163) set to zero
- Defines all Analog Input channels to measure voltage
- Configures all Digital I/O channels as inputs
- Resets HP E1531 and HP E1532 Analog Output SCP channels to zero

- **When Accepted:** Not while INITiated

- **Use VXIplug&play function:** hpe1422_reset(...)

WARNING **Note the change in character of output channels when *RST is received. Digital outputs change to inputs (appearing now is 1kW to +3v, a TTL one), and analog control outputs change to zero (current or voltage). Keep these changes in mind when applying the HP E1422 to your system, or engineering a system for operation with the HP E1422. Also note that each analog output channels disconnects for 5-6 milliseconds to discharge to zero at each *RST.**

It isn't difficult to have the HP E1422 signal your system when *RST is executed. A solution that can provide signals for several types of failures as well as signaling when *RST is executed is the HP E1535 Watchdog Timer SCP. The Watchdog SCP even has an input through which you can command all of the HP E1422's channels to disconnect from your system.

- Sets the trigger system as follows:
  -- TRIGGER:SOURCE TIMER
  -- TRIGGER:TIMER 10E-3

-- TRIGGER:COUNT 0 (infinite)
-- ARM:SOURCE IMMEDIATE

- SAMPLE:TIMER 40E-6
- Aborts all pending operations, returns to Trigger Idle state
- Disables the *OPC and *OPC? modes
- MEMORY:VME:ADDRESS 240000; MEMORY:VME:STATE OFF; MEMORY:VME:SIZE 0
- Sets STAT:QUES:COND? bit 13

*RST does not affect:

- Calibration data
- The output queue
- The Service Request Enable (SRE) register
- The Event Status Enable (ESE) register

*SRE

***SRE <*mask*>** Service Request Enable. When a service request event occurs, it sets a corresponding bit in the Status Byte Register (this happens whether or not the event has been enabled (unmasked) by *SRE). The *SRE command allows you to identify which of these events will assert an HP-IB service request (SRQ). When an event is enabled by *SRE and that event occurs, it sets a bit in the Status Byte Register and issues an SRQ to the computer (sets the HP-IB SRQ line true). You enable an event by specifying its decimal weight for <*mask*>. To enable more than one event, specify the sum of the decimal weights. Refer to "The Status Byte Register"r for a table showing the contents of the Status Byte Register. The data type for <*mask*> is **int16**.

Bit #	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Weighted Value	128	64	32	16	8	4	2	1
Event	Operation Status	Request Service	Standard Event	Message Available	Questionable Status	not used	not used	not used

Send with VXIplug&play Function: hpe1422_cmd(...)

*SRE?

***SRE?** Status Register Enable Query. Returns the weighted sum of all enabled (unmasked) events (those enabled to assert SRQ) in the Status Byte Register. The data type for this returned value is **int16**.

*STB?

***STB?** Status Byte Register Query. Returns the weighted sum of all set bits in the Status Byte Register. Refer to the *ESE command earlier in this chapter for a table showing the contents of the Status Byte Register. *STB? does not clear bit 6 (Service

Request). The Message Available bit (bit 4) may be cleared as a result of reading the response to *STB?. The data type for this returned value is **int16**.

- **Use VXIplug&play function:** hpe1422_readStatusByte_Q(...)

Send with VXIplug&play Function: hpe1422_cmd(...)

*TRG

***TRG** Trigger. Triggers an instrument when the trigger source is set to bus (TRIG:SOUR BUS command) and the instrument is in the Wait for Trigger state.

Send with VXIplug&play Function: hpe1422_cmd(...)

*TST?

***TST?** Self-Test. Causes an instrument to execute extensive internal self-tests and returns a response showing the results of the self-test.

Notes

1. During the first 5 minutes after power is applied, *TST? may fail. Allow the module to warm-up before executing *TST?.
2. Module must be screwed securely to mainframe.

Comments

- **Use VXIplug&play function:** hpe1422_self_test(...)
- **Returned Value:**

Value	Meaning	Further Action
0	*TST? OK	None
-1	*TST? Error	Query the Error Queue (SYST:ERR?) for error 3052. See explanation below.

- IF error 3052 'Self test failed. Test info in FIFO' is returned. A FIFO value of 1 through 99 or >=300 is a failed test number. A value of 100 through 163 is a channel number for the failed test. A value of 200 through 204 is an A/D range number for the failed test where 200=.0625, 201=.25V, 202=1V, 203=4V, and 204=16V ranges. For example DATA:FIFO? returns the values 72 and 108. This indicates that test number 72 failed on channel 8.

  Test numbers 20, 30-37, 72, 74-76, and 80-93 may indicate a problem with a Signal Conditioning Plug-on.

  For tests 20, and 30-37, remove all SCPs and see if *TST? passes. If so, replace SCPs one at a time until you find the one causing the problem.

For tests 72, 74-76, and 80-93, try to re-seat the SCP that the channel number(s) points to, or move the SCP and see if the failure(s) follow the SCP. If the problems move with the SCP, replace the SCP.

These are the only tests where the user should troubleshoot a problem. Other tests which fail should be referred to qualified repair personnel.

Note Executing *TST? returns the module to its *RST state. *RST causes the FIFO data format to return to its default of ASC,7. If you want to read the FIFO for *TST? diagnostic information and you want that data in other than the ASCII,7 format, be certain to set the data FIFO format to the desired format (FORMAT command) after completion of *TST? but before executing a SENSE:DATA:FIFO: query command.

- The data type for this returned value is **int16**.
- Following *TST?, the module is placed in the *RST state. This returns many of the module's programmed states to their defaults. See "*RST" on page 355 for a list of the module's default states.
- *TST? performs the following tests on the HP E1422 and installed Signal Conditioning Plug-ons:

DIGITAL TESTS:

Test#	Description
1-3:	Writes and reads patterns to registers via A16 & A24
4-5:	Checks FIFO and CVT
6:	Checks measurement complete (Measuring) status bit
7:	Checks operation of FIFO half and FIFO full IRQ generation
8-9:	Checks trigger operation

ANALOG FRONT END DIGITAL TESTS:

Test#	Description
20:	Checks that SCP ID makes sense
30-32:	Checks relay driver and fet mux interface with EU CPU
33,71:	Checks opening of all relays on power down or input overvoltage
34-37:	Check fet mux interface with A/D digital

ANALOG TESTS:

Test#	Description
40-42:	Checks internal voltage reference

ANALOG TESTS: (continued)

Test#	Description
43-44:	Checks zero of A/D, internal cal source and relay drives
45-46:	Checks fine offset calibration DAC
47-48:	Checks coarse offset calibration DAC
49:	Checks internal + and -15V supplies
50-53:	Checks internal calibration source
54-55:	Checks gain calibration DAC
56-57:	Checks that autorange works
58-59:	Checks internal current source
60-63:	Checks front end and A/D noise and A/D filter
64:	Checks zeroing of coarse and fine offset calibration DACs
65-70:	Checks current source and CAL BUS relay and relay drives and OHM relay drive
71:	See 33
72-73:	Checks continuity through SCPs, bank relays and relay drivers
74:	Checks open transducer detect
75:	Checks current leakage of the SCPs
76:	Checks voltage offset of the SCPs
80:	Checks mid-scale strain dac output. Only reports first channel of SCP.
81:	Checks range of strain dac. Only reports first channel of SCP.
82:	Checks noise of strain dac. Only reports first channel of SCP.
83:	Checks bridge completion leg resistance each channel.
84:	Checks combined leg resistance each channel.
86:	Checks current source SCP’s OFF current.
87:	Checks current source SCP’s current dac mid-scale.
88:	Checks current source SCP’s current dac range on HI and LO ranges.
89:	Checks current source compliance
90:	Checks strain SCP’s Wagner Voltage control.
91:	Checks autobalance dac range with input shorted.
92:	Sample and Hold channel holds value even when input value changed.
93:	Sample and Hold channel held value test for droop rate.

ANALOG OUTPUT AND DIGITAL I/O TESTS

301:	Current and Voltage Output SCPsdigital DAC control.
302:	Current and Voltage Output SCPsDAC noise.
303:	Current Output SCPoffset
304:	Current Output SCPgain shift
305:	Current Output SCPoffset
306:	Current Output SCPlinearity
307:	Current Output SCPlinearity
308:	Current Output SCPturn over
313:	Voltage Output SCPoffset
315:	Voltage Output SCPoffset
316:	Voltage Output SCPlinearity

317: Voltage Output SCPlinearity
318: Voltage Output SCPturn over

331: Digital I/O SCPinternal digital interface
332: Digital I/O SCPuser input
333: Digital I/O SCPuser input
334: Digital I/O SCPuser output
335: Digital I/O SCPuser output
336: Digital I/O SCPoutput current
337: Digital I/O SCPoutput current

341: Freq/PWM/FM SCPinternal data0 register
342: Freq/PWM/FM SCPinternal data1 register
343: Freq/PWM/FM SCPinternal parameter register
344: Freq/PWM/FM SCPon-board processor self-test
345: Freq/PWM/FM SCPon-board processor self-test
346: Freq/PWM/FM SCPuser inputs
347: Freq/PWM/FM SCPuser outputs
348: Freq/PWM/FM SCPoutputs ACTive/PASSive
349: Freq/PWM/FM SCPoutput interrupts

350: Watchdog SCPenable/disable timer
351: Watchdog SCPrelay drive and coil closed
352: Watchdog SCPrelay drive and coil open
353: Watchdog SCPI/O Disconnect line
354: Watchdog SCPI/O Disconnect supply

*WAI

***WAI** Wait-to-continue. Prevents an instrument from executing another command until the operation begun by the previous command is finished (sequential operation).

Note Do not use *WAI to determine when the CAL:SETUP or CAL:TARE commands have completed. Instead, use their query forms CAL:SETUP? or CAL:TARE?. CAL:SETUP? and CAL:TARE? return a value only after the CAL:SETUP or CAL:TARE operations are complete.

Send with VXIplug&play Function: hpe1422_cmd(...)

Command Quick Reference

The following tables summarize SCPI and IEEE-488.2 Common (*) commands for the HP E1422A Remote Channel Multifunction Module.

SCPI Command Quick Reference

Command	Description
ABORt	Stops scanning immediately and sets trigger system to idle state (scan lists are unaffected)
ALGorithm	Subsystem to define, configure, and enable loop control algorithms
[:EXPLicit]	
:ARRay '<*alg_name*>','<*array_name*>',<*block_data*>	Defines contents of array <*array_name*> in algorithm <*alg_name*> or if <*alg_name*> is "GLOBALS", defines values global to all algorithms.
:ARRay? '<*alg_name*>','<*array_name*>'	Returns block data from <*array_name*> in algorithm <*alg_name*> or if <*alg_name*> is "GLOBALS", returns values from a global array.
:DEFine '<*alg_name*>'[,<*swap_size*>],'<*program_data*>'	Defines algorithms or global variables. <program_data> is 'C' source of algorithm or global declaration.
:SCALar '<*alg_name*>','<*var_name*>',<*value*>	Defines value of variable <*var_name*> in algorithm <*alg_name*> or if <*alg_name*> is "GLOBALS", defines a value global to all algorithms.
:SCALar? '<*alg_name*>','<*var_name*>'	Returns value from <*var_name*> in algorithm <*alg_name*> or if <*alg_name*> is "GLOBALS", returns a value from global variable.
:SCAN	
:RATio '<*alg_name*>',<*ratio*>	Sets scan triggers per execution of <*alg_name*> (send also ALG:UPD)
:RATio? '<*alg_name*>'	Returns scan triggers per execution of <*alg_name*>
:SIZe? '<*alg_name*>'	Returns size in words of named algorithm
:STATe '<*alg_name*>',ON | OFF	Enables/disables named algorithm after ALG:UPDATE sent
:STATe? '<*alg_name*>'	Returns state of named algorithm
:TIME? '<*alg_name*>' | MAIN	Returns worst case alg execution time. Use "MAIN" for overall time.
:FUNCtion	
:DEFine '<*function_name*>',<*range*>,<*offset*>,<*func_data*>	Defines a custom conversion function
:OUTPut	
:DELay <*delay*> | AUTO	Sets the delay from scan trigger to start of outputs
:DELay?	Returns the delay from scan trigger to start of outputs
:UPDate	
[:IMMediate]	Requests immediate update of algorithm code, variable, or array
:CHANnel (@<*channel*>	Sets dig channel to synch algorithm updates
:WINDow <*num_updates*>	Sets a window for *num_updates* to occur. *RST default is 20
:WINDow?	Returns setting for allowable number variable and algorithm updates.
ARM	
[:IMMediate]	Arm if ARM:SOUR is BUS or HOLD (software ARM)
:SOURce BUS | EXT | HOLD | IMM | SCP | TTLTrg<n>	Specify the source of Trigger Timer ARM
:SOURce?	Return current ARM source
CALibration	
:CONFigure	Prepare to measure on-board references with an external multimeter
:RESistance	Configure to measure reference resistor
:VOLTage <*range*>, ZERO | FSCale	Configure to measure reference voltage range at zero or full scale
:REMote?	Calibrates Remote Signal Conditioning Units
:DATA	Sends RSC cal constants from CAL:REM:DATA? back to HP E1422A
:DATA?	Queries HP E1422A for all remote cal constants
:STORe	Copies RSCU calibration constants from working ram to non-volatile memory
:SETup	Performs Channel Calibration procedure
:SETup?	Returns state of CAL:SETup operation (returns error codes or 0 for OK)

SCPI Command Quick Reference	
Command	**Description**
CALibration (cont.)	
:STORe ADC | TARE	Store cal constants to non-volatile Flash RAM for either A/D calibration or those generated by the CAL:TARE command
:TARE (@<*ch_list*>)	Calibrate out system field wiring offsets
:RESet	Resets cal constants from CAL:TARE back to zero for all channels
:TARE?	Returns state of CAL:TARE operation (returns error codes or 0 for OK)
:VALue	
:RESistance <*ref_ohms*>	Send to instrument the value of just measured reference resistor
:VOLTage <*ref_volts*>	Send to instrument the value of just measured voltage reference
:ZERO?	Correct A/D for short term offset drift (returns error codes or 0 for OK)
DIAGnostic	
:CALibration	
:SETup	
[:MODE] 0 | 1	Set analog DAC output SCP calibration mode
[:MODE]?	Return current setting of DAC calibration mode
:TARe	
[:OTD]	
[:MODE] 0 | 1	Set mode to control OTD current during tare calibration
[:MODE]?	Return current setting of OTD control during tare calibration
:CHECksum?	Perform checksum on Flash RAM and return a '1' for OK, a '0' for corrupted or deleted memory contents
:CONNect <*source*>,<*mode*>,(@<*ch_list*>)	connect HP E1529A channels to measure internal values for verification
:CUSTom	
:MXB <*slope*>,<*offset*>,(@<*ch_list*>)	Generates and loads linear custom EU table
:PIECewise <*table_ad_range*>,<*table_block*>,(@<*ch_list*>)	Loads piecewise custom EU table
:REFerence:TEMPerature	Puts the contents of the Reference Temperature Register into the FIFO
:INTerrupt[:LINe] <*intr_line*>	Sets the VXIbus interrupt line the module will use
:INTerrupt[:LINe]?	Returns the VXIbus interrupt line the module is using
:OTDetect[:STATe] ON | OFF, (@<*ch_list*>)	Controls "Open Transducer Detect" on SCPs contained in <*ch_list*>
:OTDetect[:STATe]? (@<*channel*>)	Returns current state of OTD on SCP containing <*channel*>
:QUERy	
:SCPREAD? <*reg_addr*>	Returns value from an SCP register
:TEST?	
:REMote	
:NUMber? <*test_num*>,<*iterations*>,(@<*channel*>)	Performs single selected self-test on RSCU a selected number of times
:SELFtest? (@<*channel*>)	Performs complete self-test on Remote Signal Conditioning Units
:VERSion?	Returns manufacturer, model, serial#, flash revision #, and date e.g. HEWLETT-PACKARD,E1422B,US34000478,A.04.00, Wed Jul 08 11:06:22 MDT 1994
FETCh?	Return readings stored in VME Memory (format set by FORM cmd)
FORMat	
[:DATA] <*format*>[, <*size*>]	Set format for response data from [SENSe:]DATA?
ASCii[, 7]	Seven bit ASCII format (not as fast as 32-bit because of conversion)
PACKed[, 64]	Same as REAL, 64 except NaN, +INF, and -INF formated for HP BASIC
REAL[, 32]	IEEE 32-bit floating point (requires no conversion so is fastest)
REAL, 64	IEEE 64-bit floating point (not as fast as 32-bit because of conversion)
[:DATA]?	Returns format: REAL, +32 | REAL, +64 | PACK, +64 | ASC, +7

SCPI Command Quick Reference	
Command	**Description**
INITiate	
[:IMMediate]	Put module in Waiting for Trigger state (ready to make one scan)
INPut	
:FILTer	Control filter Signal Conditioning Plug-ons
[:LPASs]	
:FREQuency <*cutoff_freq*>,(@<*ch_list*>)	Sets the cutoff frequency for active filter SCPs
:FREQuency? (@<*channel*>)	Returns the cutoff frequency for the channel specified
[:STATe] ON | OFF, (@<*channel*>)	Turn filtering OFF (pass through) or ON (filter)
[:STATe]? (@<*channel*>)	Return state of SCP filters
:GAIN <*chan_gain*>,(@<*ch_list*>)	Set gain for amplifier-per-channel SCP
:GAIN? (@<*channel*>)	Returns the channel's gain setting
:LOW <*wvolt_type*>,(@<*ch_list*>)	Controls the connection of input LO on a Strain Bridge (Opt. 21 SCP)
:LOW? (@<*channel*>)	Returns the LO connection for the Strain Bridge at *channel*
:POLarity NORmal | INVerted,(@<*ch_list*>)	Sets input polarity on a digital SCP channel
:POLarity? (@<*channel*>)	Returns digital polarity currently set for <*channel*
MEASure	MEAS performs automatic set-up and measurement scanning:
:VOLTage	return value in volts;
:EXCitation? (@<*ch_list*>)	measure excitation voltage at strain bridge, send to EU conversion
:UNSTrained? (@<*ch_list*>)	measure unstrained bridge output voltage, send to EU conversion
MEMory	
:VME	
:ADDRess <*mem_address*>	Specify address of VME memory card to be used as reading storage
:ADDRess?	Returns address of VME memory card
:SIZE <*mem_size*>	Specify number of bytes of VME memory to be used to store readings
:SIZE?	Returns number of VME memory bytes allocate to reading storage
:STATe 1 | 0 | ON | OFF	Enable or disable reading storage in VME memory at INIT
:STATe?	Returns state of VME memory, 1=enabled, 0=disabled
OUTPut	
:CURRent	
:AMPLitude <*amplitude*>,(@<*ch_list*>)	Set amplitude of Current Source SCP channels
:AMPLitude? (@<*channel*>)	Returns the setting of the Current Source SCP channel
:STATe ON | OFF,(@<*ch_list*>)	Enable or disable the Current Source SCP channels
:STATe? (@<*channel*>)	Returns the state of the Current Source SCP channel
:POLarity NORmal | INVerted,(@<*ch_list*>)	Sets output polarity on a digital SCP channel
:POLarity? (@<*channel*>)	Returns digital polarity currently set for <*channel*>
:SHUNt ON | OFF,(@<*ch_list*>)	Adds shunt resistance to leg of Bridge Completion SCP channels
:SOURce INTernal | EXTernal,(@<*ch_list*>)	Selects either the E1529's internal shunt resistor, or an external shunt resistor
:SHUNt? (@<*channel*>)	Returns the state of the shunt resistor on Bridge Completion SCP channel
:TTLTrg	
:SOURce FTRigger | LIMit | SCPlugon | TRIGger	Sets the internal trigger source that can drive the VXIbus TTLTrg lines
:SOURce?	Returns the source of TTLTrg drive.
:TTLTrg<n>	
[:STATe] ON | OFF	When module triggered, source a VXIbus trigger on TTLTrg<n>
[:STATe]?	Returns whether the TTL trigger line specified by n is enabled
:TYPE PASSive | ACTive,(@<*ch_list*>)	sets the output drive type for a digital channel
:TYPE? (@<*channel*>)	Returns the output drive type for <*channel*>
:VOLTage	
:AMPLitude <*amplitude*>,(@<*ch_list*>)	Sets the voltage amplitude on Voltage Output and Strain SCPs
:AMPLitude? (@<*channel*>)	Returns the voltage amplitude setting

SCPI Command Quick Reference	
Command	**Description**
ROUTe	
:SEQuence	
:DEFine (@<*ch_list*>)	Defines the analog scan list.
:DEFine? AIN | AOUT | DIN | DOUT | DEST	Returns comma separated list of channels in analog I|O, dig I|O ch lists. For DEST, returns the data destination for each AIN channel; 0=none, 1=CVT, 2=FIFO, 3=CVT&FIFO, -1=set by algorithm (writefifo, writecvt, writeboth)
:POINts? AIN | AOUT | DIN | DOUT	Returns number of channels defined in above lists.
SAMPle	
:TIMer <*num_samples*>,(@<*ch_list*>)	sets the time interval between channel measurements
:TIMer? (@<*channel*>)	Returns the time interval between channel measurements
[SENSe:]	
CHANnel	
:SETTling <settle_time>,(@<*ch_list*>)	Sets the channel settling time for channels in *ch_list*
:SETTling? (@<*channel*>)	Returns the channel settling time for *channel*
DATA	
:CVTable? (@<*ch_list*>)	Returns elements of Current Value Table specified by *ch_list*
:RESet	Resets all entries in the Current Value Table to IEEE "Not-a-number"
:FIFO	
[:ALL]?	Fetch all readings until instrument returns to trigger idle state
:COUNt?	Returns the number of measurements in the FIFO buffer
:HALF?	Returns 1 if at least 32,768 readings are in FIFO, else returns 0
:HALF?	Fetch 32,768 readings (half the FIFO) when available
:MODE BLOCK | OVERwrite	Set FIFO mode.
:MODE?	Return the currently set FIFO mode
:PART? <*n_readings*>	Fetch *n_readings* from FIFO reading buffer when available
:RESet	Reset the FIFO counter to 0
FREQuency	
:APERture <*gate_time*>,(@<*ch_list*>)	Sets the gate time for frequency counting
:APERture? (@<*channel*>)	Returns the gate time set for frequency counting
FUNCtion	Equate a function and range with groups of channels
:CONDition (@<*ch_list*>)	Sets function to sense digital state
:CUSTom [<*range*>,](@<*ch_list*>)	Links channels to custom EU conversion table loaded by DIAG:CUST:MXB or DIAG:CUST:PIEC commands
:REFerence [<*range*>,](@<*ch_list*>)	Links channels to custom reference temperature EU conversion table loaded by DIAG:CUST:PIEC commands
:TC <*type*>,[<*range*>,](@<*ch_list*>)	Links channels to custom temperature EU conversion table loaded by DIAG:CUST:PIEC, and performs ref temp compensation for <*type*>
:FREQuency (@<*ch_list*>)	Configure channels to measure frequency
:RESistance <*excite_current*>,[<*range*>,](@<*ch_list*>)	Configure channels to sense resistance measurements
:STRain	Links measurement channels as having read bridge voltage from:
:FBENding [<*range*>,](@<*ch_list*>)	Full BENding
:FBPoisson [<*range*>,](@<*ch_list*>)	Full Bending Poisson
:FPOisson [<*range*>,](@<*ch_list*>)	Full POisson
:HBENding [<*range*>,](@<*ch_list*>)	Half BENding
:HPOisson [<*range*>,](@<*ch_list*>)	Half Poisson
[:QUARter] [<*range*>,](@<*ch_list*>)	QUARter
:Q120 [<*range*>,](@<*ch_list*>)	Quarter bridge with the 120Ω resistor selected (HP E1529A only)
:Q350 [<*range*>,](@<*ch_list*>)	Quarter bridge with the 350Ω resistor selected (HP E1529A only)
:USER [<*range*>,](@<*ch_list*>)	Quarter bridge with the user supplied resistor selected (HP E1529A only)

SCPI Command Quick Reference	
Command	**Description**
SENse:FUNCtion (continued)	
RTD, 85 | 92	RTDs
TCouple, CUST | E | EEXT | J | K | N | S | T	thermocouples
THERmistor, 2250 | 5000 | 10000	thermistors
:TEMPerature <*sensor_type*>,<*sub_type*>, [<*range*>,](@<*ch_list*>)	Configure channels for temperature measurement types above: excitation current comes from Current Output SCP.
:TOTalize (@<*ch_list*>)	Configure channels to count digital state transitions
:VOLTage[:DC] [<*range*>,](@<*ch_list*>)	Configure channels for DC voltage measurement
RTD, 85 | 92	RTDs
THERmistor,5000	thermistors
:REFerence <*sensor_type*>,<*sub_type*>,[<*range*>,](@<*ch_list*>)	Configure channel for reference temperature measurements above:
:CHANnels (@<*ref_channel*>),(@<*ch_list*>)	Groups reference temperature channel with TC measurement channels
:TEMPerature <*degrees_c*>	Specifies the temperature of a controlled temperature reference junction
:STRain	
:BRIDge[:TYPE] <*select*>,(@<*ch_list*>)	Set bridge configuration switches on HP E1529
:BRIDge[:TYPE]? (@<*channel*>)	Returns the current bridge configuration setting
:CONNect BRIDge | EXCitation,(@<*ch_list*>)	Set switches to sense bridge output, or excitation voltage on HP E1529A
:CONNect? (@<*channel*>)	Returns the current sense setting for the channel specified
:EXCitation <*excite_v*>,(@<*ch_list*>)	Specifies the Excitation Voltage by channel to the strain EU conversion
:EXCitation? (@<*channel*>)	Returns the Excitation Voltage set for <*channel*>
:STATe ON | OFF,(@<*ch_list*>)	controls state of excitation supply relay in HP E1529A to banks of 8 channels
:STATe? (@<*channel*>)	returns state of excitation supply relay ("ON" | "OFF"
:GFACtor <*gage_factor*>,(@<*ch_list*>)	Specifies the Gage Factor by channel to the strain EU conversion
:GFACtor? (@<*channel*>)	Returns the Gage Factor set for <*channel*>
:POISson <*poisson_ratio*>,(@<*ch_list*>)	Specifies the Poisson Ratio by channel to the strain EU conversion
:POISson? (@<*channel*>)	Returns the Poisson Ratio set for <*channel*>
:UNSTrained <*unstrained_v*>,(@<*ch_list*>)	Specifies the Unstrained Voltage by channel to the strain EU conversion
:UNSTrained? (@<*channel*>)	Returns the Unstrained Voltage set for <*channel*>
SOURce	
:FM	
[:STATe] 1 | 0 | ON | OFF,(@<*ch_list*>)	Configure digital channels to output frequency modulated signal
[:STATe]? (@<*channel*>)	Returns state of channels for FM output
:FUNCtion	
[:SHAPe]	
:CONDition (@<*ch_list*>)	Configures channels to output static digital levels
:PULSe (@<*ch_list*>)	Configures channels to output digital pulse(s)
:SQUare (@<*ch_list*>)	Configures channels to output 50/50 duty cycle digital pulse train
:PULM	
:STATe 1 | 0 | ON | OFF,(@<*ch_list*>)	Configure digital channels to output pulse width modulated signal
:STATe? (@<*channel*>)	Returns state of channels for PW modulated output
:PERiod <*period*>,(@<*ch_list*>)	Sets pulse period for PW modulated signals
:PERiod? (@<*channel*>)	Returns pulse period for PW modulated signals
:WIDTh <*width*>,(@<*ch_list*>)	Sets pulse width for FM modulated signals
:WIDTh? (@<*channel*>)	Returns pulse width setting for FM modulated signals

