VT1422A Remote Channel Multi-Function DAC Module

with VT1529A/B 32 Ch Remote Strain Conditioning Unit and VT1539A Remote Channel Signal Conditioning Plug-On

User's and SCPI Programming Manual

Where to Find it - Online and Printed Information

Module configuration and wiring	This Manual
SCPI programming	This Manual
SCPI example programs	This Manual, Driver Disc
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	VISA User's Guide
VEE programming information	VEE User's Manual

Manual Part Number: 82-0076-000
Printed in U.S.A. August 15, 2005

VXI TECHNOLOGY WARRANTY STATEMENT

PRODUCT: VT1422A Remote Channel Multi-function DAC Module,
VT1529A/B Remote Strain Conditioning Module,
VT1539A Remote Channel Signal Conditioning Plug-on,
and all other applicable Signal Conditioning Plug-ons

DURATION OF WARRANTY: 3 years

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

2. VXI Technology warrants that VXI Technology 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 VXI Technology receives notice of such defects during the warranty period, VXI Technology will replace software media which does not execute its programming instructions due to such defects.

3. VXI Technology does not warrant that the operation of VXI Technology products will be interrupted or error free. If VXI Technology 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. VXI Technology 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 VXI Technology. If customer schedules or delays VXI Technology 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 VXI Technology, (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 VXI TECHNOLOGY SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTY OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, AND FITNESS FOR A PARTICULAR PURPOSE.

8. VXI Technology 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 VXI Technology 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 VXI TECHNOLOGY 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 VXI Technology standard software agreement for the product involved.

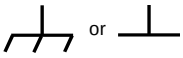
VXI Technology

VT1422A Remote Channel Multi-Function DAC Module User's and SCPI Programming Manual

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.

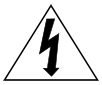
Indicates the field wiring terminal that must be connected to earth ground before operating the equipment—protects against electrical shock in case of fault.



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. VXI Technology 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 VXI Technology 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 VXI Technology 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.

Note for European Customers

If this symbol appears on your product, it indicates that it was manufactured after August 13, 2005. This mark is placed in accordance with *EN 50419, Marking of electrical and electronic equipment in accordance with Article 11(2) of directive 2002/96/EC (WEEE).* End-of-life product can be returned to VTI by obtaining an RMA number. Fees for recycling will apply if not prohibited by national law. SCP cards for use with the VT1422A have this mark placed on their packaging due to the densely populated nature of these cards.

DECLARATION OF CONFORMITY

According to ISO/IEC Guide 22 and CEN/CENELEC EN 45014

CE

Manufacturer's Name: VXI Technology, Inc.
Manufacturer's Address: 2031 Main Street
Irvine, California 92614
USA

Declares, that the product

Product Name: Remote Channel Multi-function DAC Module
Model Number: VT1422A
Product Options: *This declaration covers all options of the above product(s).*

Conforms with the following European Directives:

The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (including 93/68/EEC) and carries the CE Marking accordingly.

Conforms with the following product standards:

EMC	Standard	Limit
	IEC 61326-1:1997+A1:1998 / EN 61326-1:1997+A1:1998	
	CISPR 11:1990 / EN 55011:1991	*Group 1 Class A*
	IEC 61000-4-2:1995+A1:1998 / EN 61000-4-2:1995	*4kV CD, 8kV AD*
	IEC 61000-4-3:1995 / EN 61000-4-3:1995	*3 V/m, 80-1000 MHz*
	IEC 61000-4-4:1995 / EN 61000-4-4:1995	*0.5kV signal lines, 1kV power lines*
	IEC 61000-4-5:1995 / EN 61000-4-5:1995	*0.5 kV line-line, 1 kV line-ground*
	IEC 61000-4-6:1996 / EN 61000-4-6:1996	*3V, 0.15-80 MHz*
	IEC 61000-4-11:1994 / EN 61000-4-11:1994	*Dips: 30% 10ms; 60% 100 ms* *Interrupt > 95%@5000 ms*
	Canada: ICES-001:1998 *Australia/New Zealand: AS/NZS 2064.1*	
	The product was tested in a typical configuration with VXI Technology test systems.	
Safety	*IEC 61010-1:1990+A1:1992+A2:1995 / EN 61010-1:1993+A2:1995* *Canada: CSA C22.2 No. 1010.1:1992* *UL 3111-1: 1994*	

15 March 2002
Date

Steve Mauga
Quality Assurance Manager

DECLARATION OF CONFORMITY

According to ISO/IEC Guide 22 and CEN/CENELEC EN 45014

CE

Manufacturer's Name: VXI Technology, Inc.
Manufacturer's Address: 2031 Main Street
Irvine, California 92614
USA

Declares, that the product

Product Name: Remote Strain Conditioning Unit
Model Number: VT1529A, VT1529B
Product Options: *This declaration covers all options of the above product(s).*

Conforms with the following European Directives:

The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (including 93/68/EEC) and carries the CE Marking accordingly.

Conforms with the following product standards:

EMC	Standard	Limit
	IEC 61326-1:1997+A1:1998 / EN 61326-1:1997+A1:1998	
	CISPR 11:1990 / EN 55011:1991	*Group 1 Class A*
	IEC 61000-4-2:1995+A1:1998 / EN 61000-4-2:1995	*4kV CD, 8kV AD*
	IEC 61000-4-3:1995 / EN 61000-4-3:1995	*3 V/m, 80-1000 MHz*
	IEC 61000-4-4:1995 / EN 61000-4-4:1995	*0.5kV signal lines, 1kV power lines*
	IEC 61000-4-5:1995 / EN 61000-4-5:1995	*0.5 kV line-line, 1 kV line-ground*
	IEC 61000-4-6:1996 / EN 61000-4-6:1996	*3V, 0.15-80 MHz I cycle, 100%*
	IEC 61000-4-11:1994 / EN 61000-4-11:1994	*Dips: 30% 10 ms; 60% 100 ms* *Interrupt > 95%@5000 ms*
	Canada ICES-001 *Australia AS/NZS/2064* *Russia GOST 23450-79* *Czech Republic CSN EN55011* *Hungary MSZ EN55011*	
	The product was tested in a typical configuration with VXI Technology test systems.	
Safety	*IEC 61010-1:1990+A1:1992+A2:1995 / EN 61010-1:1993+A2:1995* *Canada: CSA C22.2 No. 1010.1:1992* *UL 3111-1*	

15 March 2002
Date

Steve Mauga
Quality Assurance Manager

DECLARATION OF CONFORMITY

According to ISO/IEC Guide 22 and CEN/CENELEC EN 45014

CE

Manufacturer's Name: VXI Technology, Inc.
Manufacturer's Address: 2031 Main Street
Irvine, California 92614
USA

Declares, that the product

Product Name: Remote Channel Signal Conditioning Plug-on
Model Number: VT1539A
Product Options: *This declaration covers all options of the above product(s).*

Conforms with the following European Directives:

The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC (including 93/68/EEC) and carries the CE Marking accordingly.