SCPI Command Quick Reference	
Command	**Description**
STATus	
:OPERation	Operation Status Group: Bit assignments; 0=Calibrating, 4=Measuring, 8=Scan Complete, 10=FIFO Half Full, 11=algorithm interrupt
:CONDition?	Returns state of Operation Status signals
:ENABle <*enable_mask*>	Bits set to 1 enable status events to be summarized into Status Byte
:ENABle?	Returns the decimal weighted sum of bits set in the Enable register
[:EVENt]?	Returns weighted sum of bits that represent Operation status events
STATus:OPERation (continued)	
:NTRansition <*transition_mask*>	Sets mask bits to enable pos. Condeition Reg. transitions to Event reg
:NTRansition?	Returns positive transition mask value
:PTRansition <*transition_mask*>	Sets mask bits to enable neg. Condeition Reg. transitions to Event reg
:PTRansition?	Returns negative transition mask value
:PRESet	Presets both the Operation and Questionable Enable registers to 0
:QUEStionable	Questionable Data Status Group: Bit assignments; 8=Calibration Lost, 9=Trigger Too Fast, 10=FIFO Overflowed, 11=Over voltage, 12=VME Memory Overflow, 13=Setup Changed.
:CONDition?	Returns state of Questionable Status signals
:ENABle <*enable_mask*>	Bits set to 1 enable status events to be summarized into Status Byte
:ENABle?	Returns the decimal weighted sum of bits set in the Enable register
[:EVENt]?	Returns weighted sum of bits that represent Questionable Data events
:NTRansition <*transition_mask*>	Sets mask bits to enable pos. Condeition Reg. transitions to Event reg
:NTRansition?	Returns positive transition mask value
:PTRansition <*transition_mask*>	Sets mask bits to enable neg. Condeition Reg. transitions to Event reg
:PTRansition?	Returns negative transition mask value
SYSTem	
:CTYPe? (@<*channel*>)	Returns the identification of the SCP at *channel*
:ERRor?	Returns one element of the error queue "0" if no errors
:VERSion?	Returns the version of SCPI this instrument complies with
TRIGger	
:COUNt <*trig_count*>	Specify the number of trigger events that will be accepted
:COUNt?	Returns the current trigger count setting
[:IMMediate]	Triggers instrument when TRIG:SOUR is TIMer or HOLD (same as *TRG and IEEE 488.1 GET commands.
:SOURce BUS | EXT | HOLD | IMM | SCP | TIMer | TTLTrg<n>	Specify the source of instrument triggers
:SOURce?	Returns the current trigger source
:TIMer	Sets the interval between scan triggers when TRIG:SOUR is TIMer
[:PERiod] <*trig_interval*>	Sets the interval between scan triggers when TRIG:SOUR is TIMer
[:PERiod]?	Returns setting of trigger timer

IEEE-488.2 Common Command Quick Reference			
Category	**Command**	**Title**	**Description**
Calibration	*CAL?	Calibrate	Performs internal calibration on all 64 channels out to the terminal module connector. Returns error codes or 0 for OK
Internal Operation	*IDN?	Identification	Returns the response: HEWLETT-PACKARD,E1422B,<*serial#*>,<*driver rev#*>
	*RST	Reset	Resets all scan lists to zero length and stops scan triggering. Status registers and output queue are unchanged.
	*TST?	Self-test	Performs self-test. Returns 0 to indicate test passed.
Status Reporting	*CLS	Clear Status	Clears all status event registers and so their status summary bits (except the MAV bit).
	*ESE <*mask*>	Event Status Enable	Set Standard Event Status Enable register bits mask.
	*ESE?	Event Status Enable query	Return current setting of Standard Event Status Enable register.
	*ESR?	Event Status Register query	Return Standard Event Status Register contents.
	*SRE <*mask*>	Service Request Enable	Set Service Request Enable register bit mask.
	*SRE?	Service Request Enable query	Return current setting of the Service Request Enable register.
	*STB?	Read Status Byte query	Return current Status Byte value.
Macros	*DMC <*name*>,<*cmd_data*>	Define Macro Command	Assigns one, or a sequence of commands to a macro.
	*EMC 1 | 0	Enable Macro Command	Enable/Disable defined macro commands.
	*EMC?	Enable Macros query	Returns 1 for macros enabled, 0 for disabled.
	*GMC? <*name*>	Get Macro query	Returns command sequence for named macro.
	*LMC?	Learn Macro query	Returns comma-separated list of defined macro names
	*PMC	Purge Macro Commands	Purges all macro commands
	*RMC <*name*>	Remove Individual Macro	Removes named macro command.
Synchronization	*OPC	Operation Complete	Standard Event register's Operation Complete bit will be 1 when all pending device operations have been finished.
	*OPC?	Operation Complete query	Places an ASCII 1 in the output queue when all pending operations have finished.
	*TRG	Trigger	Trigger s module when TRIG:SOUR is HOLD.
	*WAI	Wait to Complete	

Notes:

Appendix A

Specifications

HP E1422 Specifications

Power Requirements (with no SCPs installed)	+5V		+12V		-12V		+24V		-24V		-5.2V	
IPm=Peak Module Current	IPm	IDm	IPm	IDm	IPm	IDm	IPm	IDm	IPm	IDm	IPm	IDm
IDm=Dynamic Module Current	1.0	0.02	0.06	0.01	0.01	0.01	0.1	0.01	0.1	0.01	0.15	0.01

Cooling Requirements	Average Watts/Slot	ΔPressure (mmH_2O)	Air Flow (liters/s)
	14	0.08	0.08

Power Available for SCPs (See VXI Catalog or SCP manuals for SCP current)	1.0A ±24V, 3.5A 5V

Measurement ranges	
DC Volts	(HP E1501 or HP E1502) ±62.5mV to ±16V Full Scale
Temperature	Thermocouples - -200 to +1700 °C Thermistors - (Opt 15 required) -80 to +160 °C RTD's - (Opt 15 required) -200 to +850 °C
Resistance	(HP E1505 with HP E1501) 512 ohms to 131 Kohms FS
Strain	25,000 µe or limit of linear range of strain gage

Measurement Resolution	16 bits (including sign)

Maximum Update Rate (running PIDA algorithms)		
	1 Algorithm	2.5 KHz
	8 Algorithms	1 KHz
	32 Algorithms	250 Hz

Trigger Timer and Sample Timer Accuracy	100ppm (.01%) from -10 °C to +70 °C

External Trigger Input

TTL compatible input. Negative true edge triggered except first trigger will occur if external trigger input is held low when module is INITiated. Minimum pulse width 100nS. Since each trigger starts a complete scan of 2 or more channel readings, maximum trigger rate depends on module configuration.

Maximum input voltage
(Normal mode plus common mode)

With Direct Input, Passive Filter, or Amplifier SCPs:
Operating: < ±16 V peak Damage level: >±42 V peak
With HP E1513 Divide by 16 Attenuator SCP:
Operating: <±60 VDC, <±42 V peak

Maximum common mode voltage

With Direct Input, Passive Filter, or Amplifier SCPs:
Operating: <±16 V peak Damage level: >±42 V peak
With HP E1513 Divide by 16 Attenuator SCP:
Operating: <±60 VDC, <±42 V peak

Common mode rejection

0 to 60Hz -105dB

Input impedance

greater than 90 MOhm differential
(1 M Ohm with HP E1513 Attenuator)

On-board Current Source

122 µA ±0.02%, with ±17 Volts Compliance

Maximum tare cal. offset

SCP Gain = 1 (Maximum tare offset depends on A/D range and SCP gain)

A/D range ±V F.Scale	16	4	1	0.25	0.0625
Max Offset	3.2213	.82101	.23061	.07581	.03792

The following specifications reflect the performance of the HP E1422 with the HP E1501 Direct Input Signal Conditioning Plug-on. The performance of the HP E1422 with other SCPs is found in the Specifications section of that SCP's manual.

Measurement accuracy DC Volts

(90 days) 23°C ±1°C (with *CAL? done after 1 hr warm up and CAL:ZERO? within 5 min.).
NOTE: If autoranging is ON:
for readings <3.8V, add ±.02% to linearity specifications.
for readings ≥3.8V, add ±.05% to linearity specifications.

A/D range ±V F. Scale	Linearity % of Reading	Offset Error	Noise 3 sigma	Noise* 3 sigma
.0625	0.01%	5.3µV	18µ	8µV
.25	0.01%	10.3µV	45µV	24µV
1	0.01%	31µV	110µV	90µV
4	0.01%	122µV	450µV	366µV
16	0.01%	488µV	1.8 mV	1.5 mV

Temperature Coefficient: Gain - 10ppm/°C. Offset - (0 - 40°C) .14µV/°C, (40 - 55°C) .8µV+.38µV/°C

Temperature Accuracy

The following pages have temperature accuracy graphs that include instrument and firmware linearization errors. The linearization algorithm used is based on the ITS-90 standard transducer curves. Add your transducer accuracy to determine total measurement error.

The thermocouple graphs on the following pages include only the errors due to measuring the voltage output of the thermocouple, as well as the algorithm errors due to converting the thermocouple voltage to temperature. To this error must be added the error due to measuring the reference junction temperature with an RTD or a 5K thermistor. See the graphs for the RTD or the 5K thermistor to determine this additional error. Also, the errors due to gradients across the isothermal reference must be added. If an external isothermal reference panel is used, consult the manufacturer's specifications. If HP termination blocks are used as the isothermal reference, see the notes below.

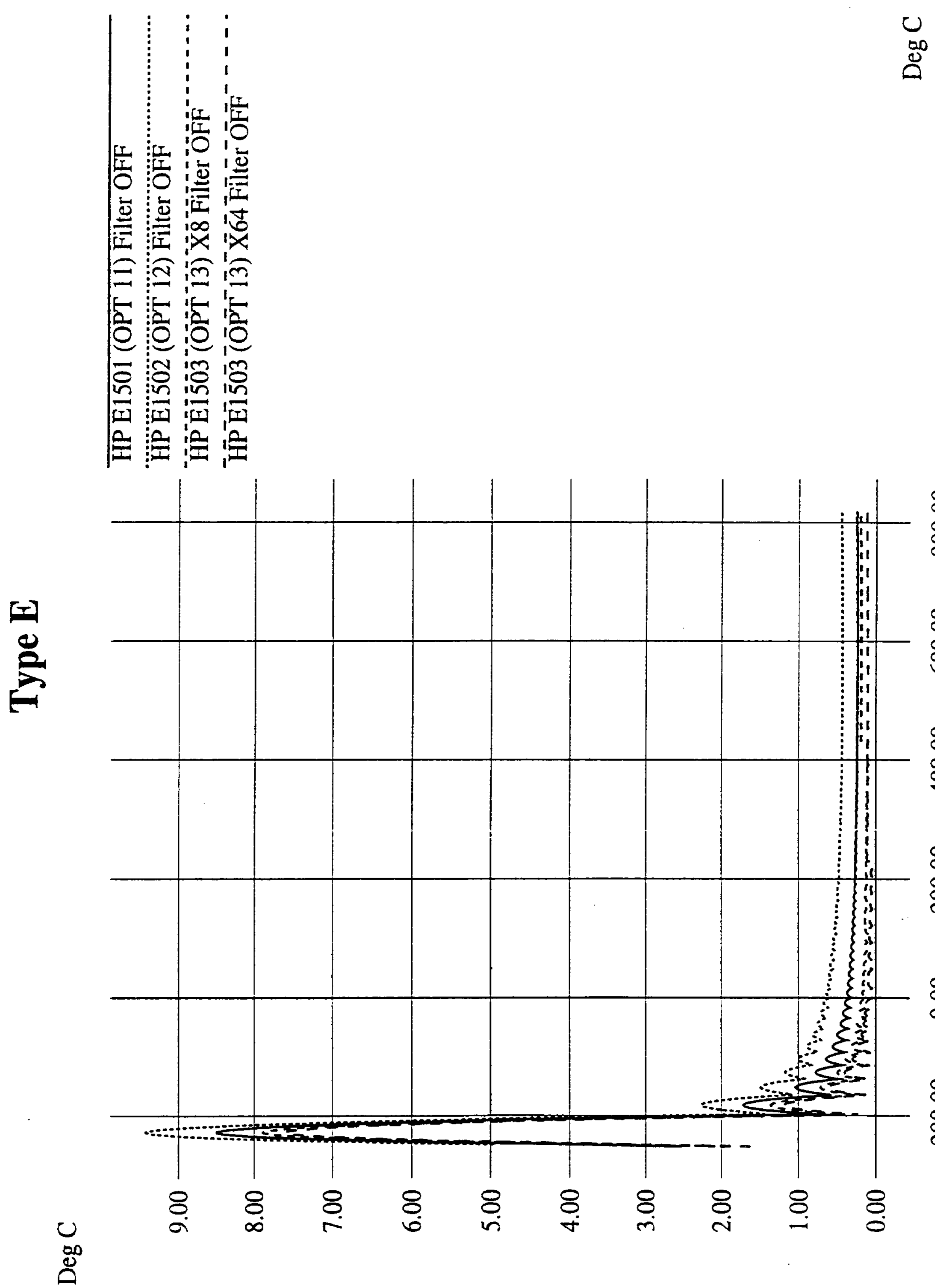
NOTES

1) When using the Terminal Module as the isothermal reference, add ±0.6°C to the thermocouple accuracy specs to account for temperature gradients across the Terminal Module. The ambient temperature of the air surrounding the Terminal Module must be within ±2°C of the temperature of the inlet cooling air to the VXI mainframe.

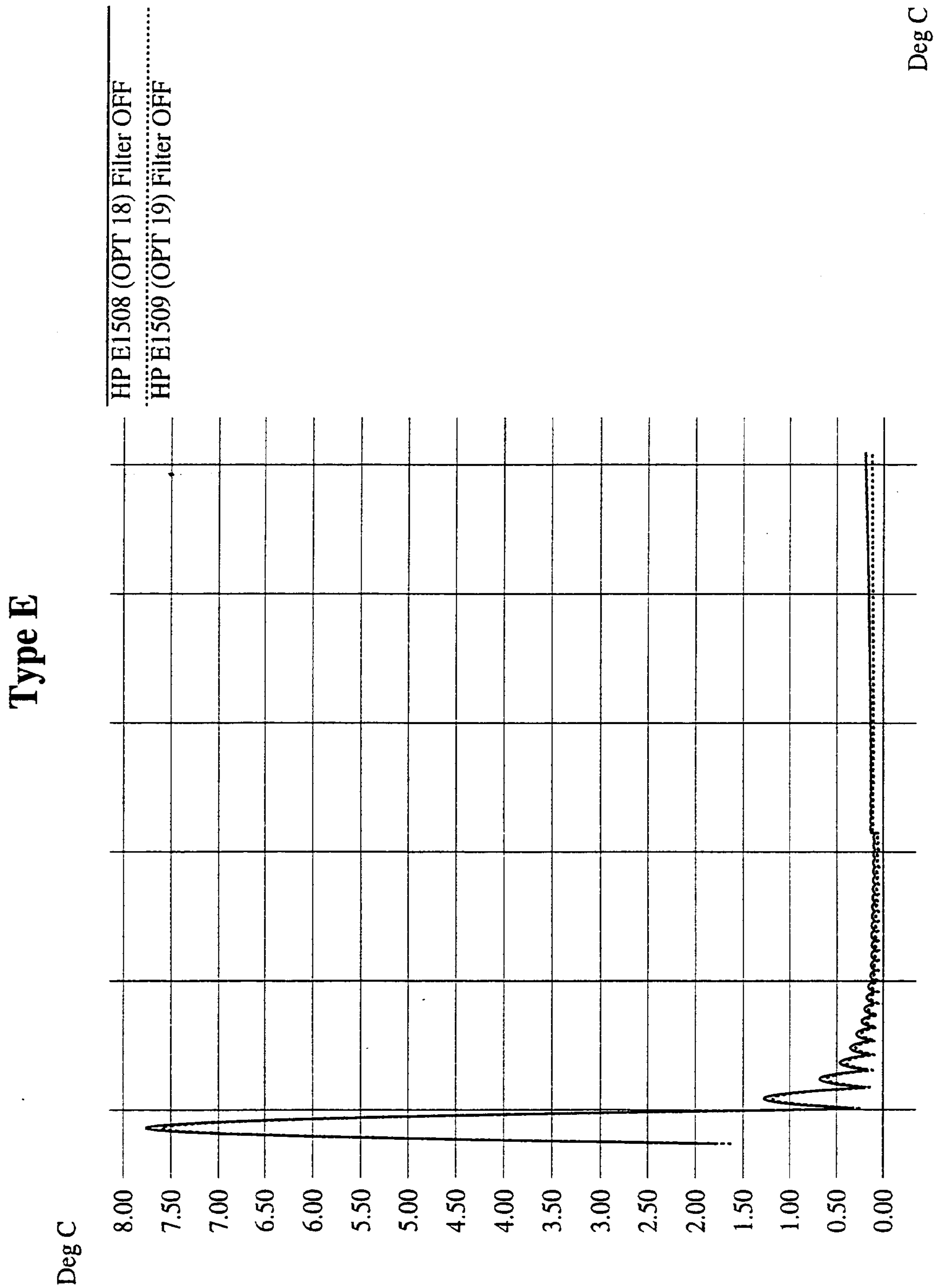
2) When using the HP E1586 Rack-Mount Terminal Panel as the isothermal reference, add ±0.2°C to the thermocouple accuracy specs to account for temperature gradients across the HP E1586. The HP E1586A should be mounted in the bottom part of the rack, below and away from other heat sources for best performance.

The temperature specification graphs are found on the following pages:

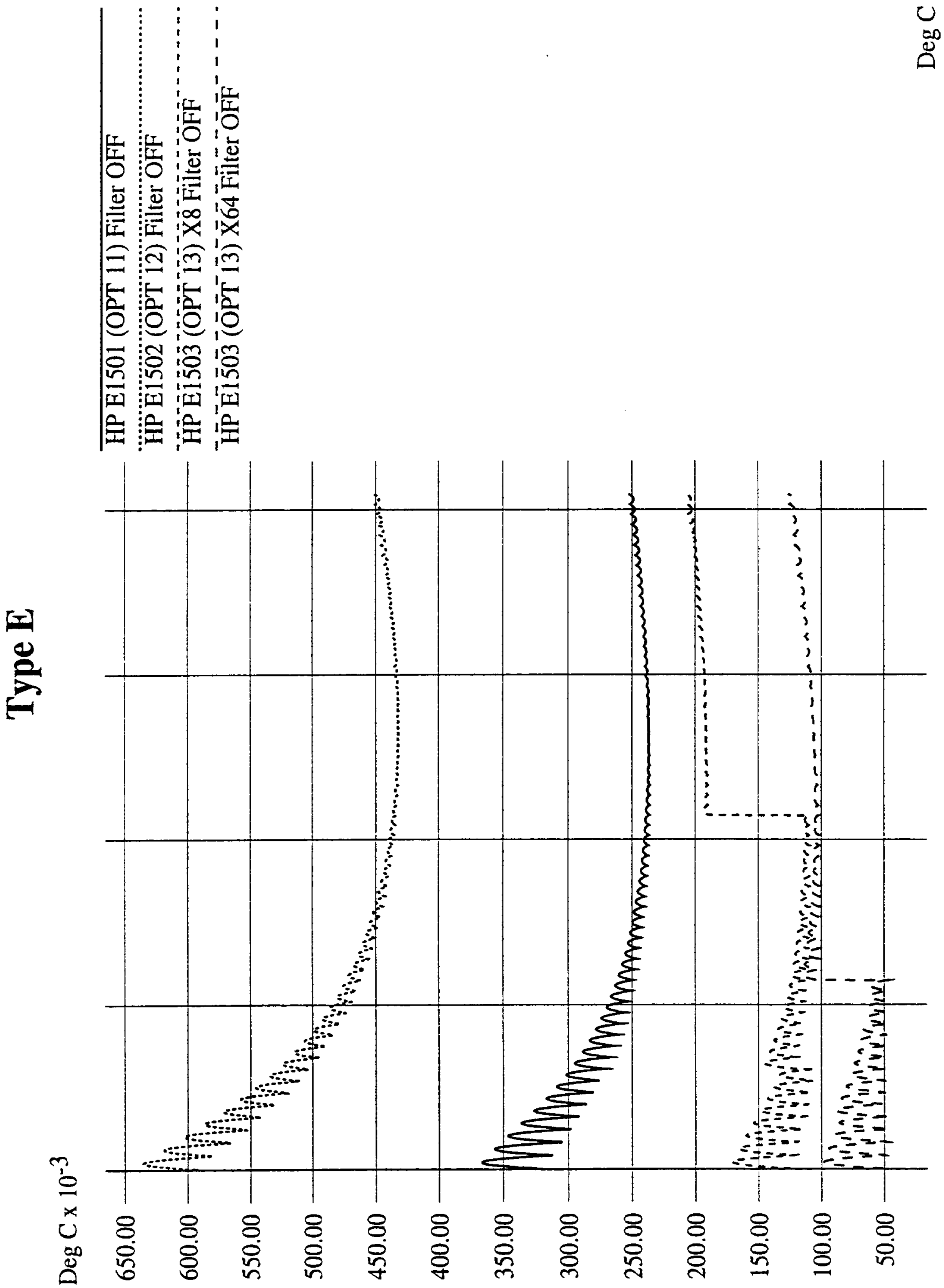
Thermocouple Type E (-200-800C), SCPs HP E1501,02,03



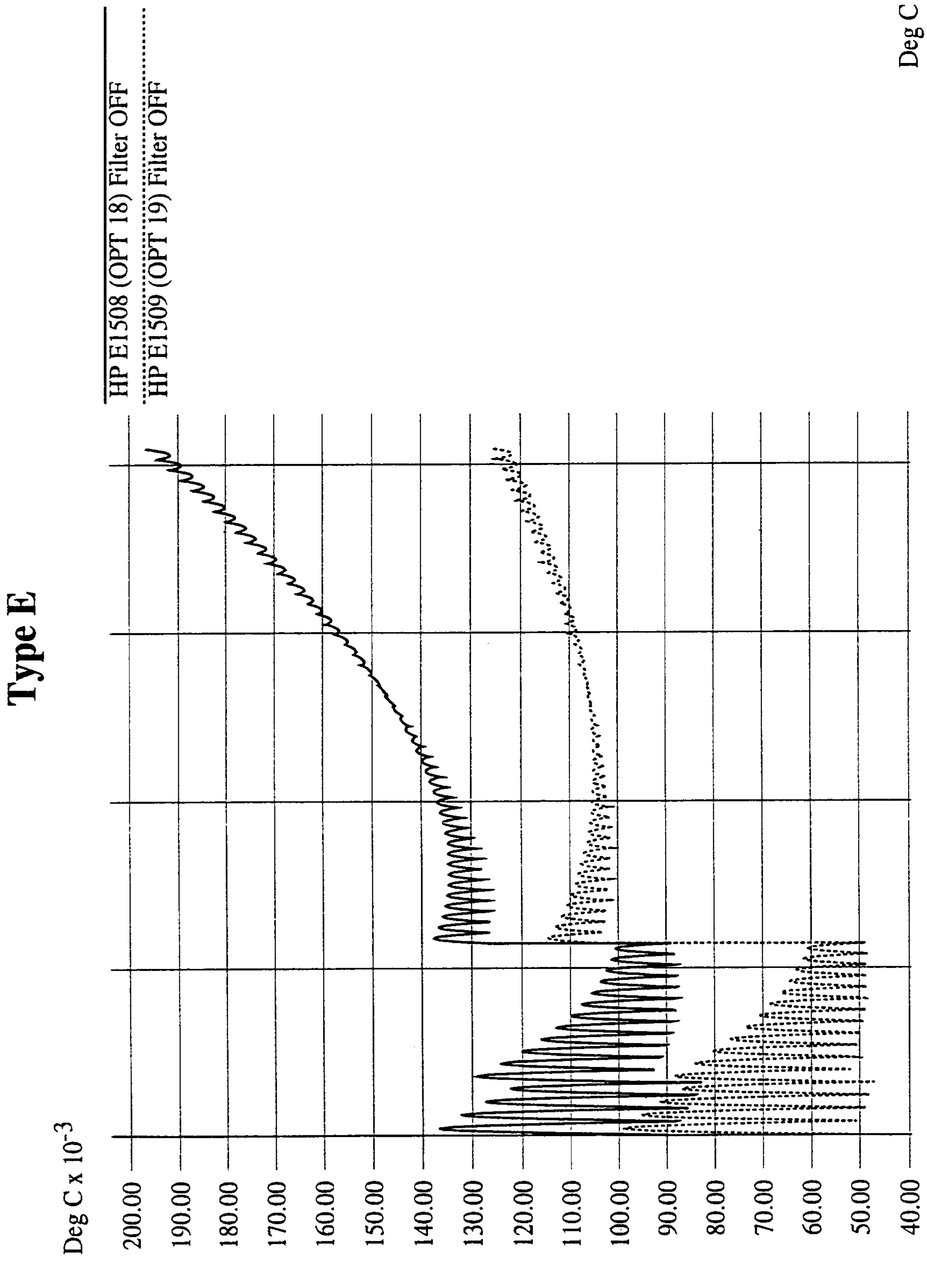
Thermocouple Type E (-200-800C), SCPs HP E1508,09



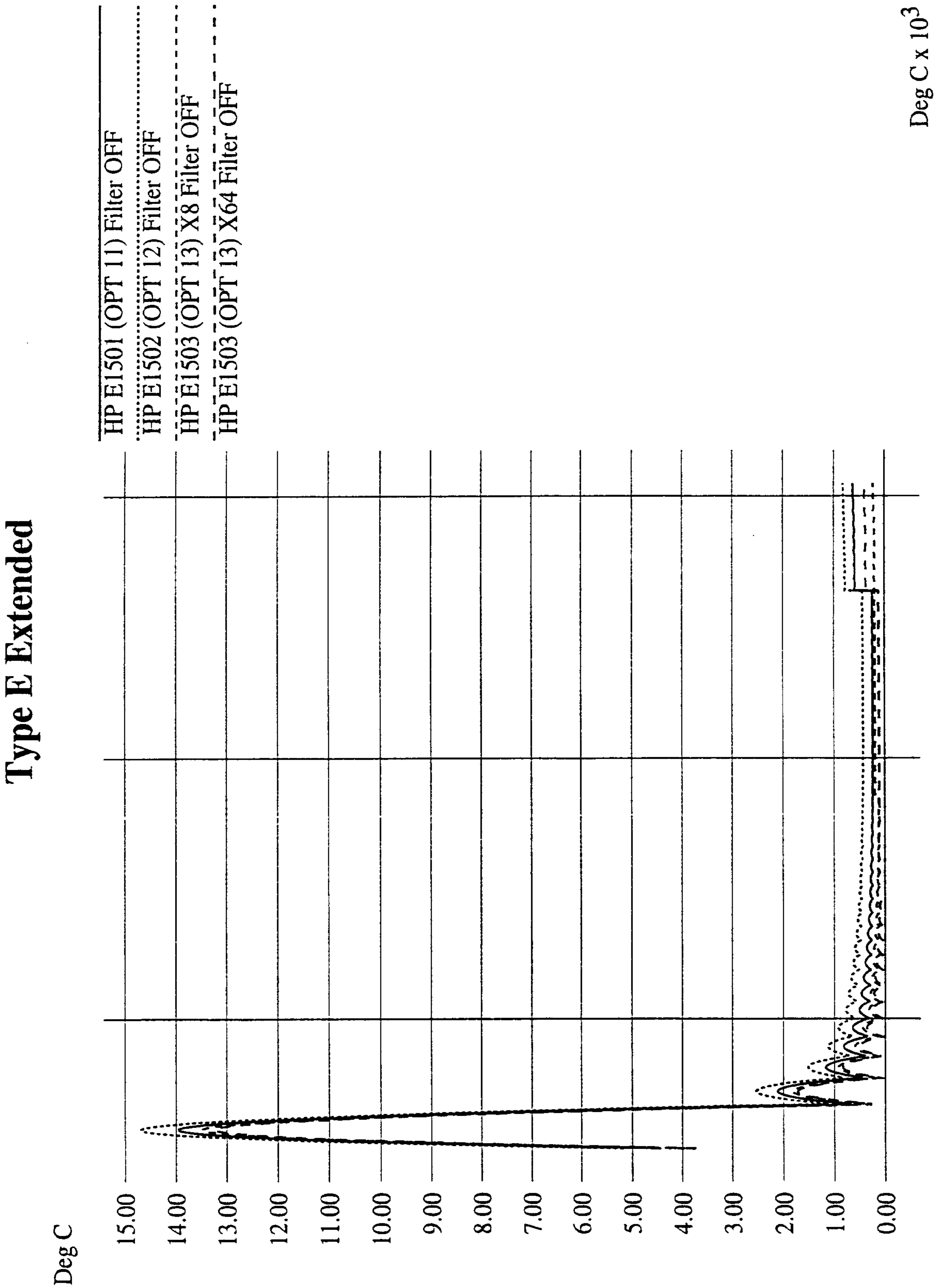
:Thermocouple Type E (0-800C), SCPs HP E1501,02,03



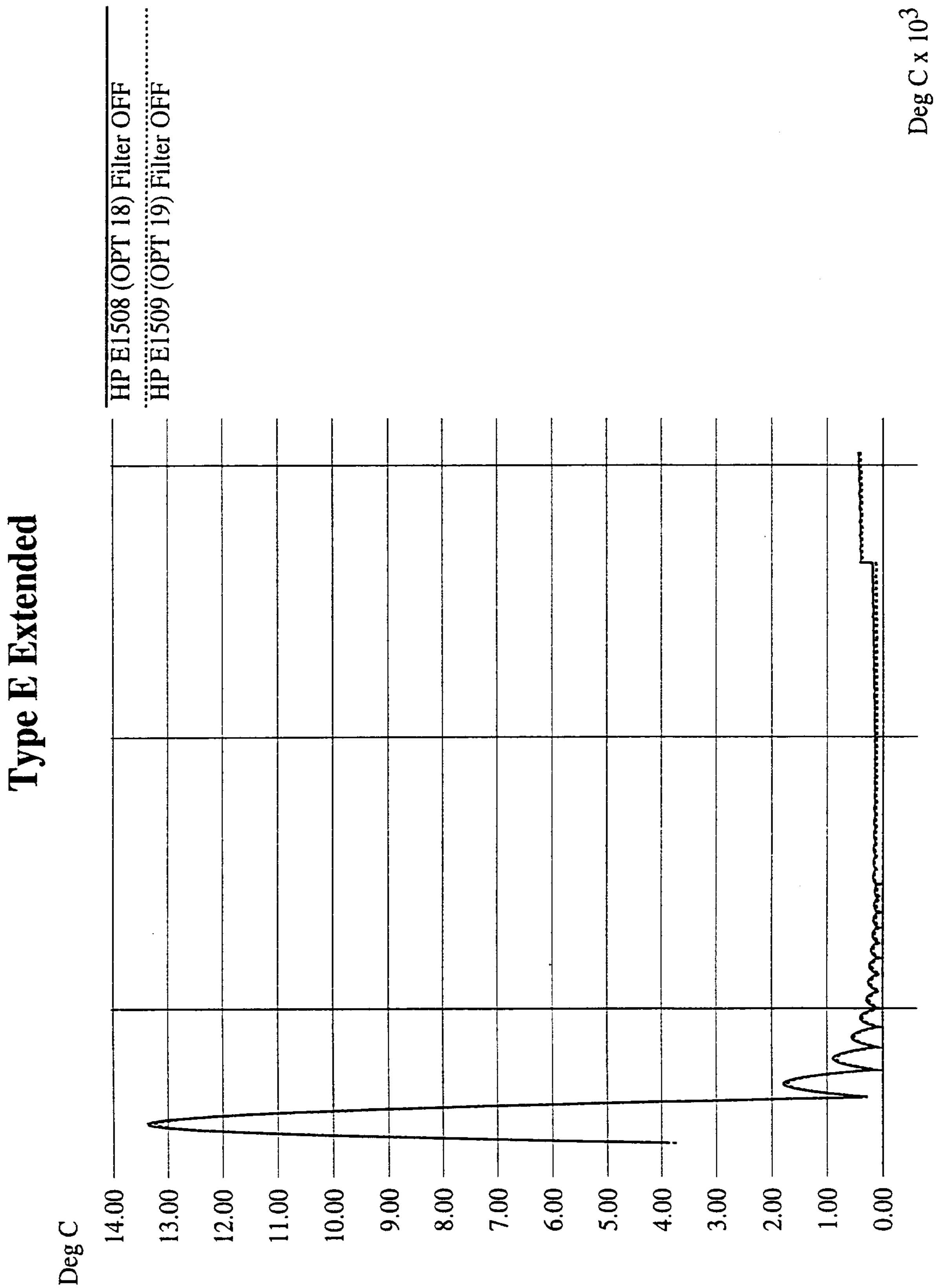
Thermocouple Type E (0-800C), SCPs HP E1509,09



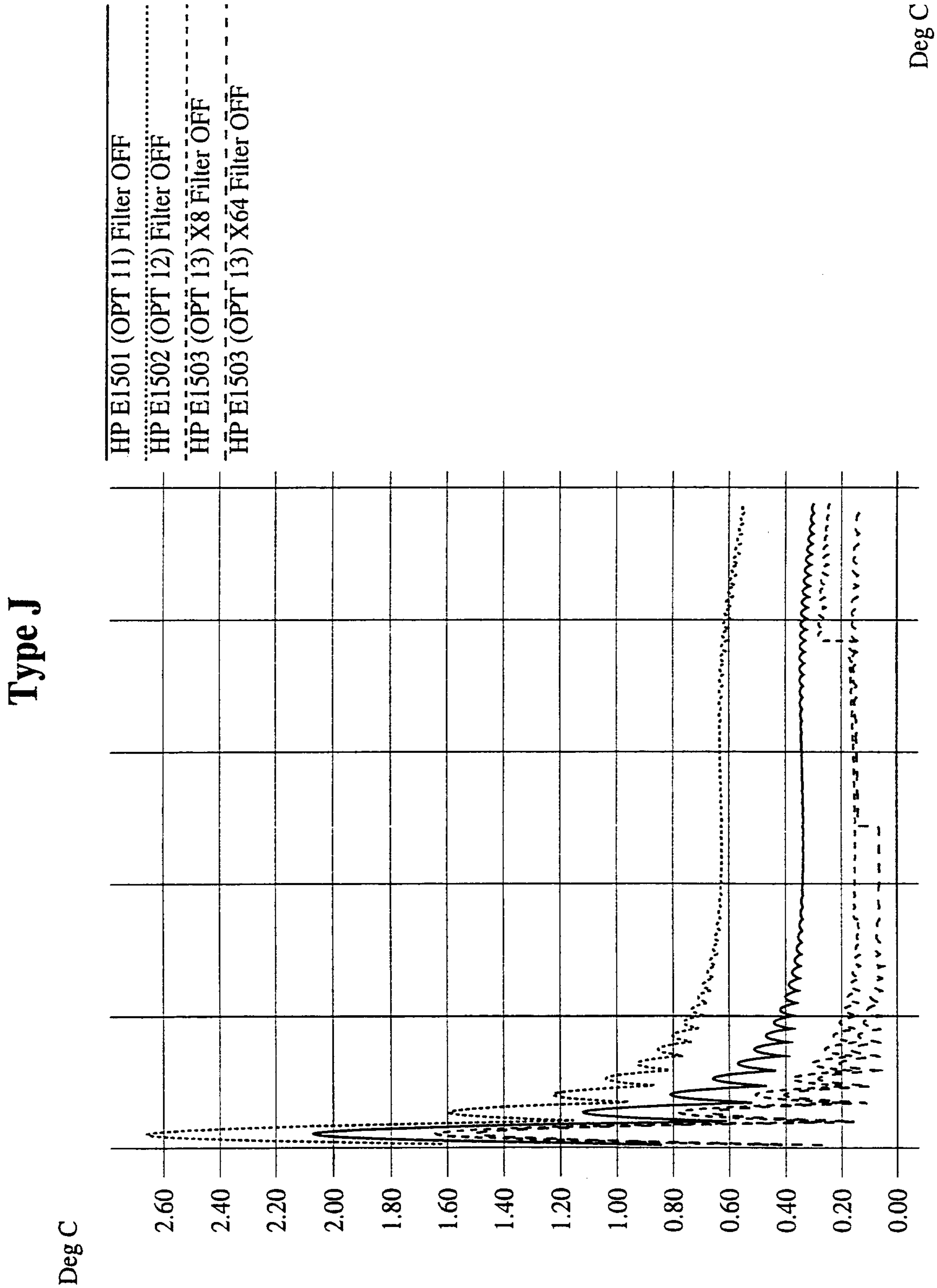
Thermocouple Type E Extended, SCPs HP E1501,02,03

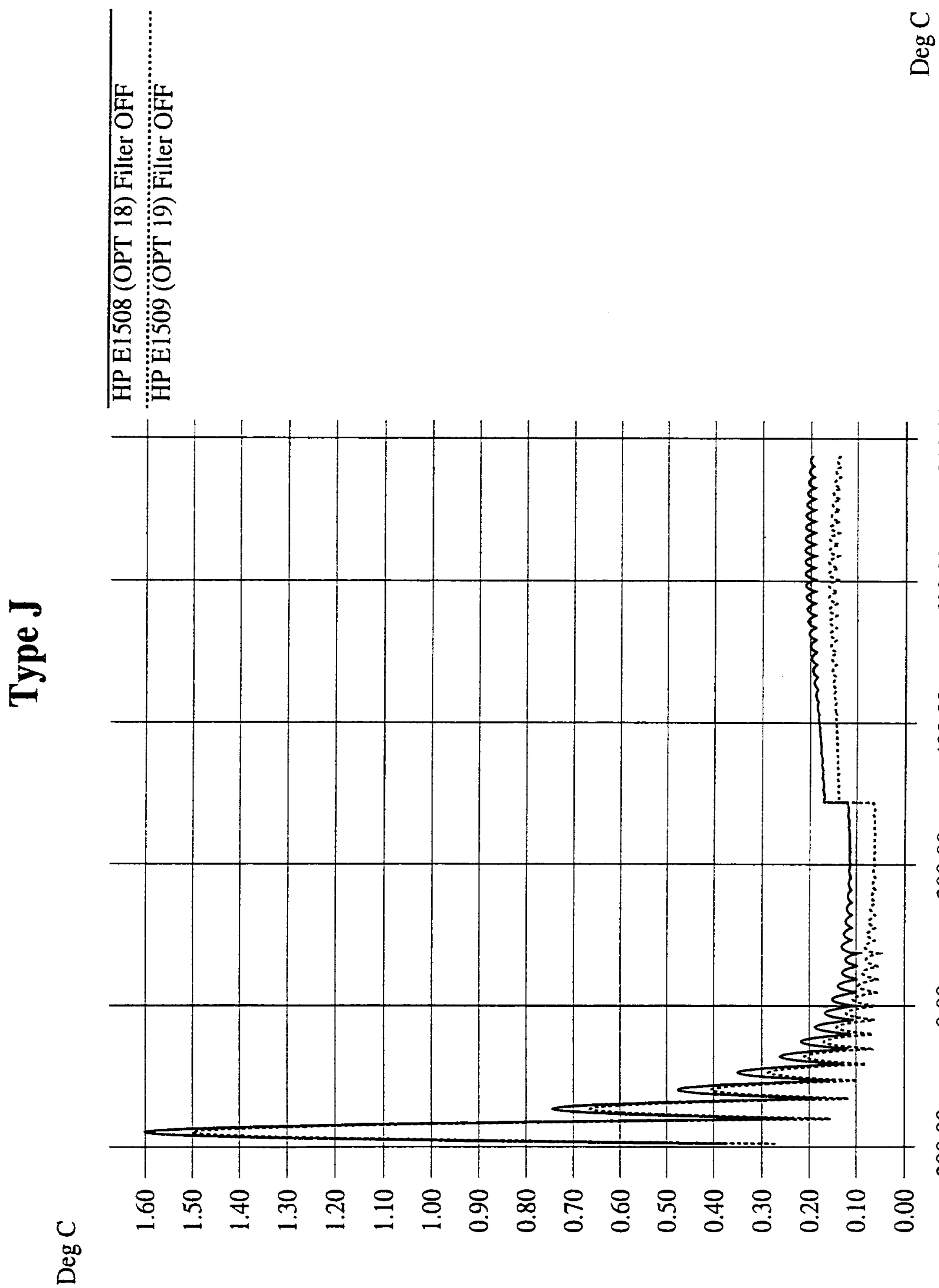


Thermocouple Type E Extended, SCPs HP E1508,09

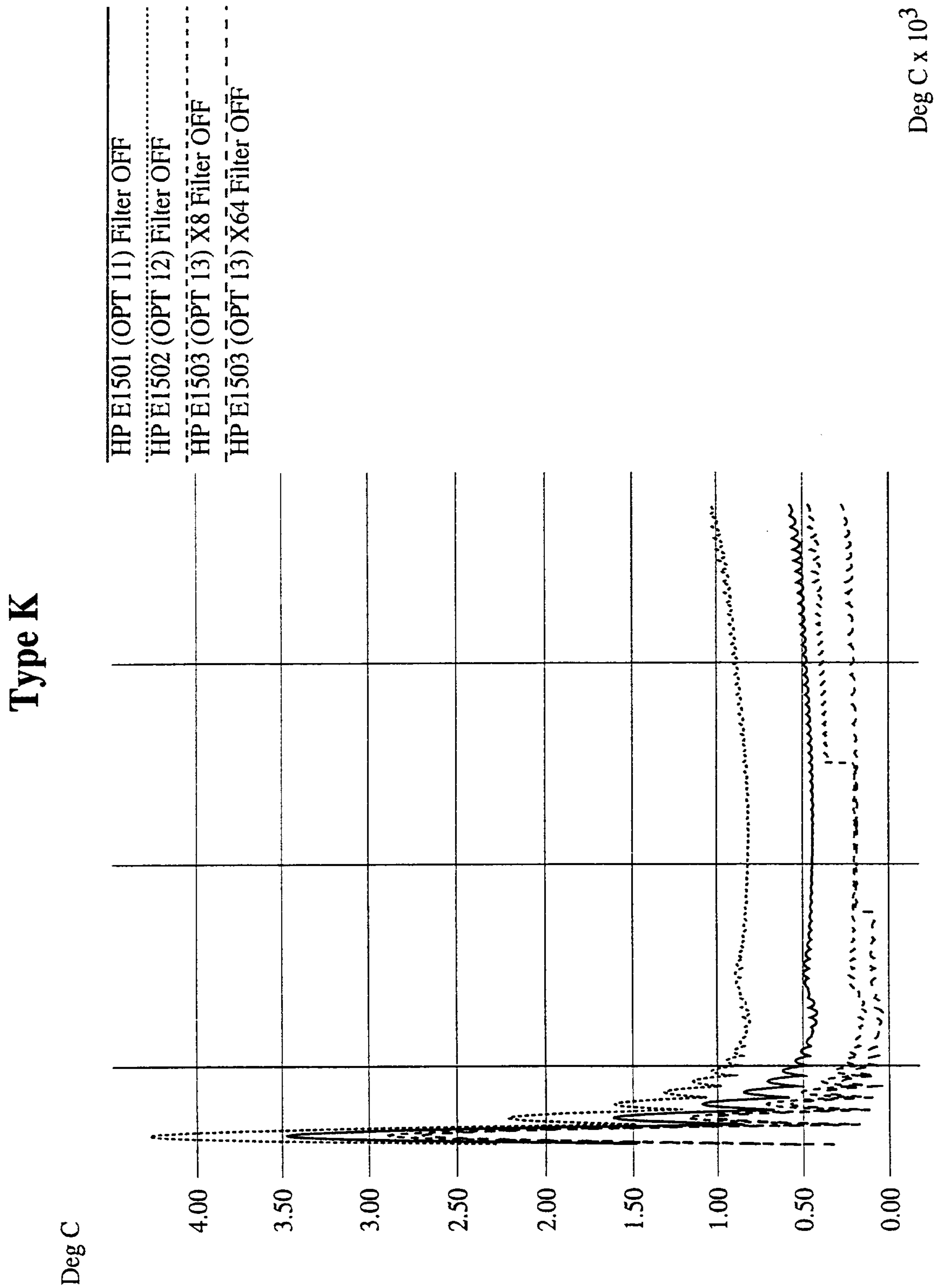


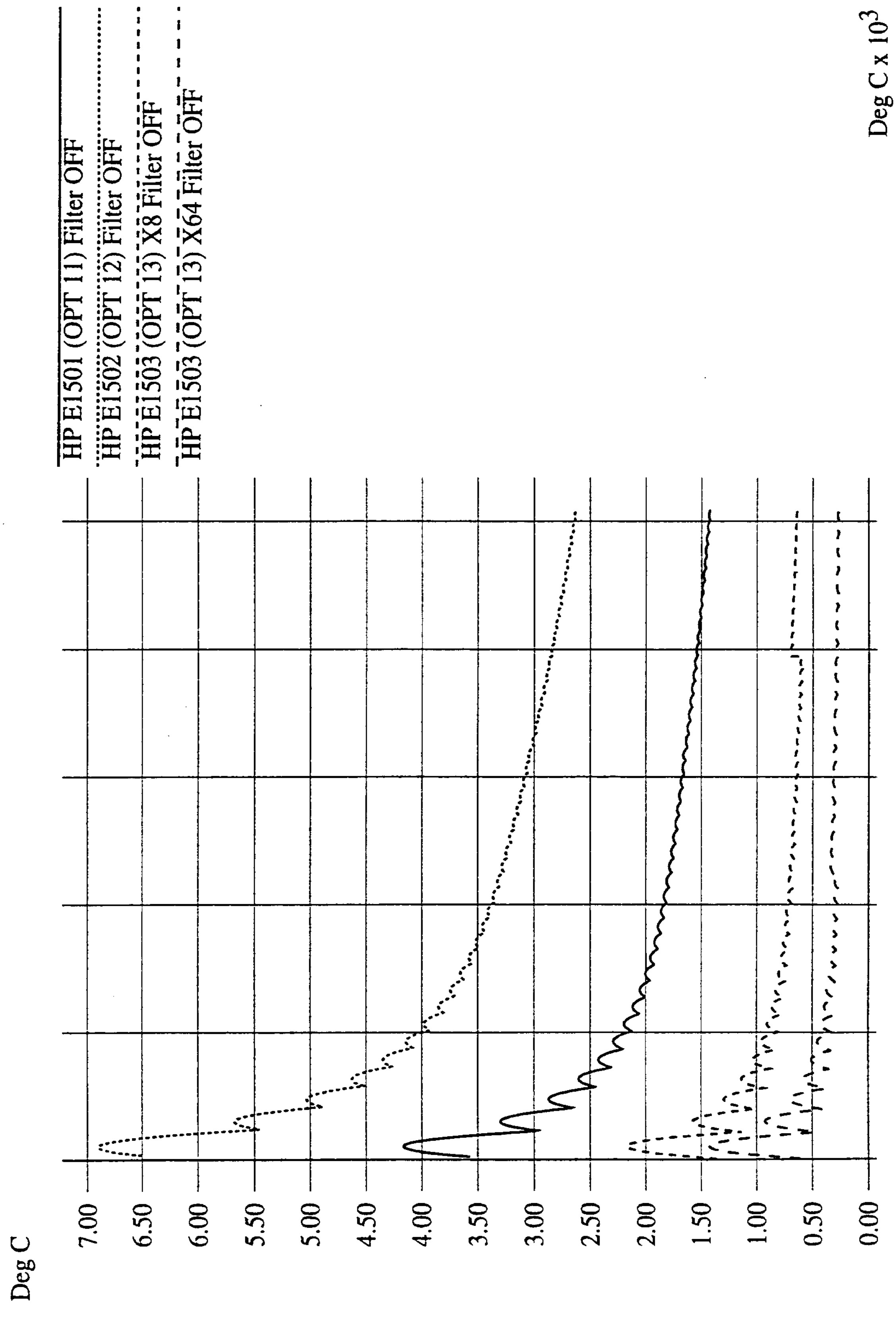
Thermocouple Type J, SCPs HP E1501,02,03



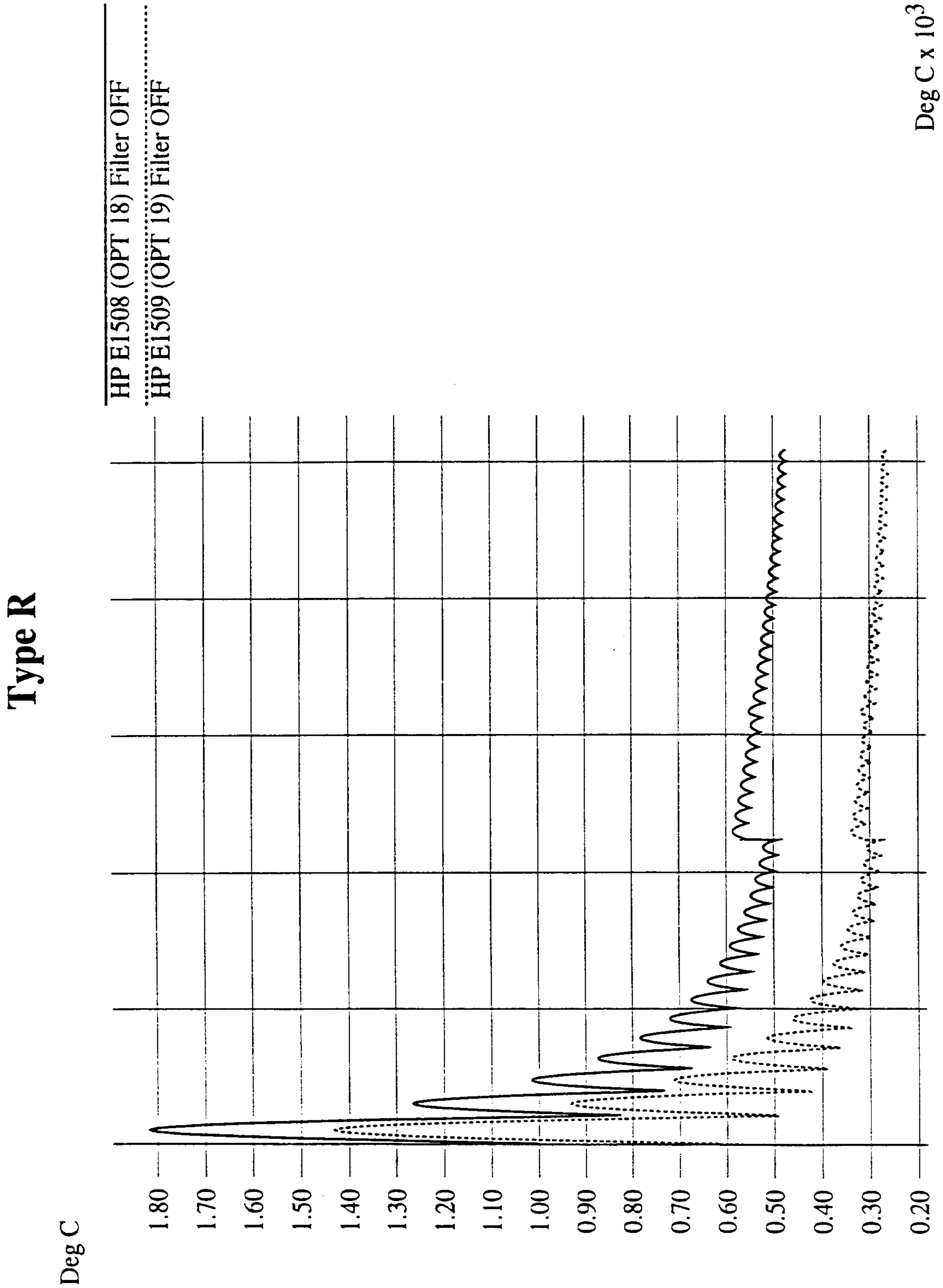

Type J
HP E1508 (OPT 18) Filter OFF
HP E1509 (OPT 19) Filter OFF
Deg C
1.60
1.50
1.40
1.30
1.20
1.10
1.00
0.90
0.80
0.70
0.60
0.50
0.40
0.30
0.20
0.10
0.00
-200.00
0.00
200.00
400.00
600.00
800.00
Deg C