Conforms with the following product standards:

EMC	**Standard**	**Limit**
	IEC 61326-1:1997+A1:1998 / EN 61326-1:1997+A1:1998	
	Canada ICES-001:1998 *Australia/New Zealand AS/NZS/2064.1* *Russia GOST 23450-79* *Czech Republic CSN EN 55011* *Hungary MSZ EN 55011*	
	The product was tested in a typical configuration with VXI Technology test systems.	
Safety	*IEC 61010-1:1990+A1:1992+A2:1995 / EN 61010-1:1993+A2:1995* *Canada: CSA C22.2 No. 1010.1:1992* *UL 3111-1: 1994*	

30 July 2001
Date

Steve Mauga
Quality Assurance Manager

Notes:

Contents

VT1422A Remote Channel Multi-function DAC Module

Support Resources

Support resources for this product are available on the Internet and at VXI Technology customer support centers.

VXI Technology
World Headquarters

VXI Technology, Inc.
2031 Main Street
Irvine, CA 92614-6509

Phone: (949) 955-1894
Fax: (949) 955-3041

VXI Technology
Cleveland Instrument Division

VXI Technology, Inc.
7525 Granger Road, Unit 7
Valley View, OH 44125

Phone: (216) 447-8950
Fax: (216) 447-8951

VXI Technology
Lake Stevens Instrument Division

VXI Technology, Inc.
1924 - 203 Bickford
Snohomish, WA 98290

Phone: (425) 212-2285
Fax: (425) 212-2289

Technical Support

Phone: (949) 955-1894
Fax: (949) 955-3041
E-mail: support@vxitech.com

Visit http://vxitech.com for worldwide support sites and service plan information.

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, the VT1422A won't have to be removed from its mainframe later. Chapter contents include:

Configuring the VT1422A

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

For most applications, **only the Logical Address switch needs to be changed** prior to installation. The other settings can be used as delivered.

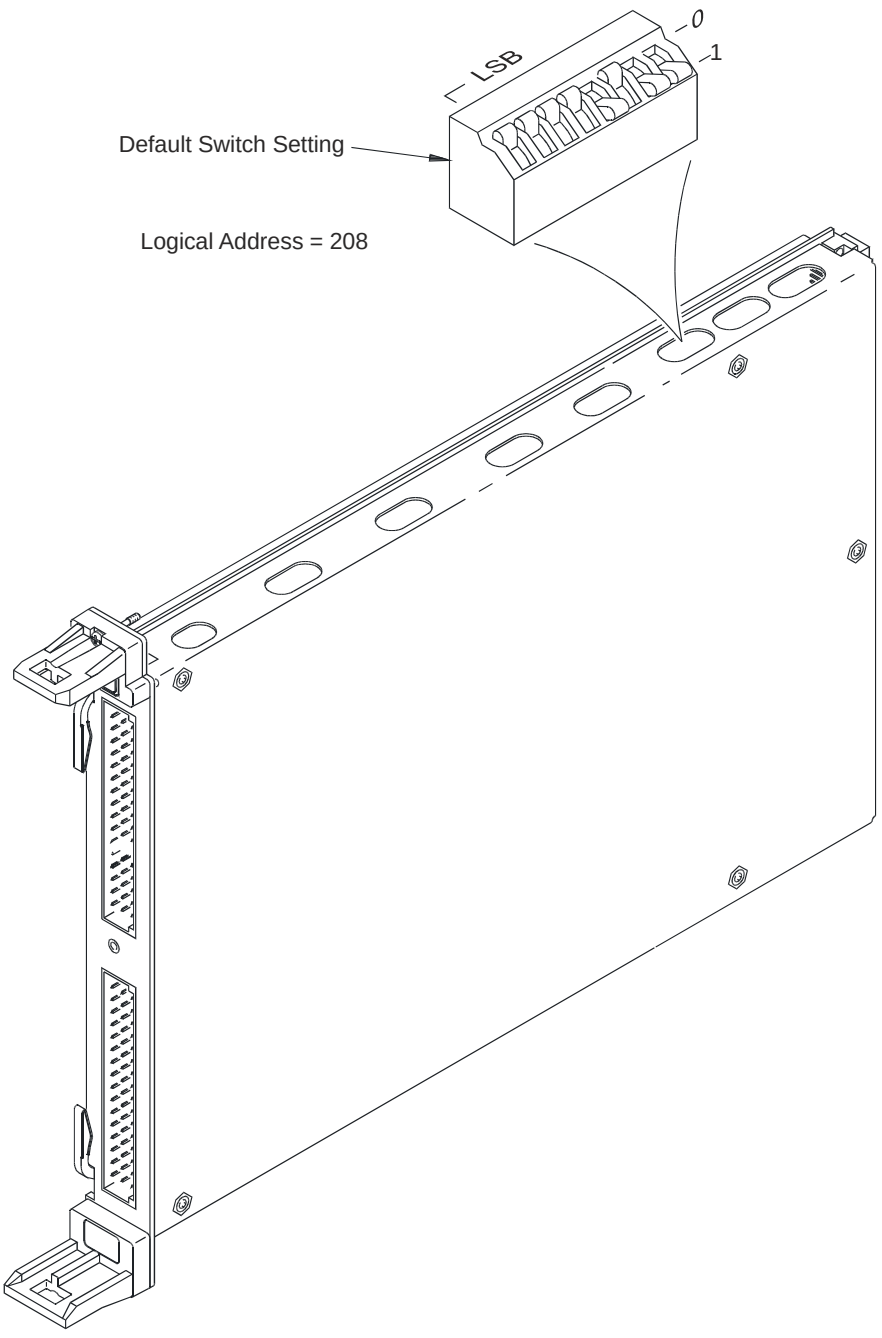
Switch/Jumper	Setting
Logical Address Switch	208
Input Protect Jumper	Protected
Flash Memory Protect Jumper	PROG

Note Setting the VXIbus Interrupt Level: The VT1422A uses a default VXIbus interrupt level of 1. The default setting is made at power-on and after an *RST command. The interrupt level can be changed by executing the DIAGnostic:INTerrupt[:LINe] command in the 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 VT1422A in a single VXIbus Mainframe, set each instrument to a different Logical Address.