Thermocouple Type K, SCPs HP E1501,02,03

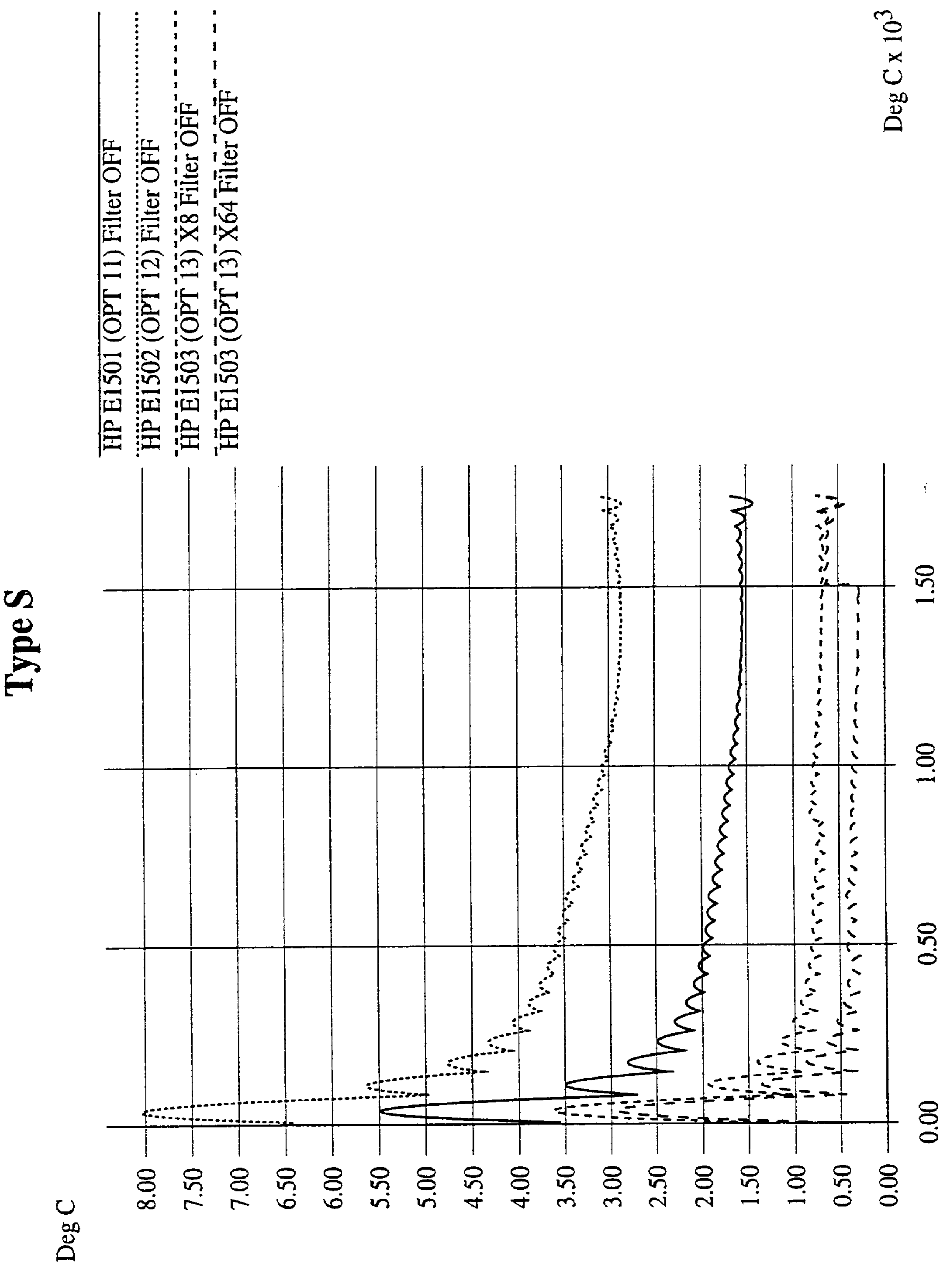



Type R
HP E1501 (OPT 11) Filter OFF
HP E1502 (OPT 12) Filter OFF
HP E1503 (OPT 13) X8 Filter OFF
HP E1503 (OPT 13) X64 Filter OFF
Deg C
7.00
6.50
6.00
5.50
5.00
4.50
4.00
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
0.00
0.20
0.40
0.60
0.80
1.00
Deg C x 10^3

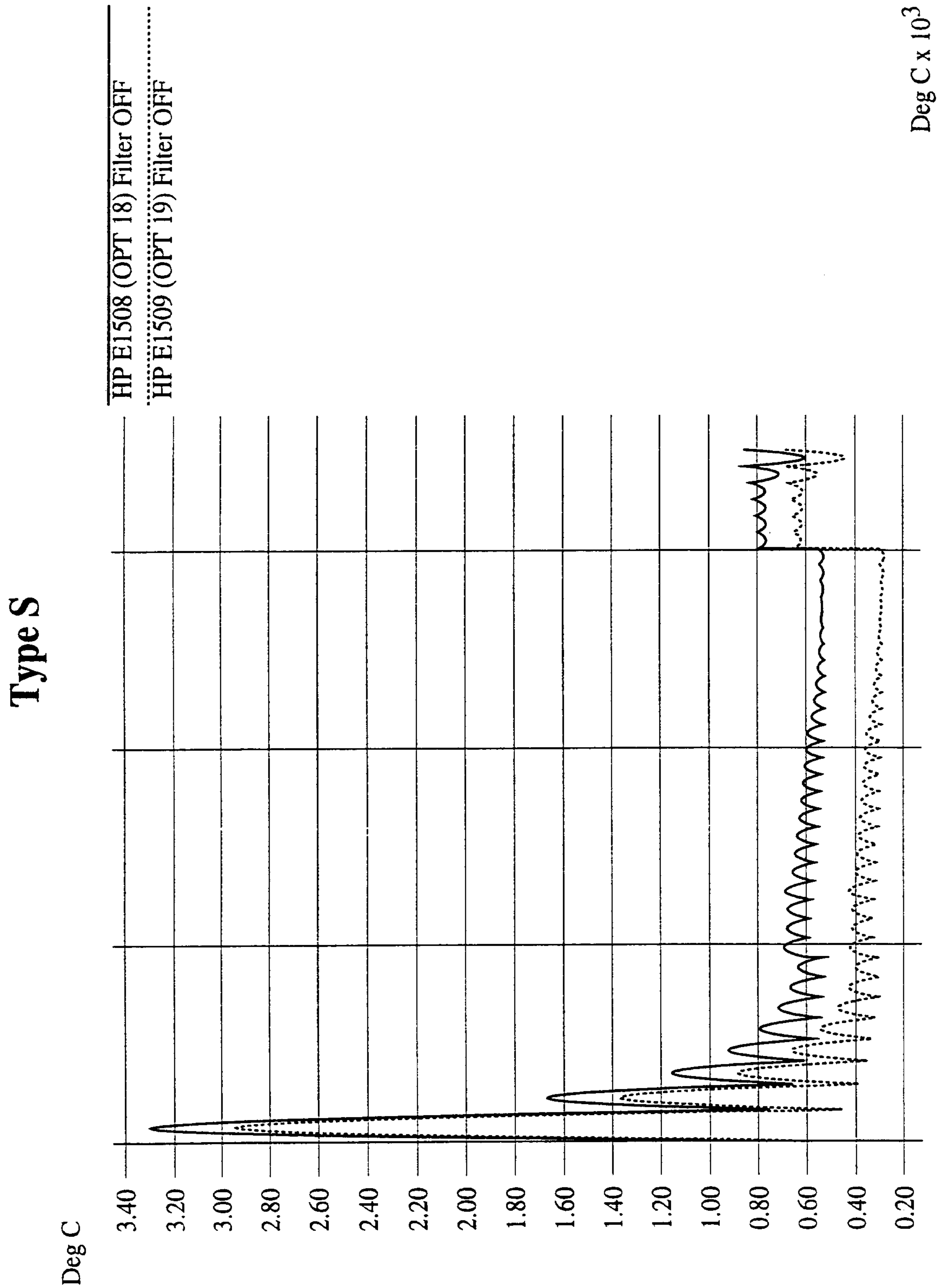
Thermocouple Type R, SCPs HP E1508,09



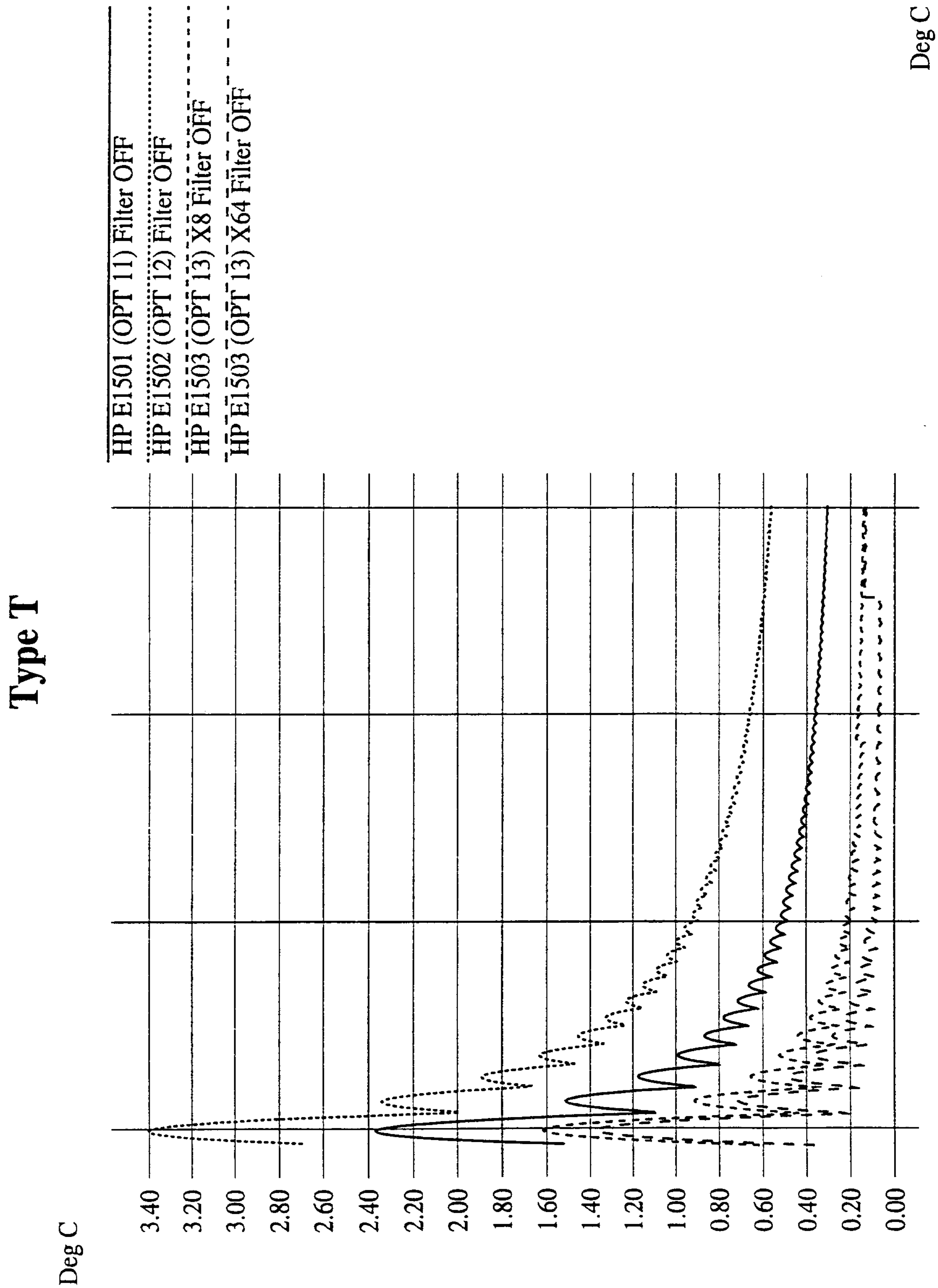
Thermocouple Type S, SCPs HP E1501,02,03



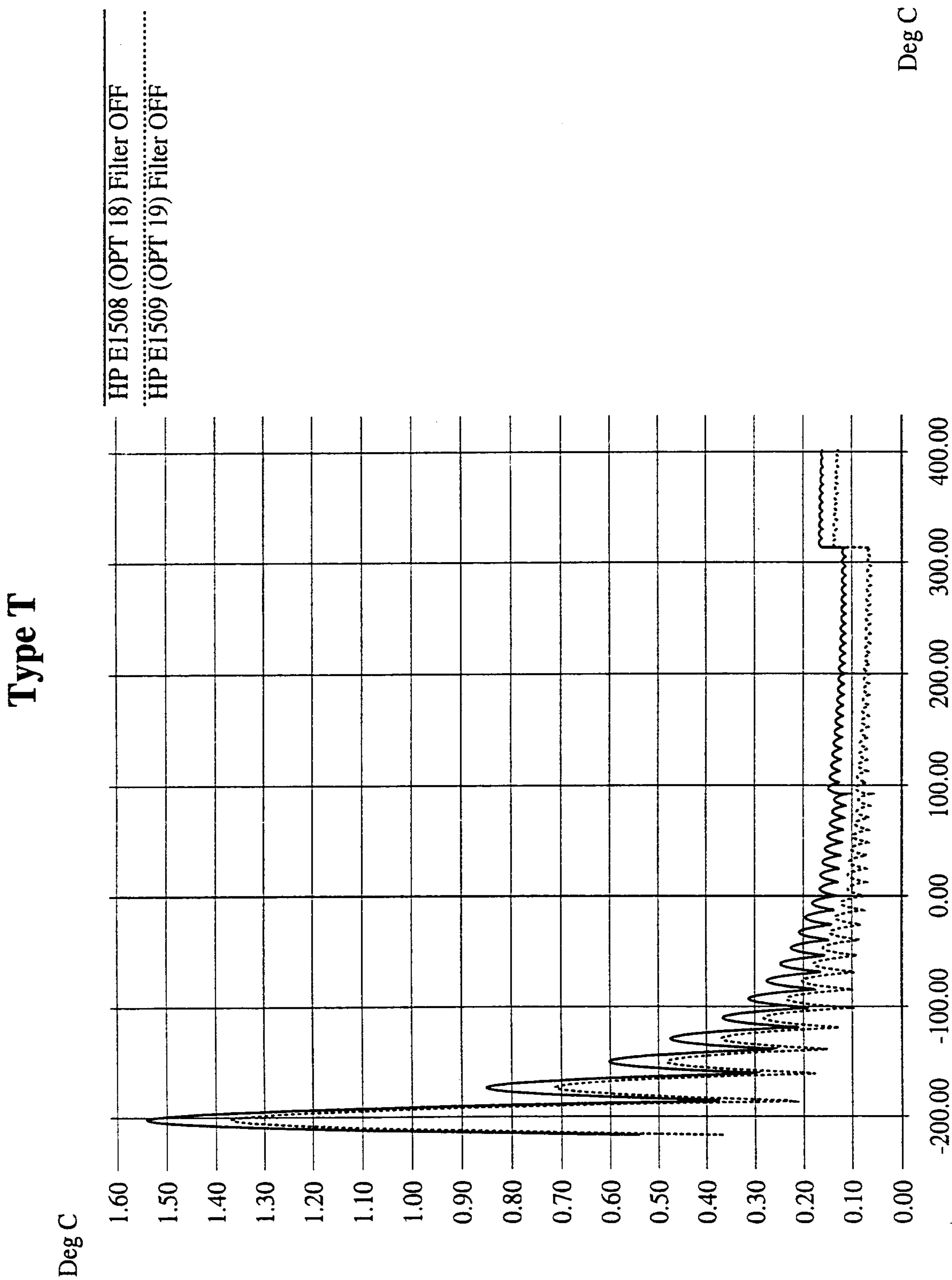
Thermocouple Type S, SCPs HP E1508,09



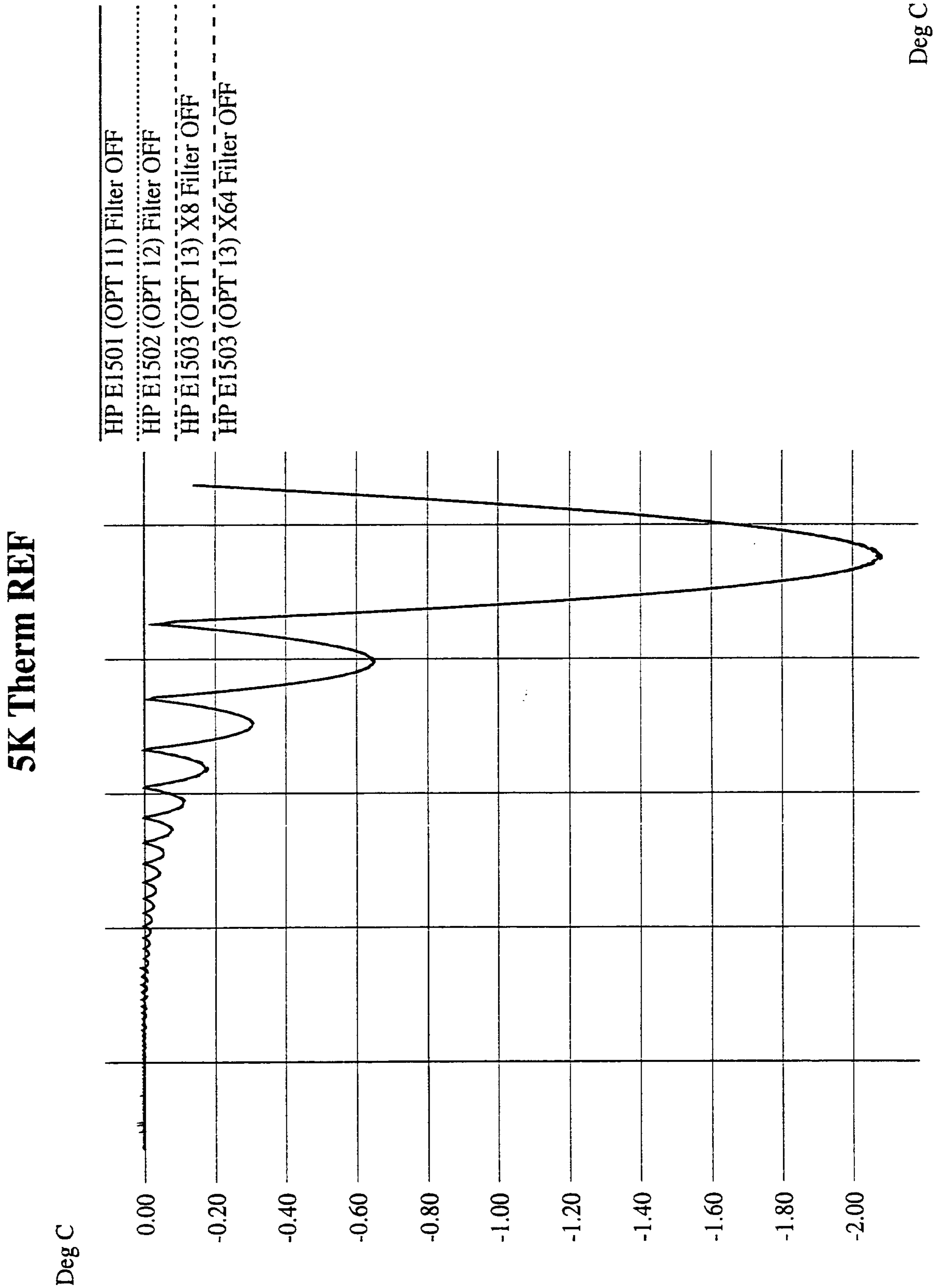
Thermocouple Type T, SCPs HP E1501,02,03



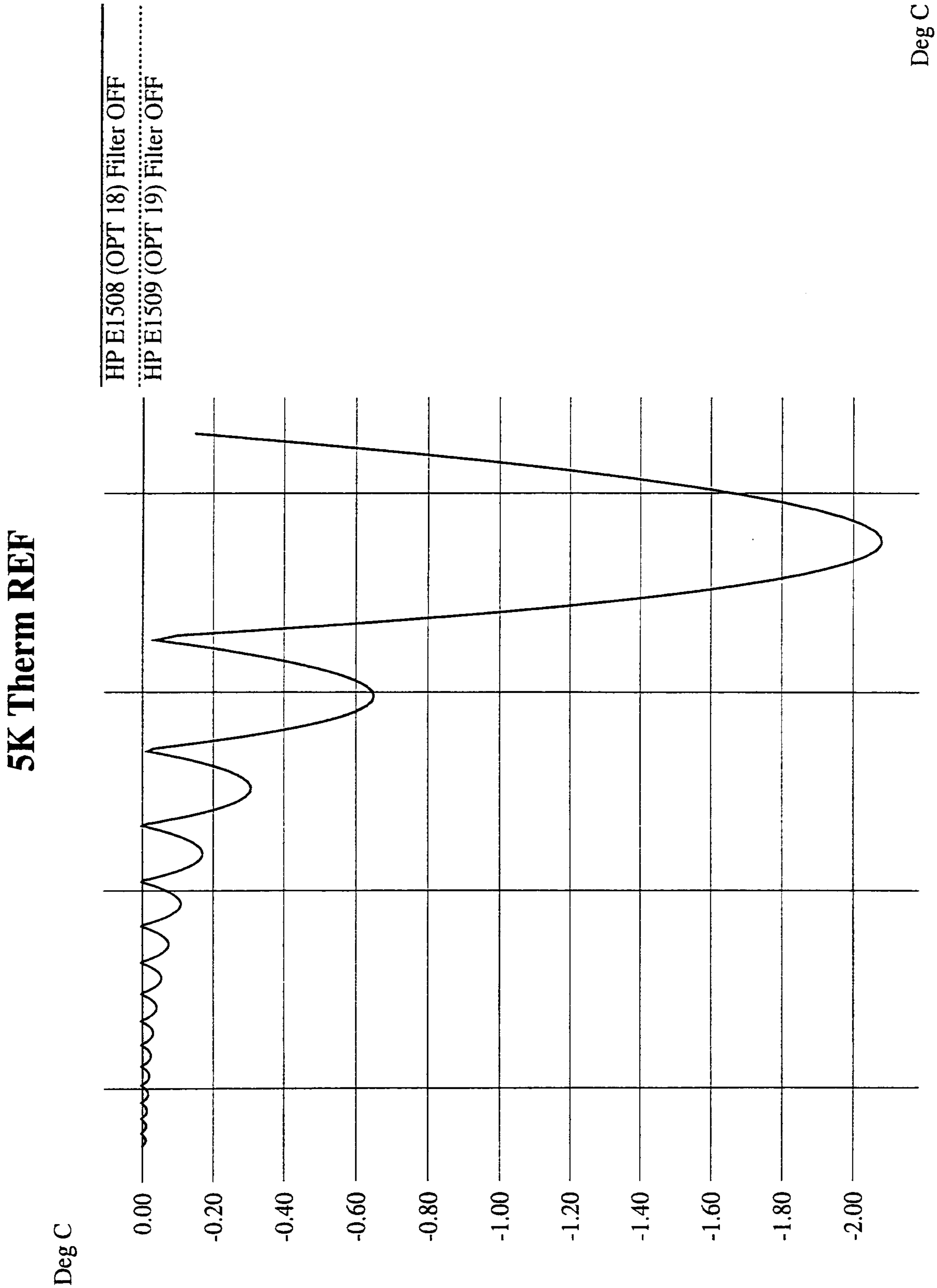
Thermocouple Type T, SCPs HP E1508,09



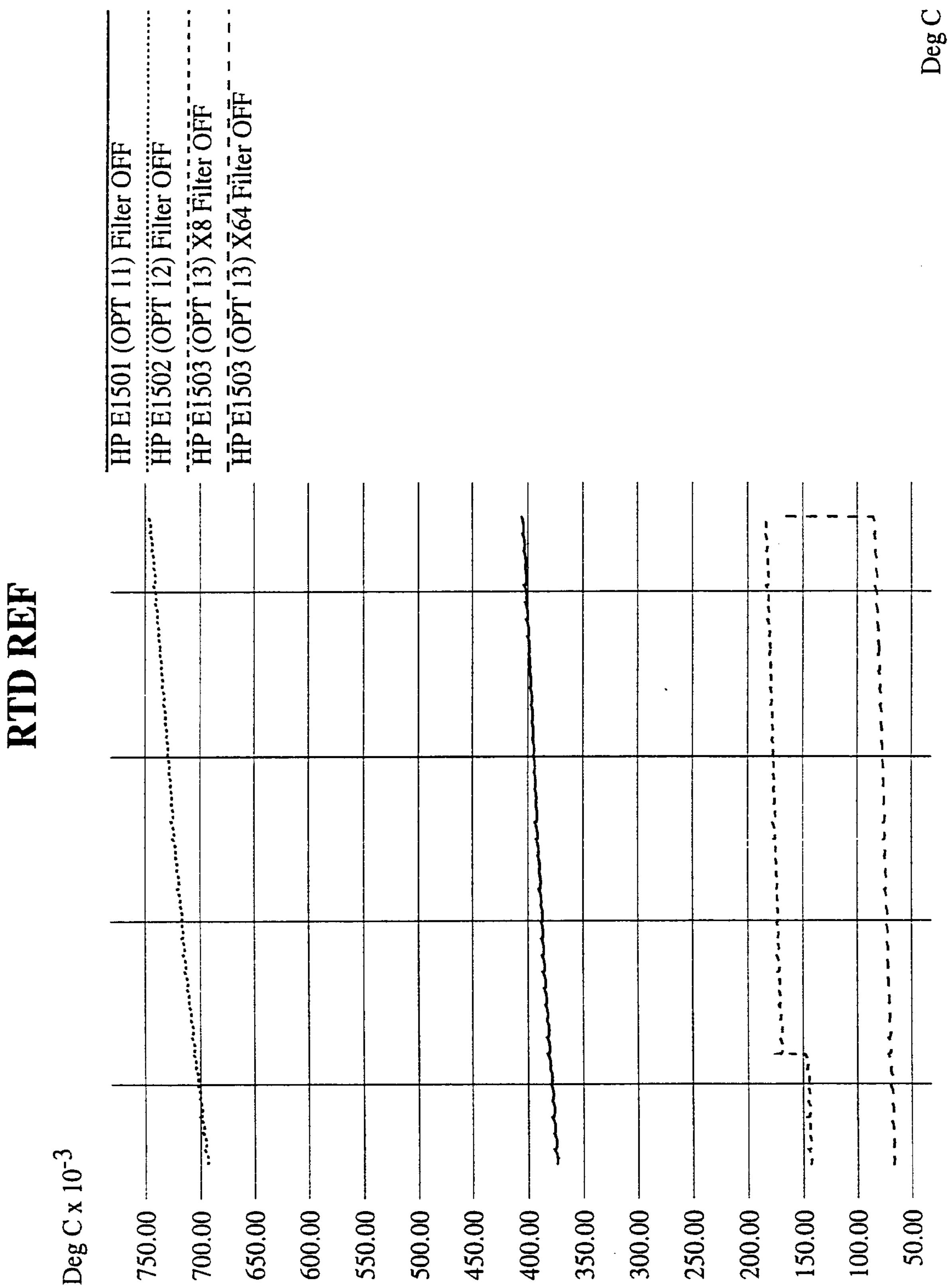
5K Thermistor Reference, SCPs HP E1501,02,03

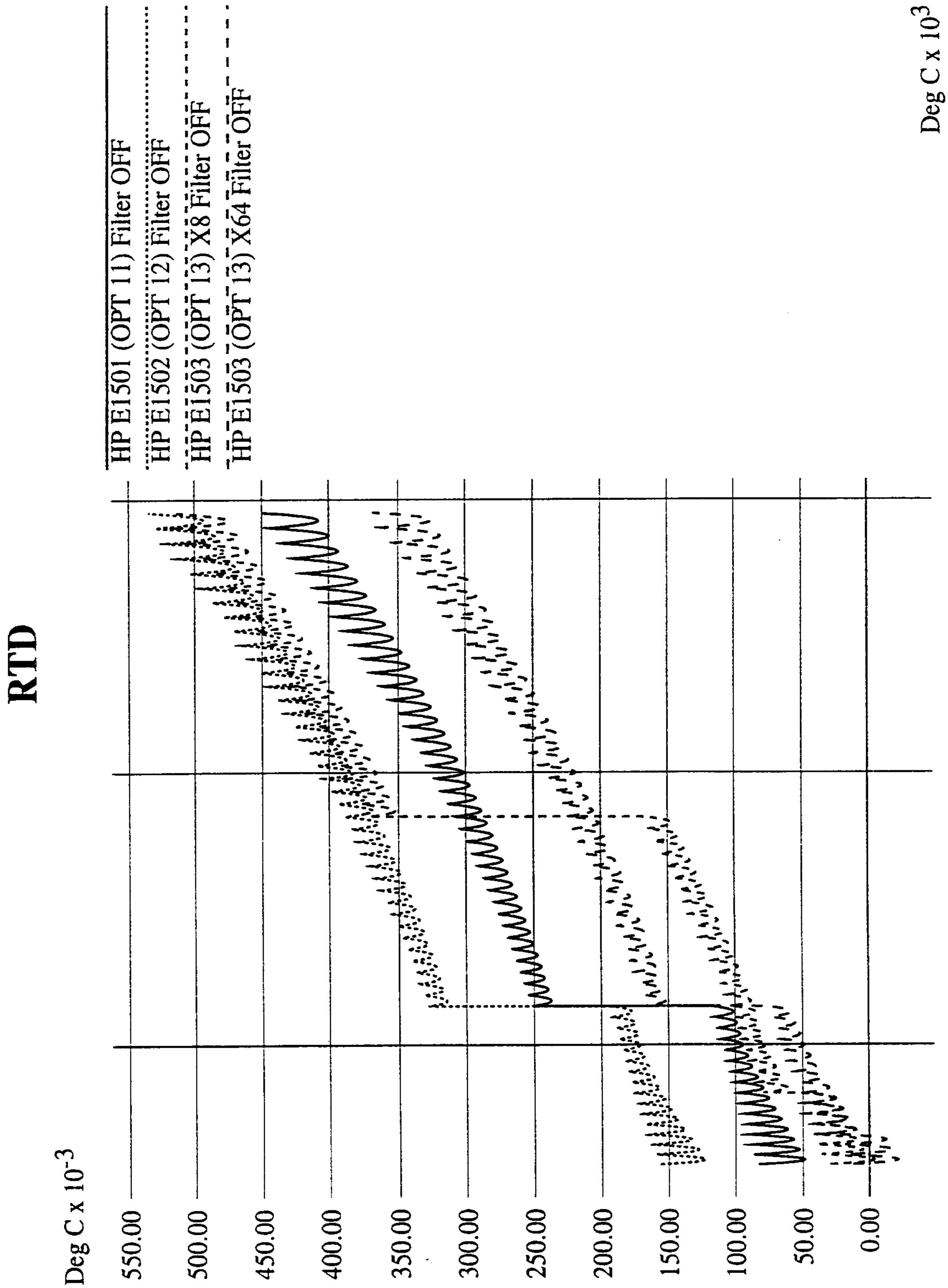


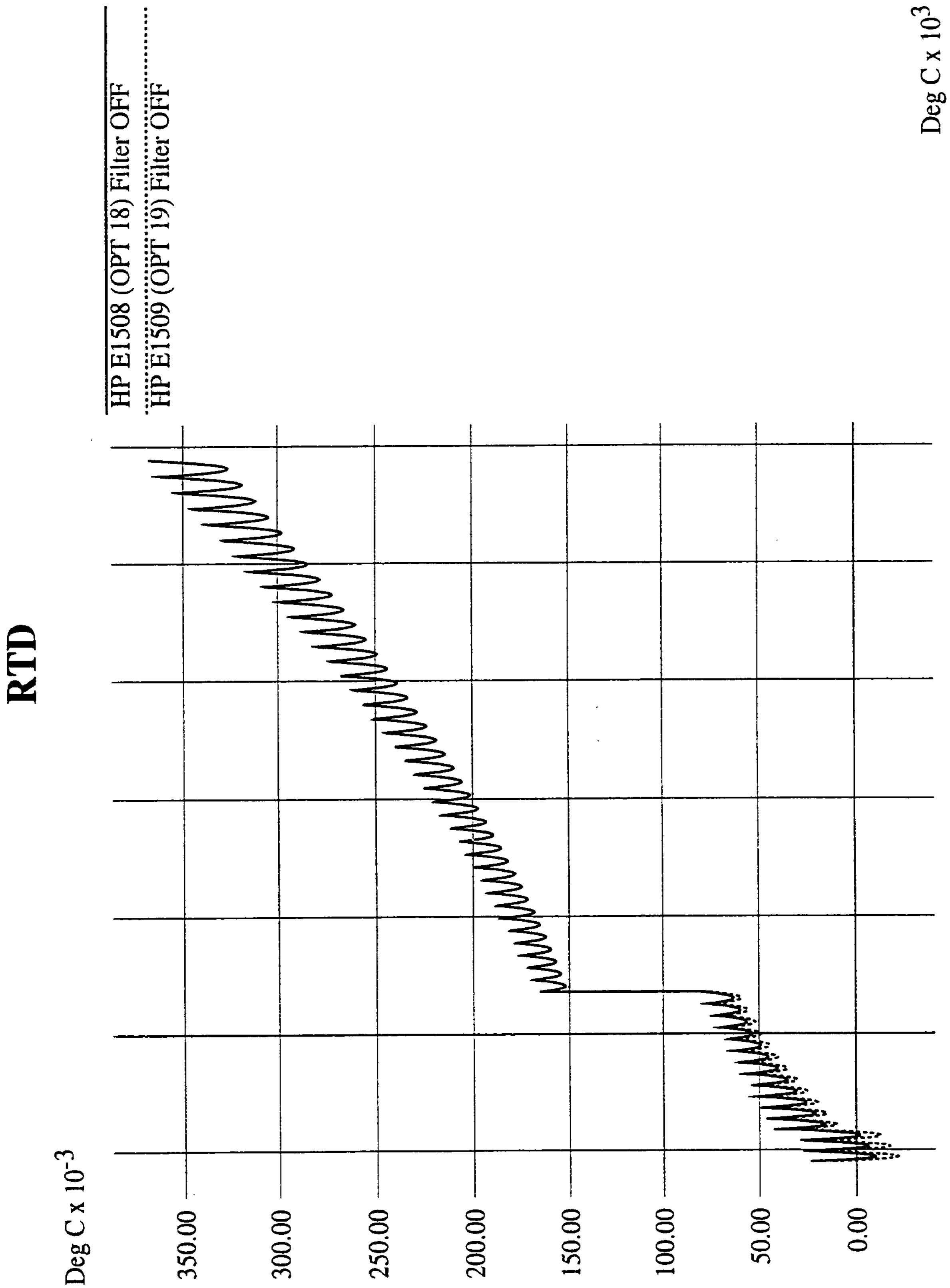
5K Thermistor Reference, SCPs HP E1508,09



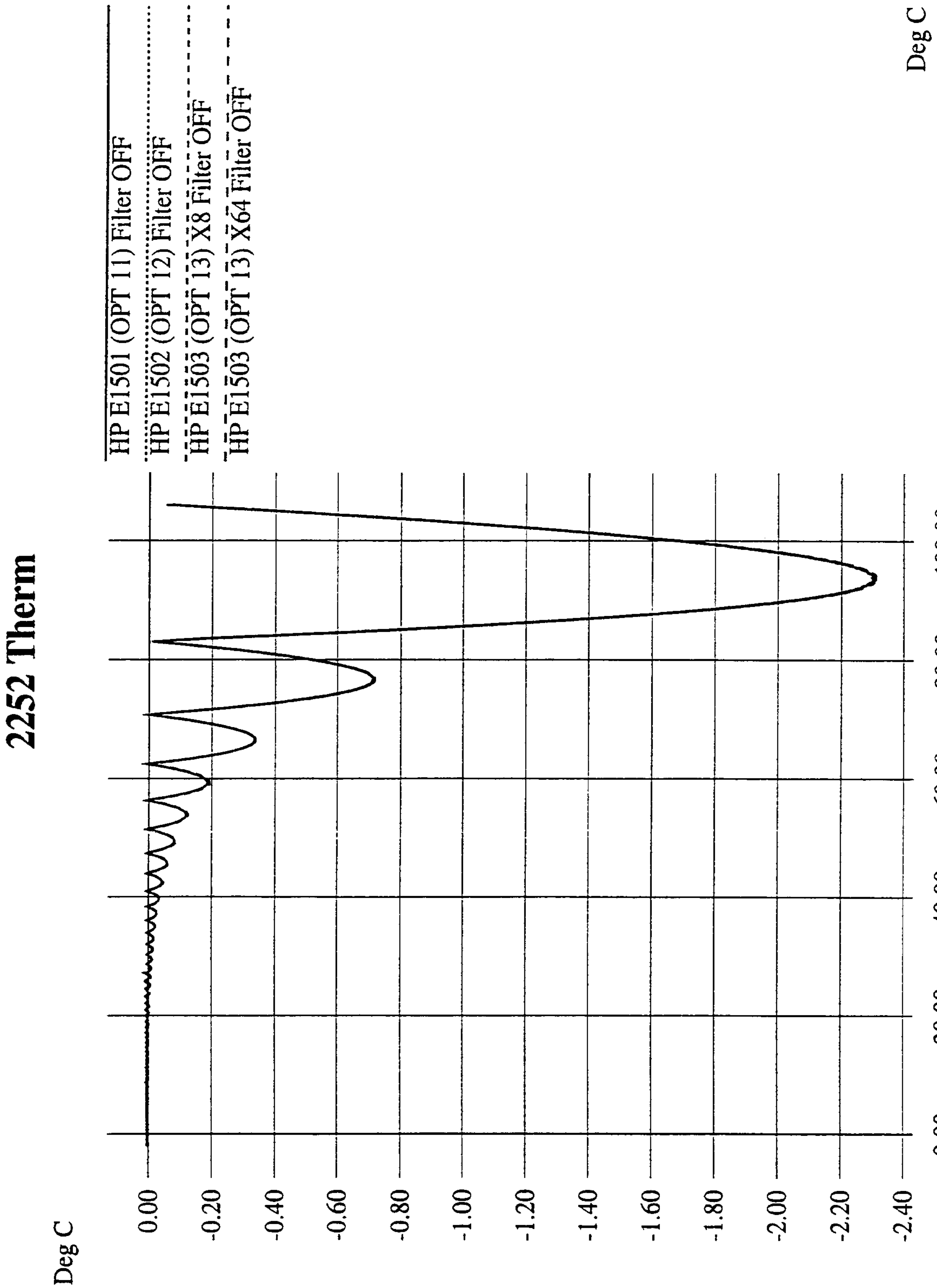
RTD Reference, SCPs HP E1501,02,03



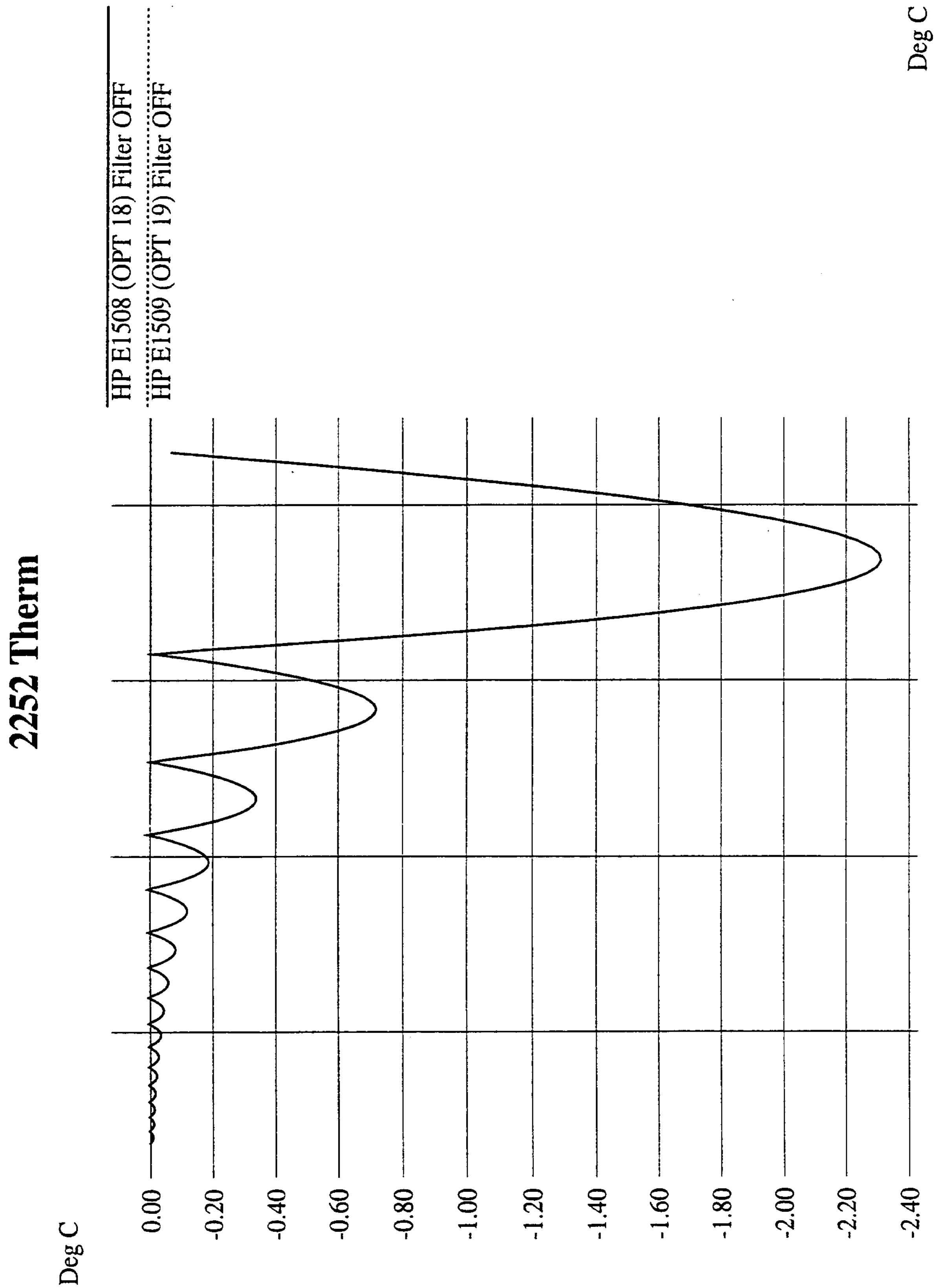

RTD
Deg C x 10-3
550.00
500.00
450.00
400.00
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
HP E1501 (OPT 11) Filter OFF
HP E1502 (OPT 12) Filter OFF
HP E1503 (OPT 13) X8 Filter OFF
HP E1503 (OPT 13) X64 Filter OFF
0.00
0.50
1.00
Deg C x 103

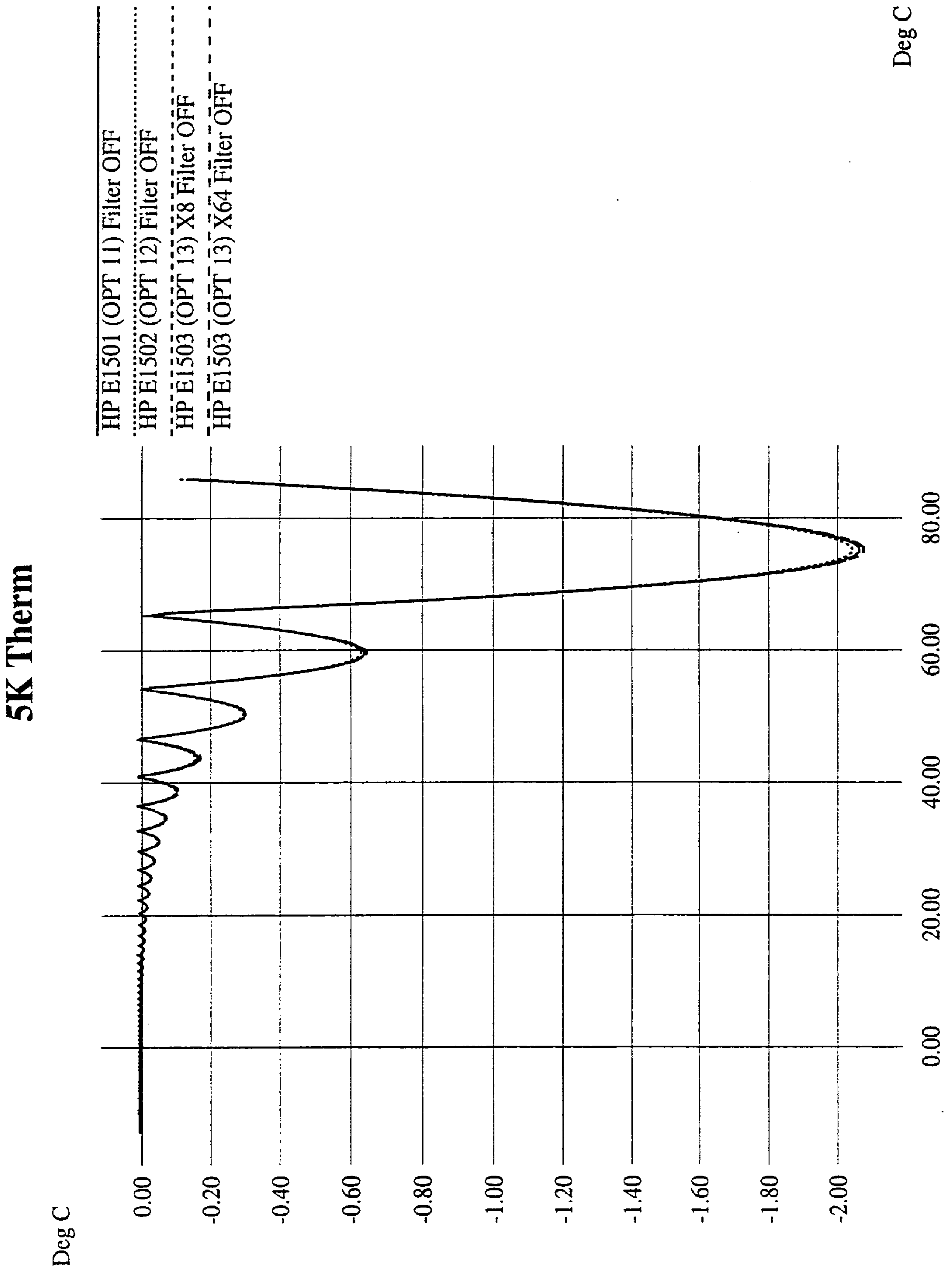

RTD
HP E1508 (OPT 18) Filter OFF
HP E1509 (OPT 19) Filter OFF
Deg C x 10^{-3}
350.00
300.00
250.00
200.00
150.00
100.00
50.00
0.00
-0.20
0.00
0.20
0.40
0.60
0.80
1.00
Deg C x 10^{3}

2250 Thermistor, SCPs HP E1501,02,03

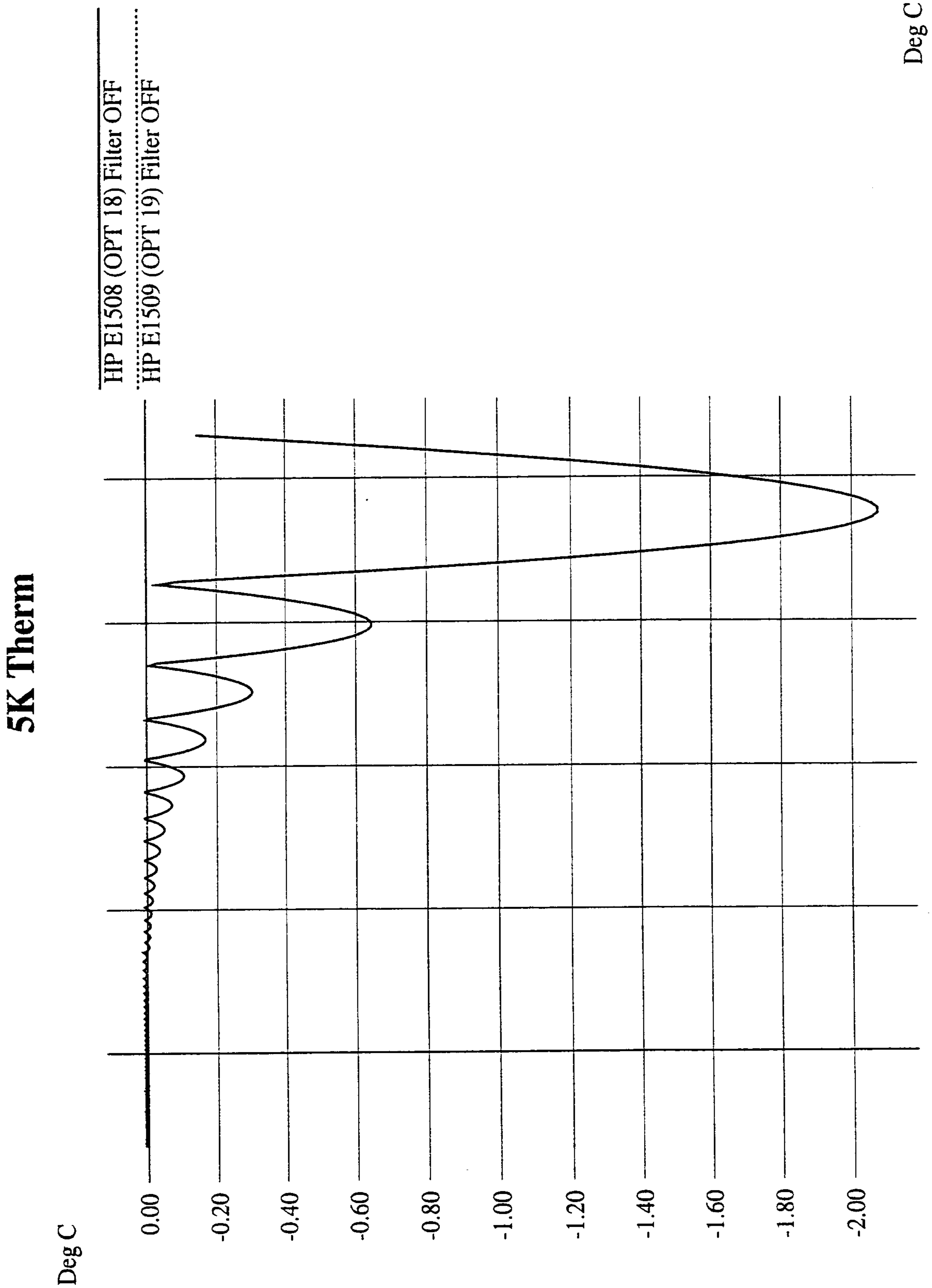


2250 Thermistor, SCPs HP E1508,09

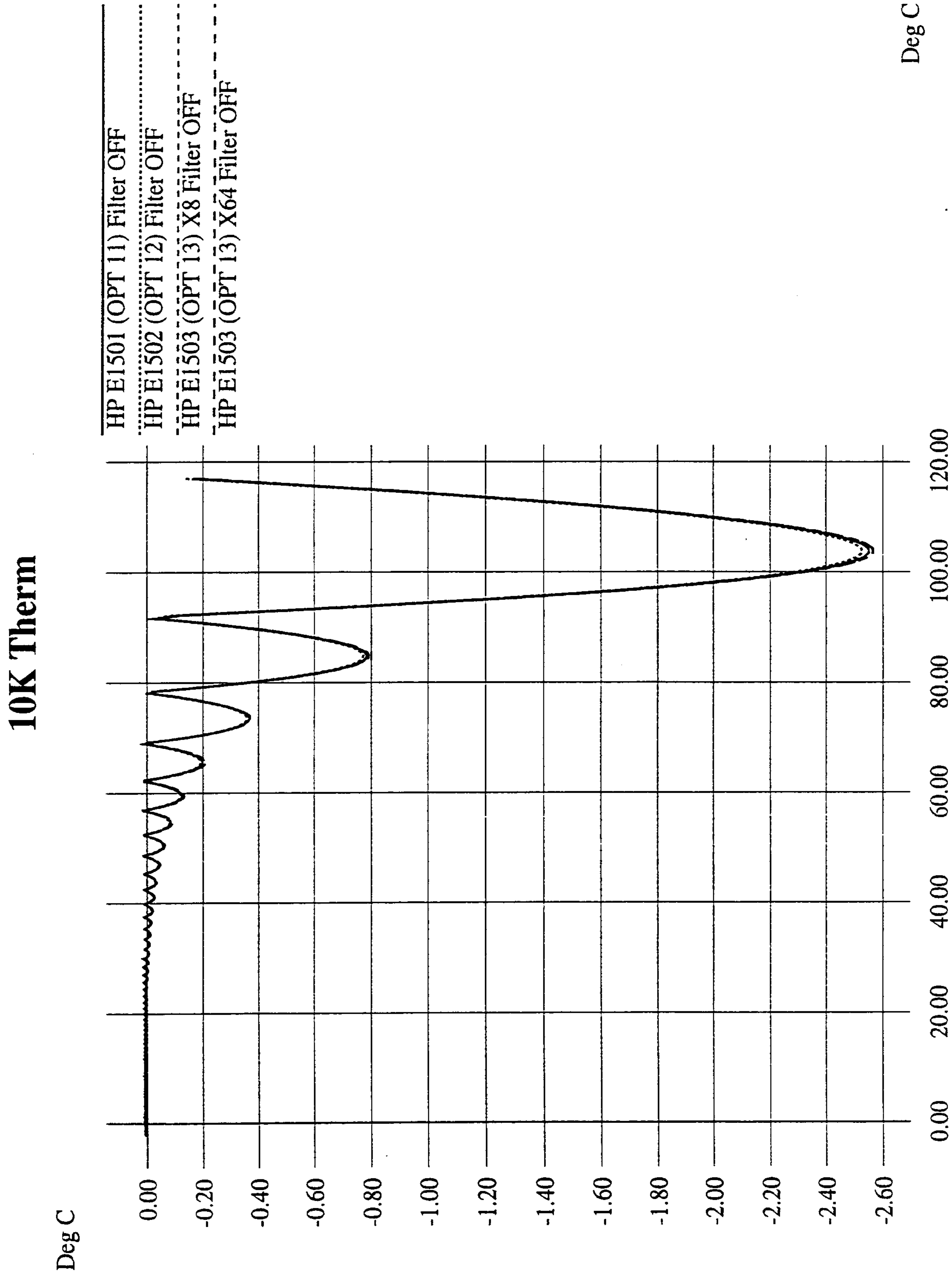



5K Therm
HP E1501 (OPT 11) Filter OFF
HP E1502 (OPT 12) Filter OFF
HP E1503 (OPT 13) X8 Filter OFF
HP E1503 (OPT 13) X64 Filter OFF
Deg C
0.00
-0.20
-0.40
-0.60
-0.80
-1.00
-1.20
-1.40
-1.60
-1.80
-2.00
0.00
20.00
40.00
60.00
80.00
Deg C