Setting the Logical Address Switch

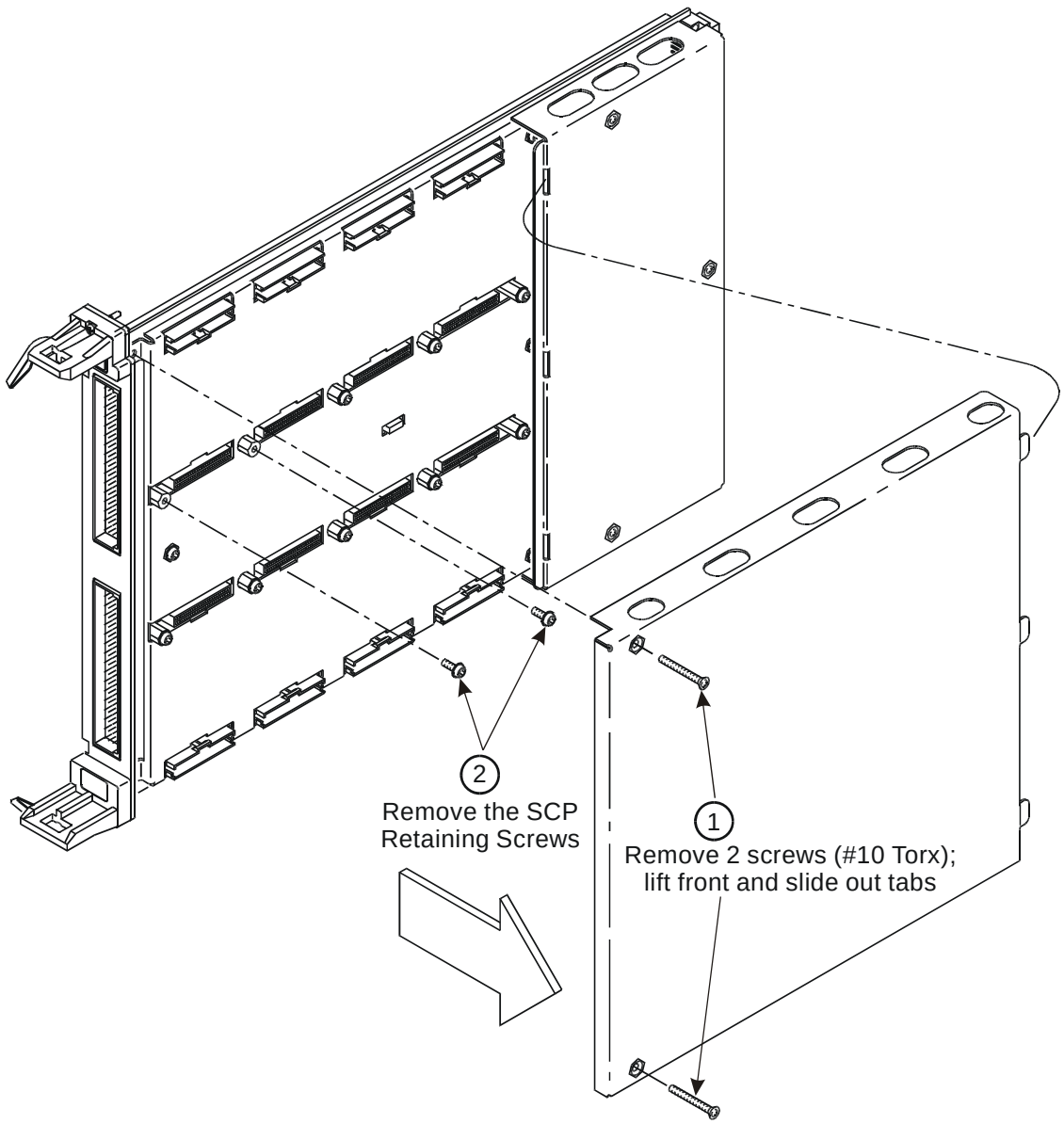


Installing Signal Conditioning Plug-Ons

The following illustrations show the steps used to install Signal Conditioning Plug-ons (SCPs). Before installing an SCPs, reading the "Separating Digital and Analog SCP Signals" in Appendix E page 483 is recommended.

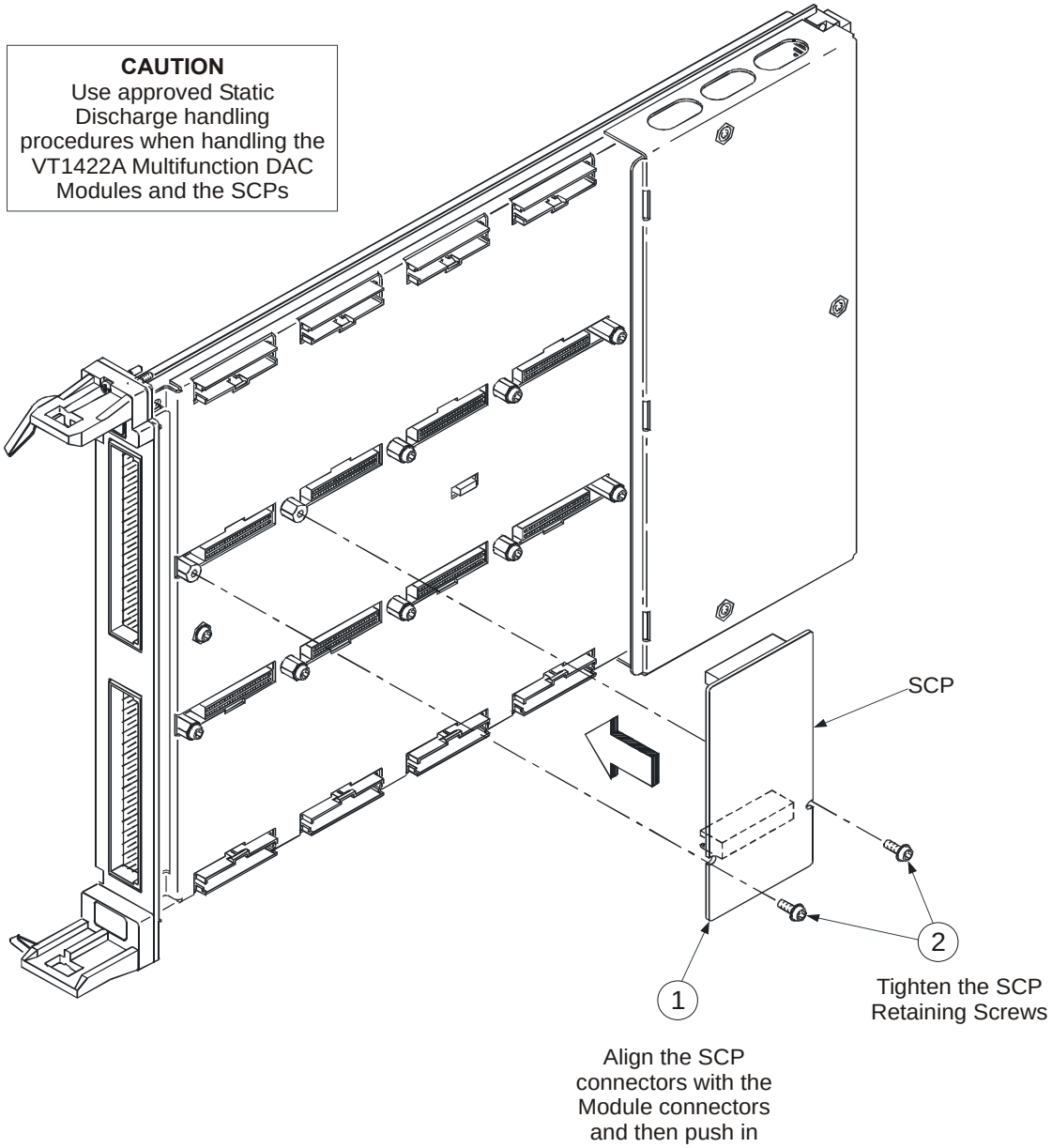
Caution **Use approved Static Discharge Safe handling procedures anytime the covers are removed from the VT1422A or when handling the SCPs.**