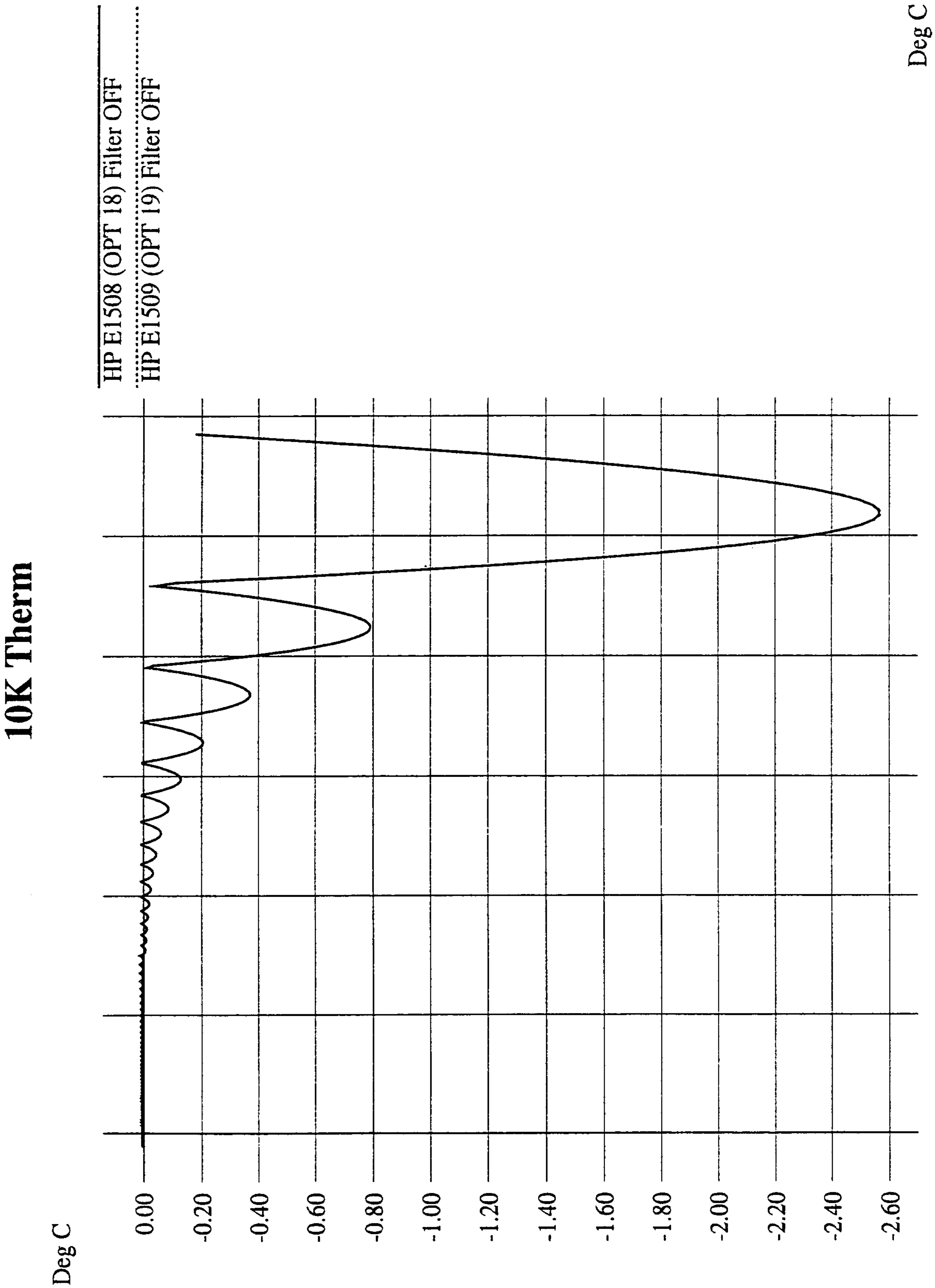
5K Thermistor, SCPs HP E1508,09



10K Thermistor, SCPs HP E1501,02,03



10K Thermistor, SCPs HP E1508,09



HP E1529A Specifications

General:	
HP E1529A outputs:	Single static output from 32:1 multiplexer 32 individually buffered dynamic outputs
Bridge completion:	120, 350 Ohm and user installed, program selectable
Bridge configurations:	Full, half, and quarter
Remote operation:	330m (1000 ft) from multiiplexed output 100m (300 ft) from buffered outputs
Bridge excitation:	User-supplied excitation in 8-channel banks
Linearization:	Mx+b on all channels
Calibration:	Internal self-calibration source 50 kOhm and user-installed shunt calibration resistor, program selectable
Measurement rate:	25 kSa/s via multiplexed output, up to 196 kSa/s dynamic
Static (multiplexed) outputs:	
Gain (HP E1529A only)	32V per V
Gain (HP E1529A + E1422A)	5000V per V
Resolution (1 LSB @ HP E1422A)	0.06 microvolt (subject to RMS noise limits)
Recommended measurement products:	Note: Companion products listed below are VXI-based. Twelve measurement module slots are typically available in systems using any one of HP's available 13-slot VXI mainframes.
Static strain measurements	HP E1422A Remote Channel DAC Module plus up to 8 HP E1539A SCPs
Dynamic strain measurements	HP E1432A 16 Chan 51.2 kSa/s Digitizer HP E1433A 8 Chan 196 kSa/s Digitizer

Bridge Specifications	
Completion resistors:	
Values	120/350 Ohm ± 0.05%, ±5 ppm/°C TC
Power	0.125W @ 125 °C
Shunt cal resistor:	50 kOhm ±0.1%, ±25 ppm/°C TC
Quarter bridge offset:	3 microstrain (±2 microvollt), ±4 °C of tare cal
Excitation sense:	
Maximum input	±8 VDC (16V centered about Gnd terminal)
Gain accuracy	±0.01% of reading
Offset	<1 mV

Strain Measurement		
Measurement range (μ∈) **(Quarter bridge, ±5V excitation:**	**Resolution (μ∈):**	**RMS noise (μ∈):**
±200,000	6.1	0.4
±50,000	1.5	0.4
±12,500	0.4	0.4
±3,125	0.1	0.4 (noise can be reduced by averaging)
System accuracy:	Note: After CAL routine, 1 hour warm-up, ±1 °C	
Voltage offset:	<2 microvolt	

Gain error:	<0.015% of reading
RMS Noise:	<1 microvolt rms
CMRR:	>100 dB, DC-10 MHz (common mode range ± 10V)
Drift:	Note: drift errors can be removed by running CAL routine
Offset drift	<1 microvolt/°C <1 microvolt/month
Gain drift	<30 ppm/°C
Dynamic outputs:	32V per V ±0.1% of reading
Gain:	<250 microvolt
Offset:	>20 kHz
Bandwidth:	
Equivalent input noise (E.I.N.):	<20 nv/sqr(Hz)

Quarter Bridge Bending Errors: (5 V Excitation, GF=2)

μ∈	**Error (μ∈)**
-50,000	160
-40,000	90
-30,000	45
-20,000	20
-10,000	8
0	1
10,000	8
20,000	20
30,000	45
40,000	90
50,000	160

Half Bridge Bending Errors: (5 V Excitation, GF=2)

μ∈	**Error (μ∈)**
-50,000	28
-40,000	23
-30,000	17
-20,000	12
-10,000	6
0	0.5
10,000	6
20,000	12
30,000	17
40,000	23
50,000	28

Full Bridge Bending Errors : (5 V Excitation, GF=2)

μ∈	**Error (μ∈)**
-50,000	28
-40,000	22
-30,000	17
-20,000	11
-10,000	6
0	0.3
10,000	6
20,000	11
30,000	17
40,000	22
50,000	28

Mechanical	
Height: 4.45 cm (1.75 in)	
Width: 49.53 cm (19.5 in)	
Weight: 1.8 Kg (4 lbs)	

Power Requirements	
Line voltage: 100-240 Vac ±10%	CAT II (2500Vpk transients)
Line frequency: 50-60 Hz	
Input power: 16 VA	

Environmental	
Polution Degree:	2
Temperature:	-5 °C to +55 °C operating
Humidity:	5 to 85% R.H.
Altitude:	3,000 meters (10,000 ft) operating 10,000 meters (30,000 ft) non-operating

Warranty	
HP E1529A:	3 year return to HP

Appendix B
Error Messages

Possible Error Messages:

-108	'Parameter not allowed'.
-109	'Missing parameter'
-160	'Block data error'.
-211	'Trigger ignored'.
-212	'Arm ignored'.
-213	'Init ignored'.
-221	'Settings conflict'.
-222	'Data out of range'.
-224	'Illegal parameter value'.
-240	'Hardware error'. Execute *TST?.
-253	'Corrupt media'.
-281	'Cannot create program'.
-282	'Illegal program name'.
-310	'System error'.
-410	'Query INTERRUPTED'.
1000	'Out of memory'
2001	'Invalid channel number'.
2003	'Invalid word address'.
2007	'Bus error'.
2008	'Scan list not initialized'.
2009	'Too many channels in channel list'.
2016	'Byte count is not a multiple of two'.
3000	'Illegal while initiated'. Operation must be performed

before INIT or INIT:CONT ON.

3004	'Illegal command. CAL:CONF not sent'. Incorrect sequence of calibration commands. Send CAL:CONF:VOLT command before CAL:VAL:VOLT and send CAL:CONF:RES command before CAL:VAL:RES
3005	'Illegal command. Send CAL:VAL:RES'. The only command accepted after a CAL:CONF:RES is a CAL:VAL:RES command.
3006	'Illegal command. Send CAL:VAL:VOLT'. The only command accepted after a CAL:CONF:VOLT is a CAL:VAL:VOLT command.
3007	'Invalid signal conditioning module'. The command sent to an SCP was illegal for its type.
3008	'Too few channels in scan list'. A Scan List must contain at least two channels.
3012	'Trigger too fast'. Scan list not completed before another trigger event occurs.
3015	'Channel modifier not permitted here'.
3019	'TRIG:TIM interval too small for SAMP:TIM interval and scan list size'. TRIG:TIM interval must allow for completion of entire scan list at currently set SAMP:TIM interval. See TRIG:TIM in Chapter 5, the Command Reference
3020	'Input overvoltage'. Calibration relays opened (if JM2202 not cut) to protect module inputs, and Questionable Data Status bit 11 set. Execute *RST to close relays and/or reset status bit.
3021	'FIFO overflow'. Lets you know that the FIFO buffer has filled and that one or more readings have been lost. Usually caused by algorithm values stored in FIFO faster than FIFO was read.
3026	'*CAL?/CAL:SET Calibration failed'.
3027	'Unable to map A24 VXI memory'.
3028	'Incorrect range value'. Range value sent is not supported by instrument.
3030	'Command not yet implemented!!'.

3032	'0x1: DSP-Unrecognized command code'.
3033	'0x2: DSP-Parameter out of range'.
3034	'0x4: DSP-Flash rom erase failure'.
3035	'0x8: DSP-Programming voltage not present'.
3036	'0x10: DSP-Invalid SCP gain value'. Check that SCP is seated or replace SCP. Channel numbers are in FIFO.
3037	'0x20: DSP-Invalid *CAL? constant or checksum. *CAL? required.'.
3038	'0x40: DSP-Couldn't cal some channels'. Check that SCP is seated or replace SCP. Channel numbers are in FIFO.
3039	'0x80: DSP-Re-Zero of ADC failed'.
3040	'0x100: DSP-Invalid Tare CAL constant or checksum'. Perform CAL:TARE - CAL:TARE? procedure.
3041	'0x200: DSP-Invalid Factory CAL constant or checksum'. Perform A/D Cal procedure.
3042	'0x400: DSP-DAC adjustment went to limit'. Execute *TST?
3043	'0x800: DSP Status--Do *CAL?'.
3044	'0x1000: DSP-Overvoltage on input'.
3045	'0x2000: DSP-cal constant out of range'. Execute *CAL?
3046	'0x4000: DSP-ADC hardware failure'.
3047	'0x8000: DSP-reserved error condition'.
3048	'Calibration or Test in Process'.
3049	'Calibration not in Process'.
3050	'ZERO must be sent before FSCale'. Perform A/D Cal sequence as shown in Command Reference under CAL:CONF:VOLT
3051	'Memory size must be multiple of 4'. From MEM:VME:SIZE. Each HP E1422 reading requires 4 bytes.

3052 'Self test failed. Test info in FIFO'. Use SENS:DATA:FIFO:ALL? to retrieve data from FIFO.

NOTE: *TST? always sets the FIFO data FORMat to ASCII,7. Read FIFO data into string variables.

FIIFO Value	Definition
1 - 99	ID number of failed test (see following table for possible corrective actions)
100 - 163	channel number(s) associated with test (ch 0-63)
164	special "channel" used for A/D tests only
200	A/D range 0.0625V associated with failed test
201	A/D range 0.25V associated with failed test
202	A/D range 1V associated with failed test
203	A/D range 4V associated with failed test
204	A/D range 16V associated with failed test

Test ID	Corrective Actions
1 - 19, 21 - 29	(HP Service)*
20, 30 -37	Remove all SCPs and see if *TST? passes. If so, replace SCPs one at a time until you find the one causing the problem.
38 - 71	(HP Service)*
72,74 - 76, 80 - 93, 301 - 354	re-seat the SCP that the channel number(s) points to, or move the SCP and see if the failure(s) follow the SCP. If the problems move with the SCP, replace the SCP.
73, 77 - 79, 94 - 99	(HP Service)*

*Must send module to an HP Service Center for repair. Record information found in FIFO to assist the HP Service Center in repairing the problem.

Refer to the Command Reference under *TST? for a list of module functions tested.

NOTE During the first 5 minutes after power is applied, *TST? may fail. Allow the module to warm-up before executing *TST?

3053	'Corrupt on board Flash memory'.
3056	'Custom EU not loaded'. May have erased custom EU conversion table with *RST. May have linked channel with standard EU after loading custom EU, this erases the custom EU for this channel. Reload custom EU table using DIAG:CUST:LIN or DIAG:CUST:PIEC.
3057	'Invalid ARM or TRIG source when S/H SCP's enabled' Don't set TRIG:SOUR or ARM:SOUR to SCP with HP E1510 or HP E1511 installed.
3058	'Hardware does not have D32, S/H, or new trigger capabilities'. Module's serial number is earlier than 3313A00530.
3067	'Multiple attempts to erase Flash Memory failed'
3068	'Multiple attempts to program Flash Memory failed'
3069	'Programming voltage jumper not set properly'. See Disabling Flash Memory Access in Chapter 1 (JM2201)
3070	'Identification of Flash ROM incorrect'
3071	'Checksum error on Flash Memory'
3074	'WARNING! Old Opt 16 or Opt 17 card can damage SCP modules' must use HP E1506 or HP E1507.
3075	'Too many entries in CVT list'
3076	'Invalid entry in CVT list' Can only be 10 to 511
3077	'Too many updates in queue. Must send UPDATE command' To allow more updates per ALG:UPD, increase ALG:UPD:WINDOW
3078	'Invalid Algorithm name' Can only be 'ALG1' through 'ALG32', or 'GLOBALS', or 'MAIN'
3079	'Algorithm is undefined' In ALG:SCAL, ALG:SCAL?, ALG:ARR, or ALG:ARR?
3080	'Algorithm already defined' Trying to repeat ALG:DEF with same <*alg_name*> (and is not enabled to swap), or trying to define 'GLOBALS' again since last *RST
3081	'Variable is undefined' Algorithm exists but has no local variable by that name.

3082	'Invalid Variable name' Must be valid 'C' identifier, see Chapter 5
3083	'Global symbol (variable or custom function) already defined' Trying to define a global variable with same name as a user defined function, or vice versa. User functions are also global.
3084	'Algorithmic error queue full' ALG:DEF has generated too many errors from your algorithm source code
3084	"Error 1:Number too big for a 32 bit float" "Error 2:Number too big for a 32 bit integer" "Error 3:'8' or '9' not allowed in an octal number" "Error 4:Syntax error" "Error 5:Expecting '('" "Error 6:Expecting ')'" "Error 7:Expecting an expression" "Error 8:Out of driver memory" "Error 9:Expecting a bit number (Bn or Bnn)" "Error 10:Expecting ']'" "Error 11:Expecting an identifier" "Error 12:Arrays can't be initialized" "Error 13:Expecting 'static'" "Error 14:Expecting 'float'" "Error 15:Expecting ';'" "Error 16:Expecting ','" "Error 17:Expecting '='" "Error 18:Expecting '{'" "Error 19:Expecting '}'" "Error 20:Expecting a statement" "Error 21:Expecting 'if'" "Error 22:Can't write to input channels" "Error 23:Expecting a constant expression" "Error 24:Expecting an integer constant expression" "Error 25:Reference to an undefined variable" "Error 26:Array name used in a scalar context" "Error 27:Scalar name used in an array context" "Error 28:Variable name used in a custom function context" "Error 29:Reference to an undefined custom function" "Error 30:Can't have executable code in GLOBALS definition" "Error 31:CVT address range is 10 - 511" "Error 32:Numbered algorithms can only be called from MAIN" "Error 33:Reference to an undefined algorithm" "Error 34:Attempt to redefine an existing symbol (var or fn)" "Error 35:Array size is 1 - 1024"

"Error 36:Expecting a default PID parameter"
"Error 37:Too many FIFO or CVT writes per scan trigger"
"Error 38:Statement is too complex"
"Error 39:Unterminated comment"

3085 ’Algorithm too big’ Algorithm exceeded 46K words (23K if enabled to swap), or exceeded size specified in <*swap_size*>.

3086 ’Not enough memory to compile Algorithm’ Your algorithm’s constructs are using too much translator memory. Need more memory in your HP E1406. Try breaking your algorithm into smaller algorithms.

3088 ’Too many functions’ Limit is 32 user defined functions

3089 ’Bad Algorithm array index’ Must be from 0 to (declared size)-1

3090 ’Algorithm Compiler Internal Error’ Call HP with details of operation.

3091 ’Illegal while not initiated’ Send INIT before this command

3092 ’No updates in queue’

3093 ’Illegal Variable Type’ Sent ALG:SCAL with identifier of array, ALG:ARR with scalar identifier, ALG:UPD:CHAN with identifier that is not a channel, etc.

3094 ’Invalid Array Size’ Must be 1 to 1024

3095 ’Invalid Algorithm Number’ Must be ’ALG1’ to ’ALG32’

3096 ’Algorithm Block must contain termination ’ Must append a null byte to end of algorithm string within the Block Data

3097 ’Unknown SCP. Not Tested’ May receive if you are using a breadboard SCP

3099 ’Invalid SCP for this product’

3100 ’Analog Scan time to big. Too much settling time’ Count of channels referenced by algorithms combined with use of SENS:CHAN:SETTLING has attempted to build an analog Scan List greater than 64 channels.

3101 'Can't define new algorithm while running' Execute ABORT, then define algorithm

3102 'Need ALG:UPD before redefining this algorithm again' Already have an algorithm swap pending for this algorithm.

3103 'Algorithm swapping already enabled; Can't change size' Only send <*swap_size*> parameter on initial definition.

3104 'GLOBALS can't be enabled for swapping' Don't send <*swap_size*> parameter for ALG:DEF 'GLOBALS'

3105 'Invalid SCP switch setting'

3106 'E1536 debounce - Ch list must contain all lower 4 and/or upper 4 channels'

3107 'Channel data direction conflicts with command' Check switches that hardware-define data direction (configure channel as input or output).

3108 'E1536 debounce - each referenced 4 Ch debounce bank must contain at least one input channel'

3110 'Channel specified is invalid for RVELocity function.' See the HP E1538A SCP manual.

3111 'Multiple channels are specified in the reference channel list' See the HP E1538A SCP manual.

3112 'Channel specified is invalid for RPULse reference channel' See the HP E1538A SCP manual.

3113 'Channel specified is not on the same SCP as reference channel' See the HP E1538A SCP manual.

3114 'First channel on SCP can not be used in RPULse output channel list' See the HP E1538A SCP manual.

3115 'Channels specified are not in ascending order' See the HP E1538A SCP manual.

3116 'Multiple channels specified are not grouped correctly' See the HP E1538A SCP manual.

3117 'Grouped channels are not adjacent' See the HP E1538A SCP manual.

3118 'Incomplete setup information for RPULse function' See the HP E1538A SCP manual.

3119	RPULse reference channel must be defined as RVELocity type' See the HP E1538A SCP manual.
3120	'Minimum velocity parameter must not exceed maximum velocity parameter' See the HP E1538A SCP manual.
3121	'Query invalid for current channel usage with the configured RPULse mode' See the HP E1538A SCP manual.
3122	'This multiple channel function must not span multiple SCPs.' See the HP E1538A SCP manual.
3123	'E1538 OE switch ON conflicts with this command' See the HP E1538A SCP manual.
3124	'E1538 OE switch OFF conflicts with this command' See the HP E1538A SCP manual.
3125	'E1538 VRS switch setting conflicts with OE switch setting' See the HP E1538A SCP manual.
3126	'E1538 VRS switch setting conflicts with PU switch setting' See the HP E1538A SCP manual.
3127	'Undefined E1538 stepper motor mode' See the HP E1538A SCP manual.
3128	'E1538 Input threshold calibration failure' See the HP E1538A SCP manual.
3129	'Incompatible Aperture and Range values'
3131	'First or last channel in specified range is invalid'
3132	'Channel modifier has illegal value' Only legal values for Data Destination modifier are 1=CVT, 2=FIFO, 3=CVT and FIFO, 0=niether CVT or FIFO
3133	'Last channel in range must be greater than first channel' 10000:10008 is OK, 10008:10000 is not OK.
3134	'Scan List contains non-input channel' Scan list defined by ROUT:SEQ:DEF can only contain analog input channel specifiers.
3135	'A 3-digit and 5-digit channel are both going to same CVT location. Channels in ROUT:SEQ:DEF Scan List have fixed CVT destinations. On-board 3-digit channels can collide with remote 5-digit channel CVT destinations. See "ROUTe:SEQuence:DEFine" on

page 287.

3136 'Only 32 channels may be scanned on each E1529' Maximum Scan List entries per HP E1529A is 32, so if you a channel more than once, you must leave out some other channel on that HP E1529A.

3137 'E1539 SCP channel 0 not responding' Indicates a communications error between the HP E1539A SCP and a Remote Signal Conditioning Unit connected to the SCP's channel 0 input. Check the Data Interface connection as well as the power connection to that RSCU.

3138 'E1539 SCP channel 1 not responding' Indicates a communications error between the HP E1539A SCP and a Remote Signal Conditioning Unit connected to the SCP's channel 1 input. Check the Data Interface connection as well as the power connection to that RSCU.

3140 'E1529 data not received properly, cable connected?'

3141 'Gain of 0.0 not allowed for E1529 channel.'

3142 'Custom EU out of date, bridge connection was EXC when created, now is BRID'

3143 'Custom EU out of date, bridge connection was DRID when created, now is EXC'

3144 'Channels 15722 to 15731 illegal to send to CVT.' Only 502 CVT elements for 512 channels so highest 10 channels can't be stored in CVT, use FIFO instead. See "ROUTe:SEQuence:DEFine" on page 287.

3145 'Shunt ON/OFF allows only a single channel per E1529' The shunt cal resistor in an HP E1529A can only be connected to one channel at a time. Your <ch_list> in the OUTPut:SHUNT contained references to more than one channel on the same HP E1529A.

3146 'Communication error reading E1529 calibration constants, defaults used'

3147 'Cecksum error reading E1529 calibration constants, defaults used' Do CAL:REMote? and CAL:REM:STORe on the affected HP E1529A.

Appendix C
Glossary

The following terms have special meaning when related to the HP E1422.

Algorithm In general, an algorithm is a tightly defined procedure that performs a task. This manual, uses the term to indicate a program executed within the HP E1422 that implements a data acquisition and control algorithm.

Algorithm Language The algorithm programming language specific to the HP E1422. This programming language is a subset of the ANSI 'C' language.

Application Program The program that runs in the VXIbus controller, either embedded within the VXIbus mainframe, or external and interfaced to the mainframe. The application program typically sends SCPI commands to configure the HP E1422, define its algorithms, then start the algorithms running. Typically, once the HP E1422 is running algorithms, the application need only "oversee" the control application by monitoring the algorithms' status. During algorithm writing, debugging, and tuning, the application program can retrieve comprehensive data from running algorithms.

Buffer In this manual, a buffer is an area in RAM memory that is allocated to temporarily hold:

Data input values that an algorithm will later access. This is the Input Channel Buffer.

Data output values from an algorithm until these values are sent to hardware output channels. This is the Output Channel Buffer.

Data output values from an algorithm until these values are read by your application program. This is the First-In-First-Out or FIFO buffer.

A second copy of an array variable containing updated values until it is "activated" by an update. This is "double buffering".

A second version of a running algorithm until it is "activated" by an update. This is only for algorithms that are enabled for swapping. This is also "double buffering".

Control Processor The Digital Signal Processor (DSP) chip that performs all of the HP E1422's internal hardware control functions as well as performing the EU Conversion process.

DSP Same as Control Processor

EU Engineering Units

EU Conversion Engineering Unit Conversion: Converting binary A/D readings (in units of A/D counts) into engineering units of voltage, resistance, temperature, strain. These are the "built in" conversions (see SENS:FUNC: ...). The HP E1422 also provides access to custom EU conversions (see SENS:FUNC:CUST in command reference and "Creating and Loading Custom EU Tables" in Chapter 3).

FIFO The First-In-First-OUT buffer that provides output buffering for data sent from an algorithm to an application program.

Flash or Flash Memory Non-volatile semiconductor memory used by the HP E1422 to store its control firmware and calibration constants

RSC unit or RSCU This stands for Remote Signal Conditioning Unit. The HP E1529 Remote Strain Conditioning unit is an example of an RSC.

Scan List A list of up to 512 channels that is built by the ROUTE:SEQUENCE:DEFINE command and analog input channels referenced in algorithms as they are defined. This list will be scanned each time the module is triggered.

SCP Signal Conditioning Plug-on: Small circuit boards that plug onto the HP E1422's main circuit board. Available analog input SCPs can provide noise canceling filters, signal amplifiers, signal attenuators, and strain bridge completion. Analog output SCPs are available to provide measurement excitation current, controlling voltage, and controlling current. Digital SCPs are available to both read and write digital states, read frequency and counts, and output modulated pulse signals (FM and PWM).