Installing SCPs: Step 1, Removing the Cover VT1422A

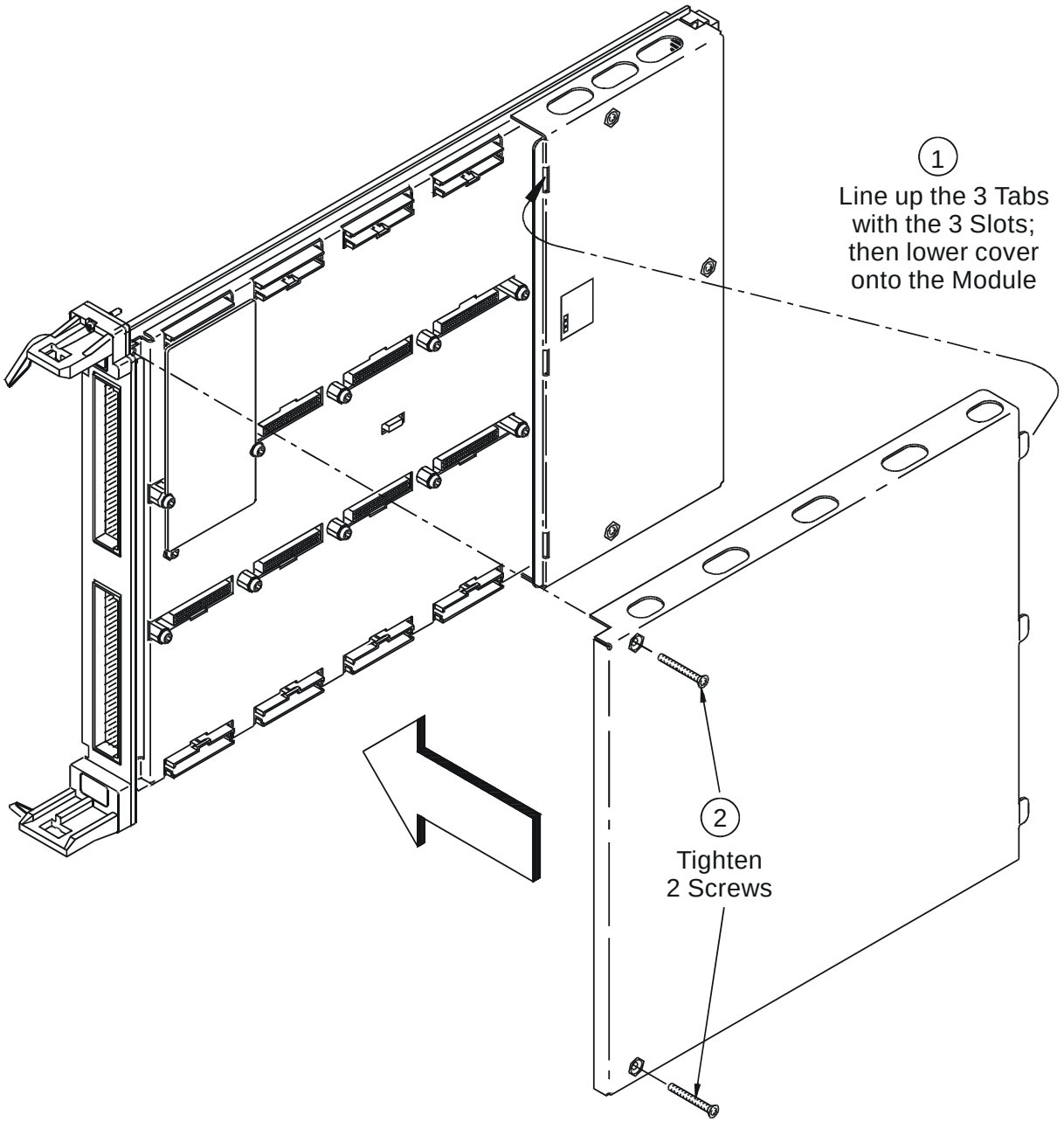


Installing SCPs: Step 2, Mounting an SCP

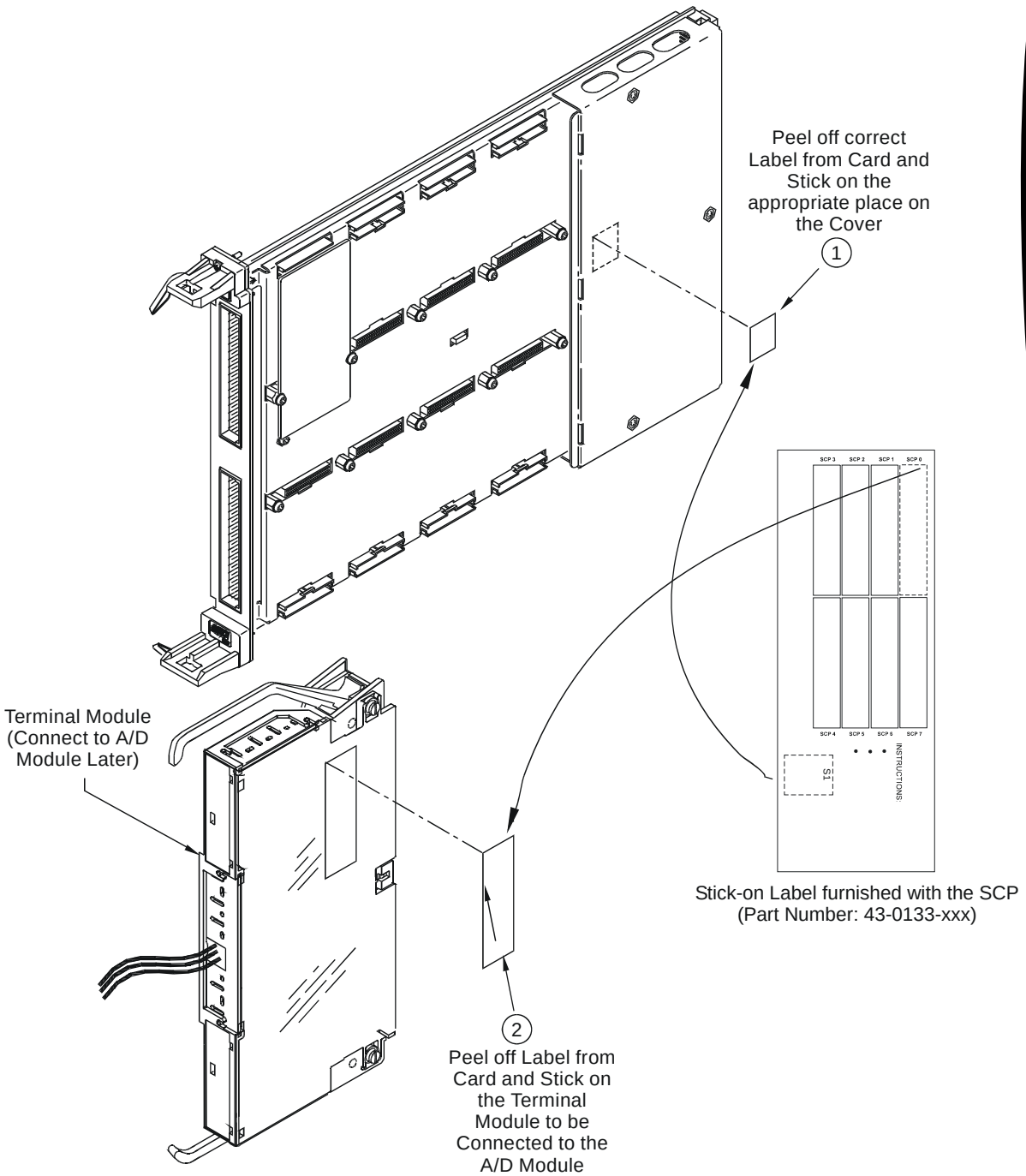
CAUTION
Use approved Static Discharge handling procedures when handling the VT1422A Multifunction DAC Modules and the SCPs



Installing SCPs: Step 3, Reinstalling the Cover VT1422A



Installing SCPs: Step 4, Labeling



Disabling the Input Protect Feature (Optional)

Disabling the Input Protect feature voids the VT1422A's warranty. The Input Protect feature allows the VT1422A 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 an application, the importance of completing a measurement run outweighs the added risk of damage to the VT1422A, the input protect feature may be disabled.**

Voids Warranty!

Disabling the Input Protection Feature voids the VT1422A'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. It is recommended that the jumper be left in this position so that all of the calibration commands can function. Changing the jumper to the protect position prevents the following from being executed:

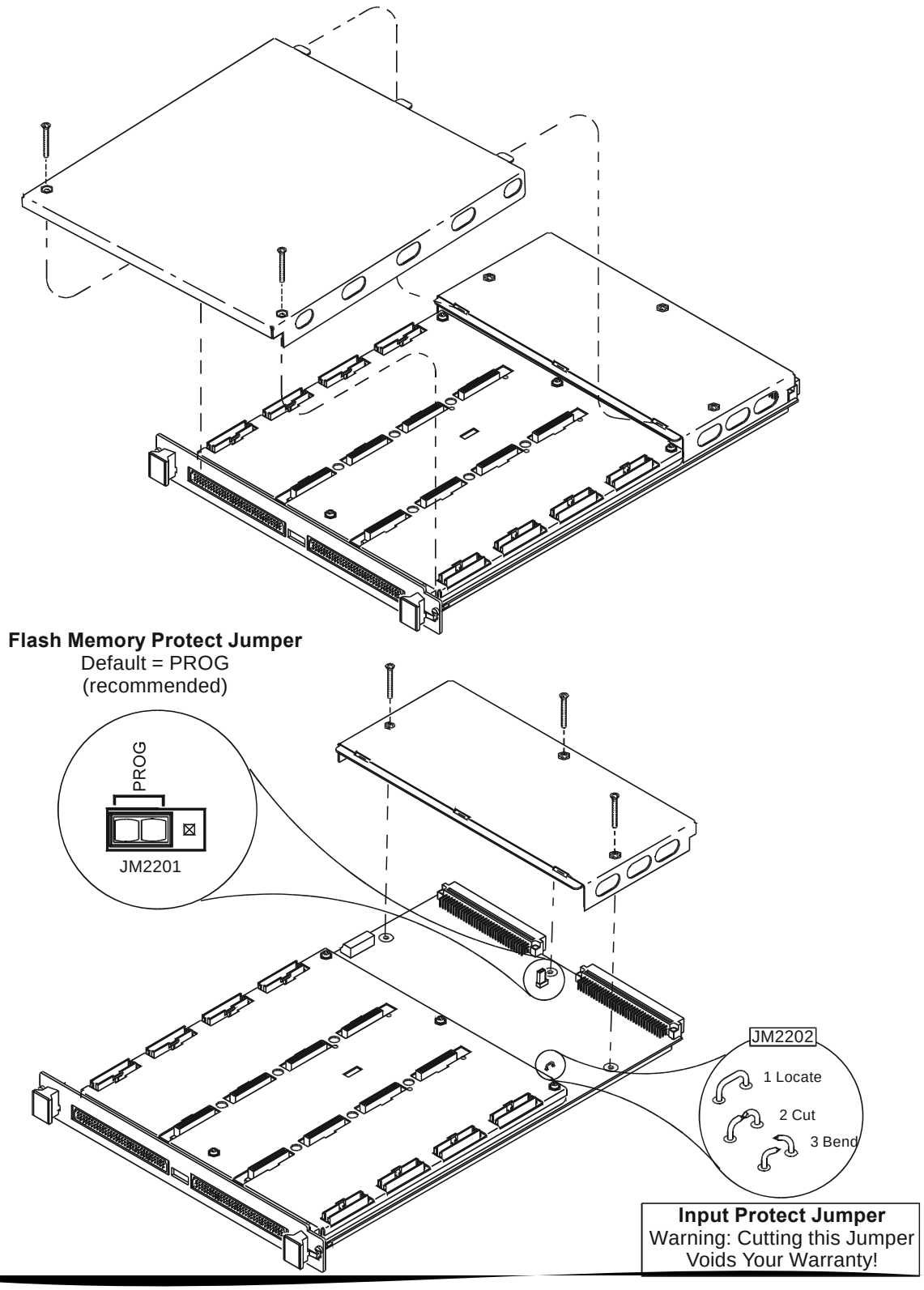
- The SCPI calibration command CAL:STORE ADC | TARE
- The register-based calibration commands STORECAL and STORETAR
- Any application that installs firmware-updates or makes any other modification to Flash Memory through the A24 window.

With the jumper in the “PROG” position, one or more VT1422As can be completely calibrated without removing them from the application system. A VT1422A calibrated in its working environment will, in general, be better calibrated than if it were calibrated separate from its application system.

The multimeter used during the periodic calibration cycle should be considered the calibration transfer standard. Allow Calibration Organization control unauthorized access to the calibration constants.

If access must be limited to the VT1422A's calibration constants, place JM2201 in the protected position and cover the shield retaining screws with calibration stickers. See the following illustration for location of JM2201.

Accessing and Locating JM2201 and JM2202 VT1422A



Installing the Module

Installation of the VT1422A VXI module is covered in the 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 VXI*plug&play* Drivers & Product Manuals CD that comes with the VT1422A. There is a VXI*plug&play* driver which includes a front panel program and help file. In addition, there is also a downloadable driver for the Agilent/HP E1406A Command Module. View the readme.txt file provided with the VXI*plug&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 VXI*plug&play* Drivers & Product Manuals CD supplied with the VT1422A. 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 VT1422A 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 VT1422A 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 139. for an example. For VXI*plug&play* users, there is an “Example VXIplug&play Driver Function Sequence” on page 140.

Typical Example program

The Verify program (file name *verif.cpp*) is printed below to show a typical VXI*plug&play* program for the VT1422A.

Verifying a Successful Configuration

An example C program source is shown on the following pages. This program is included on the VXI Technology VXI*plug&play* Drivers & Product Manuals CD that comes with the VT1422A (file name *verif.cpp*). The program uses the *IDN? query command to verify the VT1422A 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 the programs created, particularly the initial trial programs, so that instant feedback can be attained while learning about the VT1422A. 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 verifies your instrument installation by reading the
   instrument IDs and then querying for and printing the SCP types found.

   Use the "Copy Button" in the Help File's "Example" window to place this code
   into the clipboard, then paste this code text into your development tool's
    editor window. Similarly, "Copy" the actual example code from the help file's
   "Example" window and paste it into the location provided below.


   link with the hpe1422_32.lib - library file

 *****************************************************************************
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <hpe1422.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];


 /* 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 */

 /* 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(addr, 5000);
 check(addr, errStatus);

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

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

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

 /* enables automatic error checking after each driver call */
 errStatus = hpe1422_errorQueryDetect( addr, VI_TRUE);

}

/*****************************************************************************/
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)
  {
     /* read instrument error until error queue is empty*/
      do
      {
        hpe1422_error_query( addr, &err_code, err_message);
        if(err_code != 0) printf("Instrument Error : %ld, %s\n", err_code,
err_message);
      }
      while(err_code != 0);
  }
  else
  {
                        /* query the instrument */
   hpe1422_error_message( addr, errStatus, err_message);
                        /* display the error */
   printf("Driver Error : %ld, %s\n", errStatus, err_message);
  }
}