Swapping This term applies to algorithms that are enabled to swap. These algorithms can be exchanged with another of the same name while the original is running. The "new" algorithm becomes active after an update command is sent. This "new" algorithm may again be swapped with another, and so on. This capability allows changing algorithm operation without stopping and leaving this and perhaps other processes without control.

Terminal Blocks The screw-terminal blocks you connect your system field wiring to. The terminal blocks are inside the Terminal Module

Terminal Module The plastic encased module which contains the terminal blocks you connect your field wiring to. The Terminal Module then is plugged into the HP E1422's front panel.

Update This is an intended change to an algorithm, algorithm variable, or global variable that is initiated by one of the commands ALG:SCALAR, ALG:ARRAY, ALG:DEFINE, ALG:SCAN:RATIO, or ALG:STATE. This change or "update" is considered to be pending until an update command is received. Several updates can be sent to the Update Queue, waiting for an update command to cause them to take effect synchronously. The update commands are ALG:UPDATE, and ALG:UPD:CHANNEL.

Update Queue A list of scalar variable values, and/or buffer pointer values (for arrays, and swapping algorithms) that is built in response to updates (see Update). When an update command is sent, scalar values and pointer values are sent to their working locations.

User Function A function callable from the Algorithm Language in the general form <*function_name*>**(** <*expression*> **)**. These user defined functions provide advanced mathematical capability to the Algorithm Language

Notes:

Appendix D
Wiring and Noise Reduction Methods

Separating Digital and Analog SCP Signals

Signals with very fast rise time can cause interference with nearby signal paths. This is called cross-talk. Digital signals present this fast rise-time situation. Digital I/O signal lines that are very close to analog input signal lines can inject noise into them.

To minimize cross-talk you can maximize the distance between analog input and digital I/O signal lines. By installing analog input SCPs in positions 0 through 3, and digital I/O SCPs in positions 4 through 7, you can keep these types of signals separated by the width of the HP E1422 module. The signals are further isolated because they remain separated on the connector module as well. Note that in Figure D-1 , even though only 7 of the eight SCP positions are filled, the SCPs present are not installed contiguously, but are arranged to provide as much digital/analog separation as possible.

If you have to mix analog input and digital I/O SCPs on the same side, the following suggestions will help provide quieter analog measurements.

- Use analog input SCPs that provide filtering on the mixed side.
- Route only high level analog signals to the mixed side.

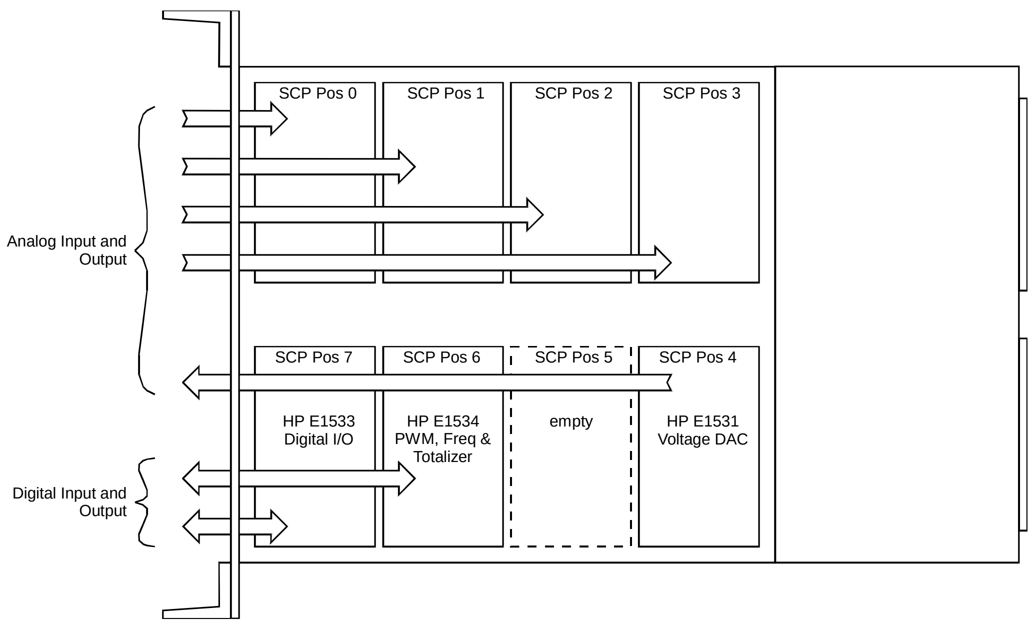


Figure D-1. Separating Analog and Digital Signals

Recommended Wiring and Noise Reduction Techniques

Unshielded signal wiring is very common in Data Acquisition applications. While this worked well for low speed integrating A/D measurements and/or for measuring high level signals, it does not work for high speed sampling A/Ds, particularly when measuring low level signals like thermocouples or strain gage bridge outputs. Unshielded wiring will pick up environmental noise, causing measurement errors. Shielded, twisted pair signal wiring, although it is expensive, is required for these measurements unless an even more expensive amplifier-at-the- signal-source or individual A/D at the source is used.

Generally, the shield should be connected to ground at the DUT and left open at the HP E1422. Floating DUTs or transducers are an exception. Connect the shield to HP E1422 GND or GRD terminals for this case, whichever gives the best performance. This will usually be the GND terminal. A single point shield to ground connection is required to prevent ground loops. This point should be as near to the noise source as possible and this is usually at the DUT.

Wiring Checklist

The following lists some recommended wiring techniques.

1. Use individually shielded, twisted-pair wiring for each channel.

2. Connect the shield of each wiring pair to the corresponding Guard (G) terminal on the Terminal Module .

3. The Terminal Module is shipped with the Ground-Guard (GND-GRD) shorting jumper installed for each channel. These may be left installed or removed, dependent on the following conditions:

   a. **Grounded Transducer with shield connected to ground at the transducer:** Low frequency ground loops (DC and/or 50/60Hz) can result if the shield is also grounded at the Terminal Module end. To prevent this, remove the GND-GRD jumper for that channel.

   b. **Floating Transducer with shield connected to the transducer at the source:** In this case, the best performance will most likely be achieved by leaving the GND-GRD jumper in place.

3. In general, the GND-GRD jumper can be left in place unless it is necessary to break low frequency (below 1 kHz) ground loops.

HP E1422 Guard Connections

The HP E1422 guard connection provides a 10 KΩ current limiting resistor between the guard terminals (G) and E1422 chassis ground for each 8 channel SCP bank. This is a safety device for the case where the Device Under Test (DUT) isn't actually floating, the shield is connected to the DUT and also connected to the HP E1422 guard terminal (G). The 10 KΩ resistor limits the ground loop current, which has been known to burn out shields. This also provides 20 KΩ isolation between shields between SCP banks which helps isolate the noise source.

Common Mode Voltage Limits

You must be very careful not to exceed the maximum common mode voltage referenced to the card chassis ground of ±16 volts (±60 volts with the HP E1513A Attenuator SCP). There is an exception to this when high frequency (1 kHz - 20 kHz) common mode noise is present (see “HP E1422 Noise Rejection” below). Also, if the DUT is not grounded, then the shield should be connected to the E1422 chassis ground.

When to Make Shield Connections

It is not always possible to state positively the best shield connection for all cases. Shield performance depends on the noise coupling mechanism which is very difficult to determine. The above recommendations are usually the best wiring method, but if feasible, experiment with shield connections to determine which provides the best performance for your installation and environment.

NOTE For a thorough, rigorous discussion of measurement noise, shielding, and filtering, see “Noise Reduction Techniques in Electronic Systems” by Henry W. Ott of Bell Laboratories, published by Wiley & Sons, ISBN 0-471-85068-3.

Noise Due to Inadequate Card Grounding

If either or both of the HP E1422 and HP E1482 (MXI Extender Modules) are not securely screwed into the VXIbus Mainframe, noise can be generated. Make sure that both screws (top and bottom) are screwed in tight. If not, it is possible that CVT data could be more noisy than FIFO data because the CVT is located in A24 space, the FIFO in A16 space; more lines moving could cause noisier readings.

HP E1422 Noise Rejection

Normal Mode Noise (Enm)

This noise is actually present at the signal source and is a differential noise (Hi to Lo). It is what is filtered out by the buffered filters on the HP E1502, E1503, E1508, and E1509 SCPs.

Common Mode Noise (Ecm)

This noise is common to both the Hi and Lo differential signal inputs. Low frequency Ecm is very effectively rejected by a good differential instrumentation amplifier, and it can be averaged out when measured through the Direct Input SCP (HP E1501). However, high frequency Ecm is rectified and generates an offset with the amplifier and filter SCPs (such as HP E1502, HP E1503, HP E1508, and HP E1509). This is since these SCPs have buffer-amplifiers on board and is a characteristic of amplifiers. The best way to deal with this is to prevent the noise from getting into the amplifier.

Keeping Common Mode Noise out of the Amplifier

Most common mode noise is about 60 Hz, so the differential amplifier rejection is very good. The amplifier Common Mode Noise characteristics are:

120 dB flat to 300 Hz, then 20 dB/octave rolloff

The HP E1422 amplifiers are selected for low gain error, offset, temperature drift, and low power. These characteristics are generally incompatible with good high frequency CMR performance. More expensive, high performance amplifiers can solve this problem, but they aren't required for many systems.

Shielded, twisted pair lead wire generally does a good job of keeping high frequency common mode noise out of the amplifier, provided the shield is connected to the HP E1422 chassis ground through a very low impedance. (Not via the guard terminal - The HP E1422 guard terminal connection shown in the HP E1422 User's manual does not consider the high frequency Ecm problem, and is there to limit the shield current and to allow the DUT to float up to some DC common mode voltage subject to the maximum ±16 volt input specification limit.

This conflicts with the often recommended good practice of grounding the shield at the signal source and only at that point to eliminate line frequency ground loops, which can be high enough to burn up a shield. We recommend that you follow this practice, and if you see high frequency common mode noise (or suspect it), tie the shield to the HP E1422 ground through a 0.1 µF capacitor. At high frequencies, this drives the shield voltage to 0 volts at the HP E1422 input. Due to inductive coupling to the signal leads, the Ecm voltage on the signal leads is also driven to zero.

Appendix E

Generating User Defined Functions

Introduction

The HP E1422A has a limited set of mathematical operations such as add, subtract, multiply, and divide. Many control applications require functions such as square root for calculating flow rate or a trigonometric function to correctly transition motion of moving object from a start to ending position. In order to represent a sine wave or other transcendental functions, one could use a power series expansion to approximate the function using a finite number of algebraic expressions. Since the above mentioned operations can take from 1.5usec to 4usec for each floating point calculation, a complex waveform such as sine(x) could take more than 100usec to get the desired result. A faster solution is desirable and available.

The HP E1422 provides a solution to approximating such complex waveforms by using a piece-wise linearization of virtually any complex waveform. The technique is simple. The Universal Instrument Drivers CD Rom supplied with your HP E1422 contains a 'C' program which calculates 128 Mx+B segments over a specified range of values for the desired function. You supply the function; the program generates the segments in a table. The resulting table can be downloaded into the HP E1422's RAM with the ALG:FUNC:DEF command where you can select any desired name of the function(i.e. sin(x), tan(x), etc.). Up to 32 functions can be created for use in algorithms. At runtime where the function is passed an 'x' value, the time to calculate the Mx+B segmented linear approximation is approximately 18µseconds.

The HP E1422 actually uses this technique to convert volts to temperature, strain, etc. The accuracy of the approximation is really based upon how well you select the range over which the table is built. For thermocouple temperature conversion, the HP E1422 fixes the range to the lowest A/D range(+/-64millivolts) so that small microvolt measurements yield the proper resolution of the actual temperature for a non-linear transducer. In addition, the HP E1422 permits you to create Custom Engineering Unit conversion for your transducer so that when the voltage measurement is actually made the EU conversion takes place(see SENS:FUNC:CUST). Algorithms deal with the resulting floating point numbers generated during the measurement phase and may require further complex mathematical operations to achieve the desired result.

With some complex waveforms, you may actually want to break up the waveform into several functions in order to get the desired accuracy. For example, suppose you need to generate a square root function for both voltage and strain calculations. The voltages are only going to range from 0 to +/-16volts, worst case. The strain measurements return numbers in microstrain which range in the 1000's. Trying to represent the square root function over the entire range would severely impact the accuracy of the approximation. Remember, the entire range is broken up into only 128 segments of Mx+B operations. If you want accuracy, you MUST limit the range over which calculations are made. Many transcendental functions are simply used as a scaling multiplier. For example, a sine wave function is typically created over a range of 360 degrees or 2*PI radians. After which, the function repeats itself. It's a simple matter to make sure the 'x' term is scaled to this range before calculating the result. This concept should be used almost exclusively to obtain the best results.

Haversine Example.

The following is an example of creating a haversine function (a sine wave over the range of -PI/2 to PI/2). The resulting function represents a fairly accurate approximation of this non-linear waveform when you limit the range as indicated. Since the tables must be built upon binary boundaries(i.e. .125, .25, .5, 1, 2, 4, etc.) and since PI/2 is a number greater than 1 but less than 2, the next binary interval to include this range will be 2. Another requirement for building the table is that the waveform range MUST be centered around 0(i.e. symmetrical about the X-axis). If the desired function is not defined on one side or the other of the Y-axis, then the table is right or left shifted by the offset from X=0 and the table values are calculated correctly, but the table is built as though it were centered about the X-axis. For the most part, you can ignore these last couple of sentences if it does not make sense to you. The only reason its brought up here is that your accuracy may suffer the farther away from the X=0 point you get unless you understand what resolution is available and how much non-linearity is present in your waveform. We'll talk about that in the "Limitations" section, later.

Figure 1 shows the haversine function as stated above. This type of waveform is typical of the kind of acceleration and deceleration one wants when moving an object from one point to another. The desired beginning point would be the location at -PI/2 and the ending point would be at PI/2. With the desired range spread over +/- PI/2, the 128 segments are actually divided over the range of +/- 2. Therefore, the 128 Mx+B line segments are divided equally on both sides of X=0: 64 segments for 0..2 and 64 segments for -2..0.

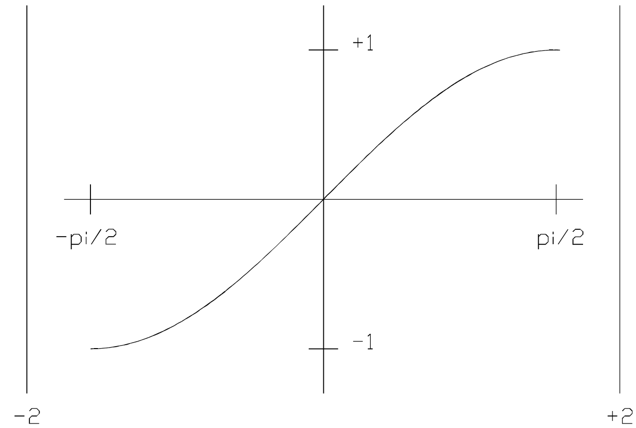


Figure E-1. A Haversine Function

A typical use of this function would be to output an analog voltage or current at each Scan Trigger of the HP E1422 and over the range of the haversine. For example, suppose you wanted a new position of an analog output to move from 1ma to 3ma over a period of 100msec. If your TRIG:TIMER setting or your EXTernal trigger was set to 2msec, then you would want to force 50 intervals over the range of the haversine. This can be easily done by using a scalar variable to count the number of times the algorithm has executed and to scale the variable value to the -PI/2 to PI/2 range. 3ma is multiplied times the custom function result over each interval which will yield the shape of the haversine (.003*sin(x)+.001). This is illustrated in the example below. The program (sine_fn.cpp on the CD illustrates the actual program used to generate this haversine function. You need only supply the algebraic expression in my_function(), the desired range over which to evaluate the function(which determines the table range), and the name of the function. The Build_table() routine creates the table for the function, and the ALG:FUNC:DEF writes that table into HP E1422 memory. The table MUST be built and downloaded BEFORE trying to use the function.

The following is a summary of what commands and parameters are used in the program example. Table 1 shows some examples of the accuracy of the custom function with various input values compared to an evaluation of the actual transcendental function found in 'C'. Please note that the Mx+B segments are located on boundaries specified by 2/64 on each side of X=0. This means that if you select the exact input value that was used for the beginning of each segment, you WILL get exactly the calculated value of that function at that point. Any point between segments will be an approximation dependent upon the linearity of that segment. Also note that values of X = 2 and X = -2 will result in Y=infinity.

'C' sin(-1.570798)	-1.000000	'HP E1422' sin(-1.570798)	-0.999905
'C' sin(-1.256639)	-0.951057	'HP E1422' sin(-1.256639)	-0.950965
'C' sin(-0.942479)	-0.809018	'HP E1422' sin(-0.942479)	-0.808944
'C' sin(-0.628319)	-0.587786	'HP E1422' sin(-0.628319)	-0.587740
'C' sin(-0.314160)	-0.309017	'HP E1422' sin(-0.314160)	-0.308998
'C' sin(0.000000)	0.000000	'HP E1422' sin(0.000000)	0.000000
'C' sin(0.314160)	0.309017	'HP E1422' sin(0.314160)	0.308998
'C' sin(0.628319)	0.587786	'HP E1422' sin(0.628319)	0.587740
'C' sin(0.942479)	0.809018	'HP E1422' sin(0.942479)	0.808944
'C' sin(1.256639)	0.951057	'HP E1422' sin(1.256639)	0.950965
'C' sin(1.570798)	1.000000	'HP E1422' sin(1.570798)	0.999905

Table 6-2. 'C' Sin(x) Vs. HP E1422 Haversine Function

Limitations

As stated earlier, there are limitations to using this custom function technique. These limitations are directly proportional to the non-linearity of the desired waveform. For example, suppose you wanted to represent the function X*X*X over a range of +/-1000. The resulting binary range would be +/-1024, and the segments would be partitioned at 1024/64 intervals. This means that every 16 units would yield an Mx+B calculation over that segment. As long as you input numbers VERY close to those cardinal points, you will get good results. Strictly speaking, you will get perfect results if you only calculate at the cardinal points, which may be reasonable for your application if you limit your input values to exactly those 128 points.

You may also shift the waveform anywhere along the X-axis, and Build_table() will provide the necessary offset calculations to generate the proper table. Be aware too that shifting the table out to greater magnitudes of X may also impact the precision of your results dependent upon the linearity of your waveform. Suffice it to say that you will get your best results and it will be easiest for you to grasp what your doing if you stay near the X=0 point since most of the results of your measurements will have 1e-6..16 values for volts.

One final note. You may see truncation errors in the fourth digit of your results. This is because only 15 bits of your input value is sent to the function. This occurs because the same technique used for Custom EU conversion is used here, and the method assumes input values are from the 16 bit A/D (15 bits = sign bit). This is evident in Table 1 where the first and last entries return ±0.9999 rather than ±1. For most applications this accuracy should be more than adequate.

Appendix F
Example PID Algorithm Listings

This appendix includes listings of the built-in PIDA and PIDB,as well as the more advanced PIDC which can be down loaded as a custom algorithm.

PIDA Algorithm

Figure F-1 shows the block diagram of the PIDA algorithm.

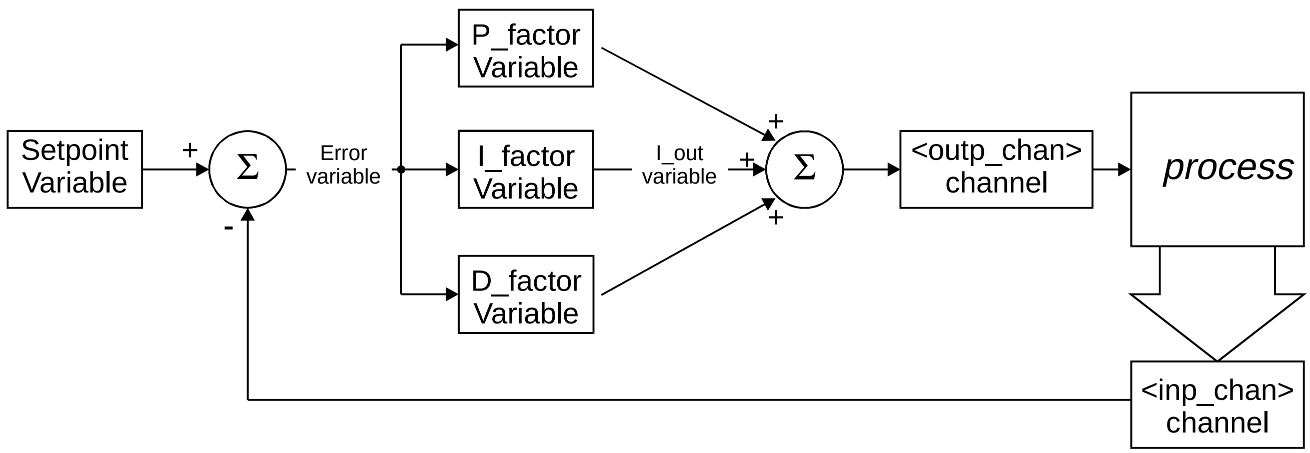


Figure F-1. The Simple PID Algorithm "PIDA"

PIDA algorithm implements the classic PID controller. This implementation was designed to be fast. In order to be fast, this algorithm provides no clipping limit, alarm limits, status management, or CVT/FIFO communication (History Modes). The algorithm performs the following calculations each time it is executed:

Error = Setpoint - <*inp_chan*>
I_out = I_out + I_factor * Error
<*outp_chan*> = P_factor * Error + I_out + D_factor * (Error - Error_old)
Error_old = Error.

PIDA Source Listing

```
/*********************************************************************************************/
/*  I/O Channels                                                                   */
/*  Must be defined by the user                                                    */
/*                                                                                 */
```

```
/* inchan - Input channel name                                            */
/* outchan - Output channel name                                          */
/*                                                                        */
/*****************************************************************************************/
/*                                                                        */
/*****************************************************************************************/
/* PID algorithm for E1415A controller module. This algorithm is called   */
/* once per scan trigger by main().  It performs Proportional, Integral   */
/* and Derivative control.                                                */
/*                                                                        */
/*                                                                        */
/* The output is derived from the following equations:                    */
/*                                                                        */
/* PID_out = P_out + I_out + D_out                                        */
/* P_out = Error * P_factor                                               */
/* I_out = I_out + (Error * I_factor)                                     */
/* D_out = ((Error - Error_old) * D_factor)                               */
/* Error = Setpoint - PV                                                  */
/*                                                                        */
/* where:                                                                 */
/*   Setpoint is the desired value of the process variable (user supplied)  */
/*   PV is process variable measured on the input channel                 */
/*   PID_out is the algorithm result sent to the output channel           */
/*   P_factor, I_factor, and D_factor are the PID constants(user supplied) */
/*                                                                        */
/*                                                                        */
/* At startup the output will abruptly change to P_factor*Error           */
/*                                                                        */
/*                                                                        */
/*****************************************************************************************/
/*                                                                        */
/* User determined control parameters                                     */
   static float Setpoint = 0;   /* The setpoint                           */
   static float P_factor = 1;   /* Proportional control constant          */
   static float I_factor = 0;   /* Integral control constant              */
   static float D_factor = 0;   /* Derivative control constant            */
/*                                                                        */
/*  Other Variables                                                       */
   static float I_out;     /* Integral term                               */
   static float Error;     /* Error term                                  */
   static float Error_old;       /* Last Error - for derivative           */
/*                                                                        */
/*PID algorithm code:                                                     */
/* Begin PID calculations */
/* First, find the Process Variable "error" */
/* This calculation has gain of minus one (-1) */
   Error = Setpoint - inchan;
/* On the first trigger after INIT, initialize the I and D terms */
   if (First_loop)
   {
/* Zero the I term and start integrating */
      I_out = Error * I_factor;
/* Zero the derivative term */
       Error_old = Error;
   }
/* On subsequent triggers, continue integrating */
   else /* not First trigger */
   {
```

```
        I_out = Error * I_factor + I_out;
    }
/* Sum PID terms */
    outchan = Error * P_factor + I_out + D_factor * (Error
- Error_old);
/* Save values for next pass */
        Error_old = Error;
```

PIDB Algorithm

Figure F-2 shows the block diagram of a more advanced algorithm that is favored in process control because of the flexibility allowed by its two differential terms. The "D" differential term is driven by changes in the process input measurement. The "SD" differential term is driven by changes in the setpoint variable value.

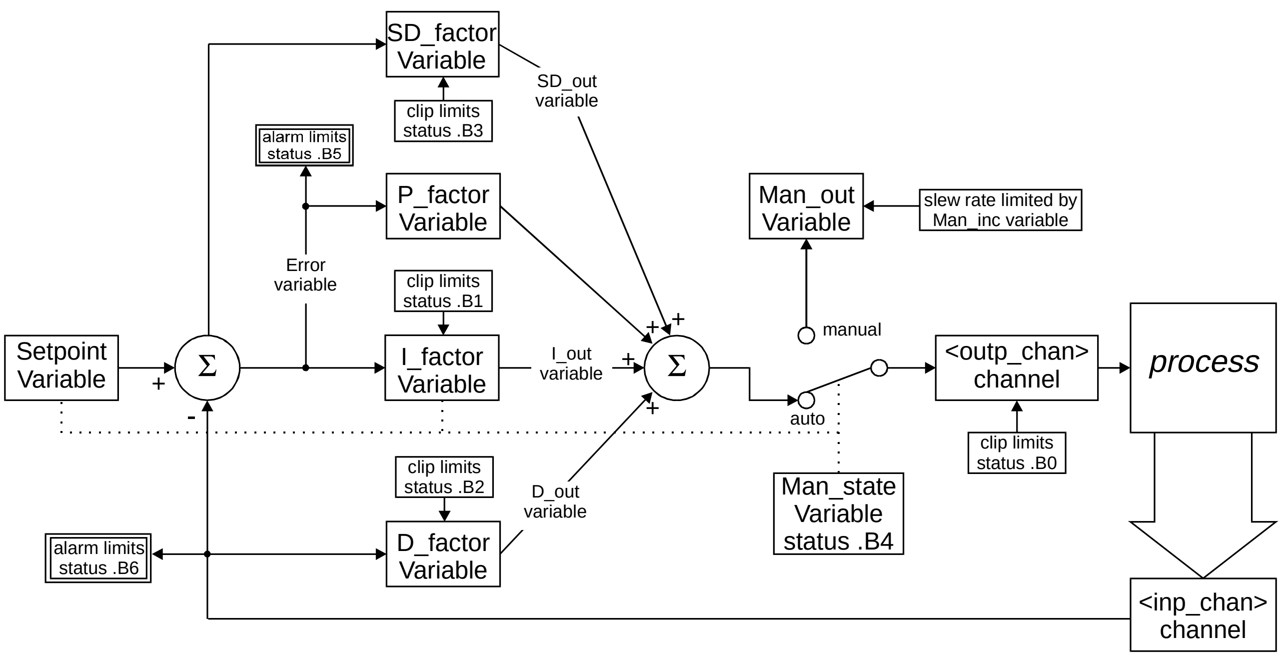


Figure F-2. The Advanced Algorithm "PIDB"

Clipping Limits

The PIDB algorithm provides clipping limits for its I, D, SD terms and the value sent to <*outp_chan*>. Values for these terms are not allowed to range outside of the set limits. The variables that control clipping are:

I term limits;	I_max, and I_min
D term limits;	D_max, and D_min
SD term limits;	SD_max, and SD_min
<*outp_chan*> limits;	Out_max, and Out_min

Alarm Limits

The PIDB algorithm provides Alarm Limits for the process variable PV and the Error term variable Error. If these limits are reached, the algorithm sets the value of <*alarm_chan*> true and generates a VXIbus interrupt. The variables that control alarm limits are:

Process Variable (from <*inp_chan*>);	PV_max, and PV_min
Error term alarm limits;	Error_max, and Error_min

The max and min limits for clipping and alarms are set to 9.9E+37 and -9.9E+37 respectively when the algorithm is defined. This effectively turns the limits off until you change these values with the ALG:SCALAR and ALG:UPDATE commands as described in "Pre-setting PID Variables and Coefficients" later in this section.