  return;
}
```

Chapter 2
Field Wiring

About This Chapter

This chapter shows how to plan and connect field wiring to the VT1422A's Terminal Module. The chapter explains proper connection of analog signals to the VT1422A, 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 the Wiring Layout

The first point to understand is that the VT1422A makes no assumptions about the relationship between Signal Conditioning Plug-on (SCP) function and the position in the VT1422A that it can occupy. Any type of SCP can be placed into any SCP position. There are, however, some factors which should be considered when planning what mix of SCPs should be installed in each of the VT1422As. The following discussions will help explain and clarify these factors.

SCP Positions and Channel Numbers

The VT1422A 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.

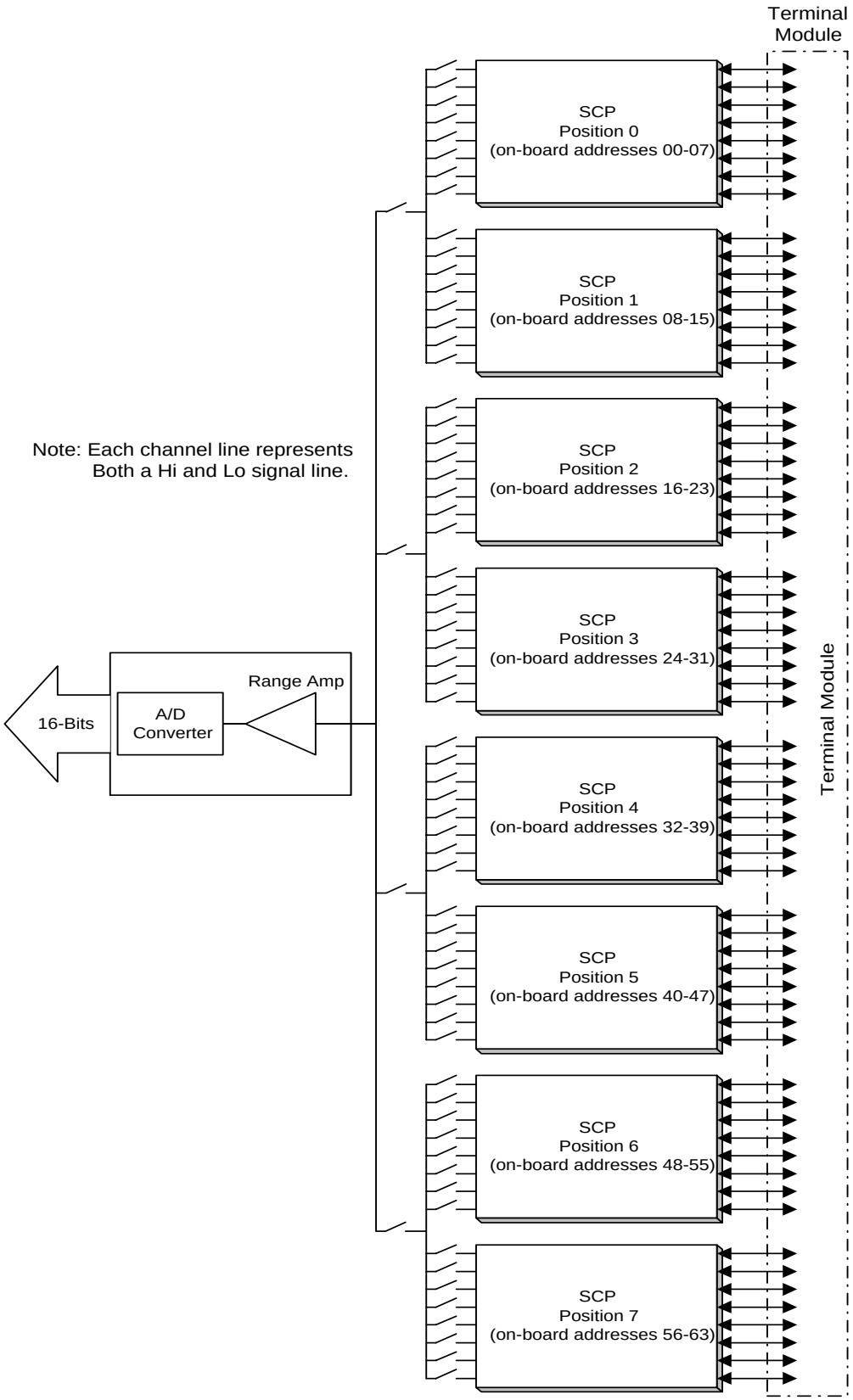


Figure 2-1. Channel Numbers at SCP Positions

Sense SCPs and Output SCPs

Some SCPs provide input signal conditioning (sense SCPs such as filters and amplifiers) while others provide stimulus to the 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 VT1422As. 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 the measurement circuit, reduce the number of external sense channels available to the VT1422A.

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

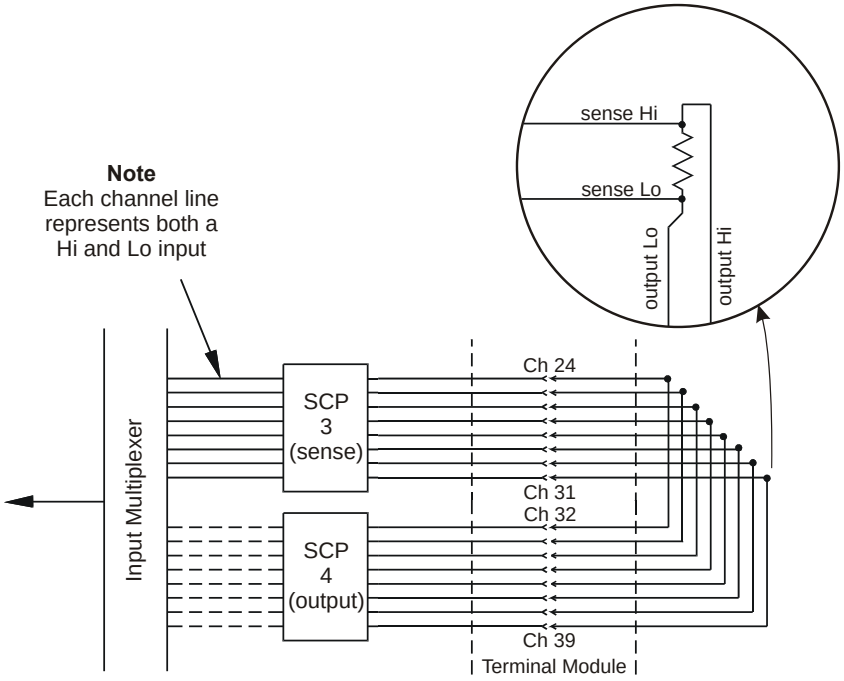


Figure 2-2. Pairing Output and Sense SCP Channels

Planning for Thermocouple Measurements

Using either the Screw Terminal or Spring Terminal Modules, thermocouples and the thermocouple reference temperature sensor can be wired to any of the VT1422A's channels. When the scan list is executed, one must ensure that the reference temperature sensor is specified in the channel sequence before any of the associated thermocouple channels.

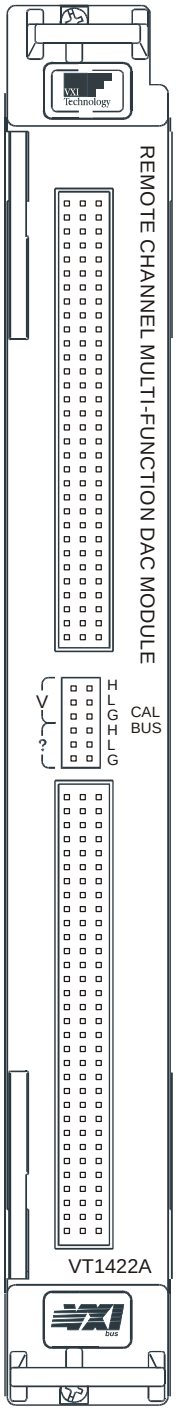
External wiring and connections to the VT1422A are made using Terminal Modules (see Figures 2-4 through 2-6).