Manual Control

The PIDB algorithm provides for manual control with "bumpless" transfer between manual and automatic control. The variables that control the manual mode are:

Auto/Manual control;	Man_state (0 = automatic (default), 1 = manual)
Manual output control;	Man_out (defaults to current auto value)
Manual control slew rate;	Man_inc (defaults to 9.99E+37 (fast change))

Use the ALG:SCALAR and ALG:UPDATE commands to change the manual control variables before or after the algorithm is running.

Status Variable

The PIDB algorithm uses 7 bits in a status variable (Status) to record the state of clipping and alarm limits, and the automatic/manual mode. When a limit is reached or the manual mode is set, the algorithm sets a status bit to 1.

Output (<outp_chan>) at clipping limit;	Status.B0
I term (I_out) at clipping limit;	Status.B1
D term (D_out) reached at limit;	Status.B2
SD term (SD_out) at clipping limit;	Status.B3
Control mode (Man_state) is manual;	Status.B4
Error term (Error) out of limits;	Status.B5
Process Variable (<*inp_chan*>) out of limits;	Status.B6

History Mode

The PIDB algorithm provides two modes of reporting the values of its operating variables. A variable *History_mode* controls the two modes. The default history mode (*History_mode* = 0) places the following algorithm values into elements of the Current Value Table (the CVT):

Process Variable (<*inp_chan*>) value to CVT element	(10 * n) + 0
Error Term variable (Error) value to CVT element	(10 * n) + 1
Output (<outp_chan>) value to CVT element	(10 * n) + 2
Status word bits 0 through 6 (Status) to CVT element	(10 * n) + 3

Where n is the number of the algorithm from ’ALGn’
So ALG1 places values into CVT elements 10 through 13, ALG2 places values in CVT elements 20 through 23 ... ALG32 places values into CVT elements 320 through 323

When you set *History_mode* to 1, the operating values are sent to the CVT as above and they are sent to the FIFO buffer as well. The algorithm writes a header entry first. The header value is (n * 256) + 4, where n is the algorithm number from ’ALGn’, and the number 4 indicates the number of FIFO entries that follow for this algorithm. This identifies which PIDB algorithm the 5 element FIFO entry is from.

PIDB Source Listing

```
/******************************************************************************************/
/*  PID_B                                                                      */
/******************************************************************************************/
/*  I/O Channels                                                               */
/*  Must be defined by the user                                                */
/*                                                                             */
/* inchan - Input channel name                                                 */
/* outchan - Output channel name                                               */
/* alarmchan  - Alarm channel name                                             */
/*                                                                             */
/******************************************************************************************/
/*                                                                             */
/******************************************************************************************/
/* PID algorithm for E1415A controller module.  This algorithm is called   */
/* once per scan trigger by main().  It performs Proportional, Integral       */
/* and Derivative control.                                                     */
/*                                                                             */
/*                                                                             */
/* The output is derived from the following equations:                        */
/*                                                                             */
/* PID_out = P_out + I_out + D_out + SD_out                                    */
/* P_out = Error * P_factor                                                    */
/* I_out = I_out + (Error * I_factor)                                          */
/* D_out = ((PV_old - PV) * D_factor)                                          */
/* SD_out = (Setpoint - Setpoint_old) * SD_factor                              */
/* Error = Setpoint - PV                                                       */
/*                                                                             */
/* where:                                                                      */
/*   Setpoint is the desired value of the process variable (user supplied)  */
/*   PV is the process variable measured on the input channel                 */
/*   PID_out is the algorithm result sent to the output channel               */
/*   P_factor, I_factor, D_factor, and SD_factor are the PID constants        */
/*  (user supplied)                                                            */
/*                                                                             */
/* Alarms may be generated when either the Process Variable or the           */
/* error exceeds user supplied limits.  The alarm condition will cause       */
/* an interrupt to the host computer, set the (user-specified) alarm          */
/* channel output to one (1), and set a bit in the Status variable to        */
/* one (1).  The interrupt is edge-sensitive. ( It will be asserted only     */
/* on the transition into the alarm state.)  The alarm channel digital       */
/* output will persist for the duration of all alarm conditions.  The       */
/* Status word bits will also persist for the alarm duration.  No user       */
/* intervention is required to clear the alarm outputs.                      */
/*                                                                             */
/* This version provides for limiting (or clipping) of the Integral,         */
/* Derivative, Setpoint Derivative, and output to user specified limits.     */
/* The Status Variable indicates when terms are being clipped.               */
/*                                                                             */
/* Manual control is activated when the user sets the Man_state variable */
/* to a non-zero value.  The output will be held at its last value.  The     */
/* user can change the output by changing the Man_out variable.  User   */
/* initiated changes in Man_out will cause the output to slew to the         */
/* Man_out value at a rate of Man_inc per scan trigger.                      */
/*                                                                             */
/* Manual control causes the Setpoint to continually change to match        */
```

```
/* the Process Variable, and the Integral term to be constantly updated     */
/* to the output value such that a return to automatic control will        */
/* be bumpless and will use the current Process Variable value as the      */
/* new setpoint.                                                           */
/* The Status variable indicates when the Manual control mode is active.   */
/*                                                                         */
/* At startup in the Manual control mode, the output will slew to Man_out  */
/* at a rate of Man_inc per scan trigger.                                  */
/*                                                                         */
/* At startup, in the Automatic control mode, the output will abruptly     */
/* change to P_factor * Error.                                             */
/*                                                                         */
/* For process monitoring, data may be sent to the FIFO and current        */
/* value table (CVT).  There are two levels of data logging, controlled    */
/* by the History_mode variable.  The location in the CVT is based         */
/* on 'n', where n is the algorithm number (as returned by ALG_NUM, for    */
/* example).  The first value is placed in the (10 * n)th 32-bit word of   */
/* the CVT.  The other values are written in subsequent locations.         */
/*                                                                         */
/*  History_mode = 0:  Summary to CVT only.  In this mode, four values     */
/*    are output to the CVT.                                               */
/*                                                                         */
/*      Location     Value                                                 */
/*          0        Input                                                 */
/*          1        Error                                                 */
/*          2        Output                                                */
/*          3        Status                                                */
/*                                                                         */
/*  History_mode = 1:  Summary to CVT and FIFO.  In this mode, the four    */
/*    summary values are written to both the CVT and FIFO.  A header       */
/*    tag (256 * n + 4) is sent to the FIFO first, where n is the Algorithm */
/*    number (1 - 32).<N>                                                  */
/*                                                                         */
/******************************************************************************************/
/*                                                                         */
/* User determined control parameters                                      */
   static float Setpoint = 0; /* The setpoint                              */
   static float P_factor = 1; /* Proportional control constant             */
   static float I_factor = 0; /* Integral control constant                 */
   static float D_factor = 0; /* Derivative control constant               */
   static float Error_max = 9.9e+37; /* Error alarm limits                 */
   static float Error_min = -9.9e+37;
   static float PV_max = 9.9e+37; /* Process Variable alarm limits         */
   static float PV_min = -9.9e+37;
   static float Out_max = 9.9e+37; /* Output clip limits                   */
   static float Out_min = -9.9e+37;
   static float D_max = 9.9e+37; /* Derivative clip limits                 */
   static float D_min = 9.9e+37;
   static float I_max = 9.9e+37; /* Integral clip limits                   */
   static float I_min = -9.9e+37;
   static float Man_state = 0; /* Activates manual control                 */
   static float Man_out = 0; /* Target Manual output value                 */
   static float Man_inc = 9.9e+37;/* Manual outout change increment        */
   static float SD_factor = 0; /* Setpoint Derivative constant             */
   static float SD_max = 9.9e+37; /* Setpoint Derivative clip limits       */
   static float SD_min = 9.9e+37;
   static float History_mode = 0; /* Activates fifo data logging           */
/*                                                                         */
```

```
/*  Other Variables                                                  */
    static float I_out; /* Integral term                             */
    static float D_out; /* Derivative term                           */
    static float Error; /* Error term                                */
    static float PV_old; /* Last process variable                    */
    static float Setpoint_old; /* Last setpoint - for derivative     */
    static float SD_out; /* Setpoint derivative term                 */
    static float Status = 0; /* Algorithm status word                */
                            /*                                       */
                            /*  B0 - PID_out at clip limit           */
                            /*  B1 - I_out at clip limit             */
                            /*  B2 - D_out at clip limit             */
                            /*  B3 - SD_out at clip limit            */
                            /*  B4 - in Manual control mode          */
                            /*  B5 - Error out of limits             */
                            /*  B6 - PV out of limits                */
                            /*  others - unused                      */
                            /*                                       */
/*                                                                   */
/*PID algorithm code:                                                */
/* Test for Process Variable out of limits                           */
    if ( (inchan  >>  PV_max) || ( PV_min  >>  inchan ) ) /* PV alarm test */
    {
        if ( !Status.B6 )
        {
         Status.B6 = 1;
         alarmchan = 1;
         interrupt();
        }
    }
    else
    {
        Status.B6 = 0;
    }
/* Do this when in the Manual control mode */
  if ( Man_state )
  {
/* Slew output towards Man_out */
    if (Man_out  >>  outchan + abs(Man_inc))
    {
        outchan = outchan + abs(Man_inc);
    }
    else if (outchan  >>  Man_out + abs(Man_inc))
    {
        outchan = outchan - abs(Man_inc);
    }
    else
    {
        outchan = Man_out;
    }
/* Set manual mode bit in status word */
    Status.B4 = 1;
/* No error alarms while in Manual mode */
    Status.B5 = 0;
/* In case we exit manual mode on the next trigger */
/* Set up for bumpless transfer */
    I_out = outchan;
    Setpoint = inchan;
```

```
   PV_old = inchan;
   Setpoint_old = inchan;
  }
/* Do PID calculations when not in Manual mode */
  else  /* if ( Man_state )  */
  {
   Status.B4 = 0;
/* First, find the Process Variable "error" */
/* This calculation has gain of minus one (-1) */
   Error = Setpoint - inchan;
/* Test for error out of limits */
   if ( (Error  >>  Error_max) || (Error_min  >>  Error) )
   {
       if ( !Status.B5 )
       {
           Status.B5 = 1;
           alarmchan = 1;
           interrupt();
       }
   }
   else
   {
       Status.B5 = 0;
   }
/* On the first trigger after INIT, initialize the I and D terms */
   if (First_loop)
   {
/* Zero the I term and start integrating */
      I_out = Error * I_factor;
/* Zero the derivative terms */
       PV_old = inchan;
       Setpoint_old = Setpoint;
   }
/* On subsequent triggers, continue integrating */
   else /* not First trigger */
   {
       I_out = Error * I_factor + I_out;
   }
/* Clip the Integral term to specified limits */
   if ( I_out   >>  I_max )
   {
       I_out = I_max;
       Status.B1=1;
   }
   else if ( I_min  >>  I_out )
   {
       I_out = I_min;
       Status.B1=1;
   }
   else
   {
       Status.B1 = 0;
   }
/* Calculate the Setpoint Derivative term */
   SD_out = SD_factor * ( Setpoint - Setpoint_old );
/* Clip to specified limits */
   if ( SD_out  >>  SD_max )/* Clip Setpoint derivative */
   {
```

```
        SD_out = SD_max;
        Status.B3=1;
    }
    else if ( SD_min  >>  SD_out )
    {
        SD_out = SD_min;
        Status.B3=1;
    }
    else
    {
        Status.B3 = 0;
    }
/* Calculate the Error Derivative term */
    D_out = D_factor *( PV_old - inchan );
/* Clip to specified limits */
    if ( D_out   >>  D_max )/* Clip derivative */
    {
        D_out = D_max;
        Status.B2=1;
    }
    else if ( D_min  >>  D_out )
    {
        D_out = D_min;
        Status.B2=1;
    }
    else
    {
        Status.B2 = 0;
    }
/* Sum PID&SD terms */
    outchan = Error * P_factor + I_out + D_out + SD_out;
/* Save values for next pass */
        PV_old = inchan;
        Setpoint_old = Setpoint;
/* In case we switch to manual on the next pass */
/* prepare to hold output at latest value */
    Man_out = outchan;
  }  /* if ( Man_state ) */
/* Clip output to specified limits */
    if ( outchan   >>  Out_max )
    {
        outchan = Out_max;
        Status.B0=1;
    }
    else if ( Out_min  >>  outchan )
    {
        outchan = Out_min;
        Status.B0=1;
    }
    else
    {
        Status.B0 = 0;
    }
/* Clear alarm output if no alarms */
    if (!(Status.B6 || Status.B5) ) alarmchan = 0;
/* Log appropriate data */
    if ( History_mode )
    {
```

```
/* Output summary to FIFO & CVT */
        writefifo( (ALG_NUM*256)+4 );
        writeboth( inchan, (ALG_NUM*10)+0 );
        writeboth( Error, (ALG_NUM*10)+1);
        writeboth( outchan, (ALG_NUM*10)+2);
        writeboth( Status, (ALG_NUM*10)+3  );
    }
    else
    {
/* Output summary to CVT only */
        writecvt( inchan, (ALG_NUM*10)+0 );
        writecvt( Error, (ALG_NUM*10)+1);
        writecvt( outchan, (ALG_NUM*10)+2);
        writecvt( Status, (ALG_NUM*10)+3  );
    }
```

PIDC Algorithm

PIDC is very similar to PIDB with the addition of extended history mode. See comments in source code below.

PIDC Source Listing

```
/******************************************************************************************/
/*  PID_C                                                                  */
/******************************************************************************************/
/*  I/O Channels                                                           */
/*  Must be defined by the user                                            */
/*                                                                         */
/* inchan - Input channel name                                             */
/* outchan - Output channel name                                           */
/* alarmchan  - Alarm channel name                                         */
/*                                                                         */
/******************************************************************************************/
/*                                                                         */
/******************************************************************************************/
/* PID algorithm for E1415A controller module.  This algorithm is called   */
/* once per scan trigger by main().  It performs Proportional,             */
Integral                                                                   */
/* and Derivative control.                                                 */
/*                                                                         */
/*                                                                         */
/* The output is derived from the following equations:                     */
/*                                                                         */
/* PID_out = P_out + I_out + D_out + SD_out                                */
/* P_out = Error * P_factor                                                */
/* I_out = I_out + (Error * I_factor)                                      */
/* D_out = ((PV_old - PV) * D_factor)                                      */
/* SD_out = (Setpoint - Setpoint_old) * SD_factor                          */
/* Error = Setpoint - PV                                                   */
/*                                                                         */
/* where:                                                                  */
/*   Setpoint is the desired value of the process variable (user supplied) */
/*   PV is the process variable measured on the input channel              */
/*   PID_out is the algorithm result sent to the output channel            */
```

```
/*   P_factor, I_factor, D_factor, and SD_factor are the PID constants       */
/*  (user supplied)                                                          */
/*                                                                           */
/*  Alarms may be generated when either the ProcessVariable or the           */
/*  error exceeds user supplied limits.                                      */
/*  The alarm condition will cause an interrupt to the host computer,        */
/*  set the (user-specified) alarm channel output to one (1), and set a bit  */
/*  in the Status variable to one (1).                                       */
/*  The interrupt is edge-sensitive. ( It will be asserted only              */
/* on the transition into the alarm state.)  The alarm channel digital       */
/* output will persist for the duration of all alarm conditions.  The        */
/* Status word bits will also persist for the alarm duration.  No user       */
/* intervention is required to clear the alarm outputs.                      */
/*                                                                           */
/* This version provides for limiting (or clipping) of the Integral,         */
/* Derivative, Setpoint Derivative, and output to user specified limits.     */
/* The Status Variable indicates when terms are being clipped.               */
/*                                                                           */
/* Manual control is activated when the user sets the Man_state variable */
/* to a non-zero value.  The output will be held at its last value.  The     */
/* user can change the output by changing the Man_out variable.  User        */
/* initiated changes in Man_out will cause the output to slew to the         */
/* Man_out value at a rate of Man_inc per scan trigger.                      */
/*                                                                           */
/* Manual control causes the Setpoint to continually change to match         */
/* the Process Variable, and the Integral term to be constantly updated      */
/* to the output value such that a return to automatic control will          */
/* be bumpless and will use the current Process Variable value as the        */
/* new setpoint.                                                             */
/* The Status variable indicates when the Manual control mode is active. */
/*                                                                           */
/* At startup in the Manual control mode, the output will be held at         */
/* its current value.                                                        */
/*                                                                           */
/* At startup, in the Automatic control mode, the output will slew           */
/* from its initial value towards P_factor * Error at a rate determined      */
/* by the Integral control constant (I_out is initialized to cancel P_out).  */
/*                                                                           */
/* For process monitoring, data may be sent to the FIFO and current          */
/* value table (CVT).  There are three levels of data logging, controlled    */
/* by the History_mode variable.  The location in the CVT is based           */
/* on 'n', where n is the algorithm number (as returned by ALG_NUM, for*/
/* example).  The first value is placed in the (10 * n)th 32-bit word of     */
/* the CVT.  The other values are written in subsequent locations.           */
/*                                                                           */
/*  History_mode = 0:  Summary to CVT only.  In this mode, four values  */
/*    are output to the CVT.                                                 */
/*                                                                           */
/*        Location      Value                                                */
/*            0         Input                                                */
/*            1         Error                                                */
/*            2         Output                                               */
/*            3         Status                                               */
/*                                                                           */
/*  History_mode = 1:  Summary to CVT and FIFO.  In this mode, the four*/
/*    summary values are written to both the CVT and FIFO.  A header         */
/*    tag (256 * n + 4) is sent to the FIFO first.                           */
/*                                                                           */
```

```
/*  History_mode = 2:  All to FIFO and CVT.  In this mode, nine values     */
/*    are output to both the CVT and FIFO.  A header tag (256 * n + 9)     */
/*    is sent to the FIFO first.                                           */
/*                                                                         */
/*      Location      Value                                                */
/*          0         Input                                                */
/*          1         Error                                                */
/*          2         Output                                               */
/*          3         Status                                               */
/*          4         Setpoint                                             */
/*          5         Proportional term                                    */
/*          6         Integral term                                        */
/*          7         Derivative term                                      */
/*          8         Setpoint Derivative term                             */
/*                                                                         */
/*******************************************************************************************/
/*                                                                         */
/* User determined control parameters                                      */
   static float Setpoint = 0; /* The setpoint                              */
   static float P_factor = 1; /* Proportional control constant             */
   static float I_factor = 0; /* Integral control constant                 */
   static float D_factor = 0; /* Derivative control constant               */
   static float Error_max = 9.9e+37; /* Error alarm limits                 */
   static float Error_min = -9.9e+37;
   static float PV_max = 9.9e+37; /* Process Variable alarm limits         */
   static float PV_min = -9.9e+37;
   static float Out_max = 9.9e+37; /* Output clip limits                   */
   static float Out_min = -9.9e+37;
   static float D_max = 9.9e+37; /* Derivative clip limits                 */
   static float D_min = 9.9e+37;
   static float I_max = 9.9e+37; /* Integral clip limits                   */
   static float I_min = -9.9e+37;
   static float Man_state = 0; /* Activates manual control                 */
   static float Man_out = 0; /* Target Manual output value                 */
   static float Man_inc = 0; /* Manual outout change increment             */
   static float SD_factor = 0; /* Setpoint Derivative constant             */
   static float SD_max = 9.9e+37; /* Setpoint Derivative clip limits       */
   static float SD_min = 9.9e+37;
   static float History_mode = 0; /* Activates fifo data logging           */
/*                                                                         */
/*  Other Variables                                                        */
   static float I_out; /* Integral term                                    */
   static float P_out; /* Proportional term                                */
   static float D_out; /* Derivative term                                  */
   static float Error; /* Error term                                       */
   static float PV_old; /* Last process variable                           */
   static float Setpoint_old; /* Last setpoint - for derivative            */
   static float SD_out; /* Setpoint derivative term                        */
   static float Status = 0; /* Algorithm status word                       */
                              /*                                           */
                              /*  B0 - PID_out at clip limit               */
                              /*  B1 - I_out at clip limit                 */
                              /*  B2 - D_out at clip limit                 */
                              /*  B3 - SD_out at clip limit                */
                              /*  B4 - in Manual control mode              */
                              /*  B5 - Error out of limits                 */
                              /*  B6 - PV out of limits                    */
                              /*  others - unused                          */
```

```
                              /*                                          */
/*                                                                        */
/*PID algorithm code:                                                     */
/* Test for Process Variable out of limits                                */
   if ( (inchan  >>  PV_max) || ( PV_min  >>  inchan ) )  /* PV alarm test */
   {
        if ( !Status.B6 )
        {
          Status.B6 = 1;
          alarmchan = 1;
          interrupt();
        }
   }
   else
   {
        Status.B6 = 0;
   }
/* Do this when in the Manual control mode */
  if ( Man_state )
  {
/* On the first trigger after INIT only */
   if (First_loop)
   {
      Man_out= outchan;/* Maintain output at manual smooth start */
   }
/* On subsequent triggers, slew output towards Man_out */
   else if (Man_out  >>  outchan + abs(Man_inc))
   {
        outchan = outchan + abs(Man_inc);
   }
   else if (outchan  >>  Man_out + abs(Man_inc))
   {
        outchan = outchan - abs(Man_inc);
   }
   else
   {
        outchan = Man_out;
   }
/* Set manual mode bit in status word */
   Status.B4 = 1;
/* No error alarms while in Manual mode */
   Status.B5 = 0;
/* In case we exit manual mode on the next trigger */
/* Set up for bumpless transfer */
   I_out = outchan;
   Setpoint = inchan;
   PV_old = inchan;
   Setpoint_old = inchan;
  }
/* Do PID calculations when not in Manual mode */
  else  /* if ( Man_state )  */
  {
   Status.B4 = 0;
/* First, find the Process Variable "error" */
/* This calculation has gain of minus one (-1) */
   Error = Setpoint - inchan;
/* Test for error out of limits */
   if ( (Error  >>  Error_max) || (Error_min  >>  Error) )
```

```
    {
         if ( !Status.B5 )
         {
             Status.B5 = 1;
             alarmchan = 1;
             interrupt();
         }
    }
    else
    {
         Status.B5 = 0;
    }
/* On the first trigger after INIT, initialize the I and D terms */
    if (First_loop)
    {
/* For no abrupt output change at startup make the I term cancel the P term */
         I_out = outchan + Error * ( I_factor - P_factor );
/* Zero the derivative terms */
         PV_old = inchan;
         Setpoint_old = Setpoint;
    }
/* On subsequent triggers, continue integrating */
    else /* not First trigger */
    {
         I_out = Error * I_factor + I_out;
    }
/* Clip the Integral term to specified limits */
    if ( I_out   >>  I_max )
    {
         I_out = I_max;
         Status.B1=1;
    }
    else if ( I_min  >>  I_out )
    {
         I_out = I_min;
         Status.B1=1;
    }
    else
    {
         Status.B1 = 0;
    }
/* Calculate the Setpoint Derivative term */
    SD_out = SD_factor * ( Setpoint - Setpoint_old );
/* Clip to specified limits */
    if ( SD_out  >>  SD_max )/* Clip Setpoint derivative */
    {
         SD_out = SD_max;
         Status.B3=1;
    }
    else if ( SD_min  >>  SD_out )
    {
         SD_out = SD_min;
         Status.B3=1;
    }
    else
    {
         Status.B3 = 0;
    }
```

```
/* Calculate the Error Derivative term */
   D_out = D_factor *( PV_old - inchan );
/* Clip to specified limits */
   if ( D_out   >>  D_max )/* Clip derivative */
   {
        D_out = D_max;
        Status.B2=1;
   }
   else if ( D_min  >>  D_out )
   {
        D_out = D_min;
        Status.B2=1;
   }
   else
   {
        Status.B2 = 0;
   }
/* Calculate Proportional term */
   P_out = Error * P_factor;
/* Sum PID&SD terms */
   outchan = P_out + I_out + D_out + SD_out;
/* Save values for next pass */
        PV_old = inchan;
        Setpoint_old = Setpoint;
/* In case we switch to manual on the next pass */
/* prepare to hold output at latest value */
   Man_out = outchan;
  }  /* if ( Man_state ) */
/* Clip output to specified limits */
   if ( outchan   >>  Out_max )
   {
        outchan = Out_max;
        Status.B0=1;
   }
   else if ( Out_min  >>  outchan )
   {
        outchan = Out_min;
        Status.B0=1;
   }
   else
   {
        Status.B0 = 0;
   }
/* Clear alarm output if no alarms */
   if (!(Status.B6 || Status.B5) ) alarmchan = 0;
/* Log appropriate data */
   if ( History_mode  >>  1 )
   {
/* Output everything to FIFO & CVT */
        writefifo( (ALG_NUM*256)+9 );
        writeboth( inchan, (ALG_NUM*10)+0 );
        writeboth( Error, (ALG_NUM*10)+1);
        writeboth( outchan, (ALG_NUM*10)+2);
        writeboth( Status, (ALG_NUM*10)+3  );
        writeboth( Setpoint, (ALG_NUM*10)+4 );
        writeboth( P_out, (ALG_NUM*10)+5 );
        writeboth( I_out, (ALG_NUM*10)+6 );
        writeboth( D_out, (ALG_NUM*10)+7 );
```

```
        writeboth( SD_out, (ALG_NUM*10)+8 );
    }
    else if ( History_mode )
    {
/* Output summary to FIFO & CVT */
        writefifo( (ALG_NUM*256)+4 );
        writeboth( inchan, (ALG_NUM*10)+0 );
        writeboth( Error, (ALG_NUM*10)+1);
        writeboth( outchan, (ALG_NUM*10)+2);
        writeboth( Status, (ALG_NUM*10)+3  );
    }
    else
    {
/* Output summary to CVT only */
        writecvt( inchan, (ALG_NUM*10)+0 );
        writecvt( Error, (ALG_NUM*10)+1);
        writecvt( outchan, (ALG_NUM*10)+2);
        writecvt( Status, (ALG_NUM*10)+3  );
    }
```

Symbols

Numerics

A

D

N

O

P

Q

R

S

T

Z