Note The isothermal reference temperature measurement made by a VT1422A applies only to thermocouple measurements made by that instrument. In systems with multiple VT1422As, each instrument must make its own reference measurements. The reference measurement made by one VT1422A cannot be used to compensate thermocouple measurements made by another VT1422A.

Note To make good low-noise measurements, shielded wiring must be used from the device under test to the Terminal Module at the VT1422A. 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 483.

Faceplate Connector Pin-Signal Lists

Figure 2-3 shows the Faceplate Connector Pin Signal List for the VT1422A.



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. VT1422A Faceplate Connector Pin Signals

Optional Terminal and Connector Modules

The VT1422A is comprised of the main A/D module and optionally, a Connector or Terminal Module. The Option 001 Connector Module provides sixteen RJ-45 jacks to allow easy connection of the VT1422A to Remote Signal Conditioning Units (RSCUs) like the VT1529A/B 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 Remote Signal Conditioning Units.
- 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 VT1422A'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 28

Terminal Module Layout

Figures 2-4 through 2-6 show the layout and feature location of the Terminal Modules available for the VT1422A.

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 30 V ac rms or 60 V dc.**

The RJ-45 Connector Module

Figure 2-4 shows the VT1422A Option 001 RJ-45 Connector Module with connector pin numbering.

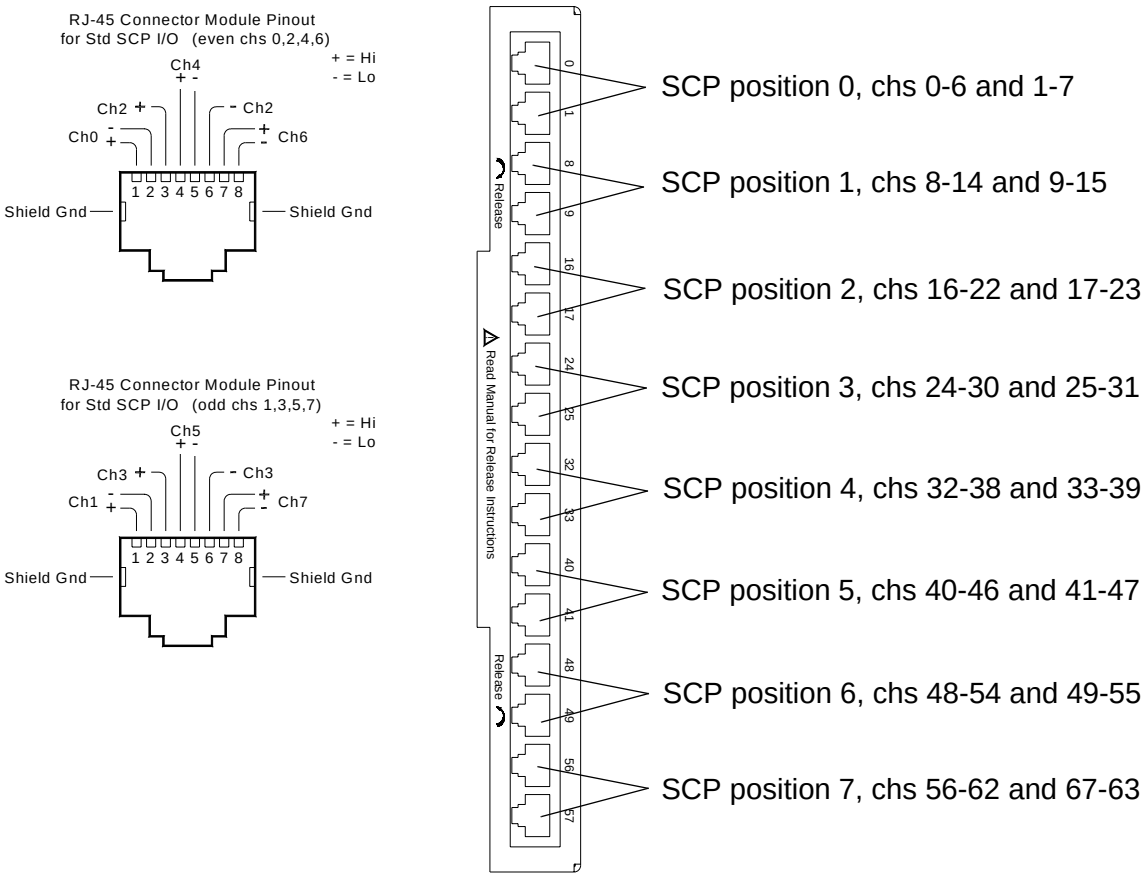


Figure 2-4. RJ-45 Connector Module and Pin-out

Spring Terminal Module Layout

Figure 2-5 shows the VT1422A 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 5 mm strip back of insulation to prevent the possibility of shorting to other wiring on adjacent terminals.

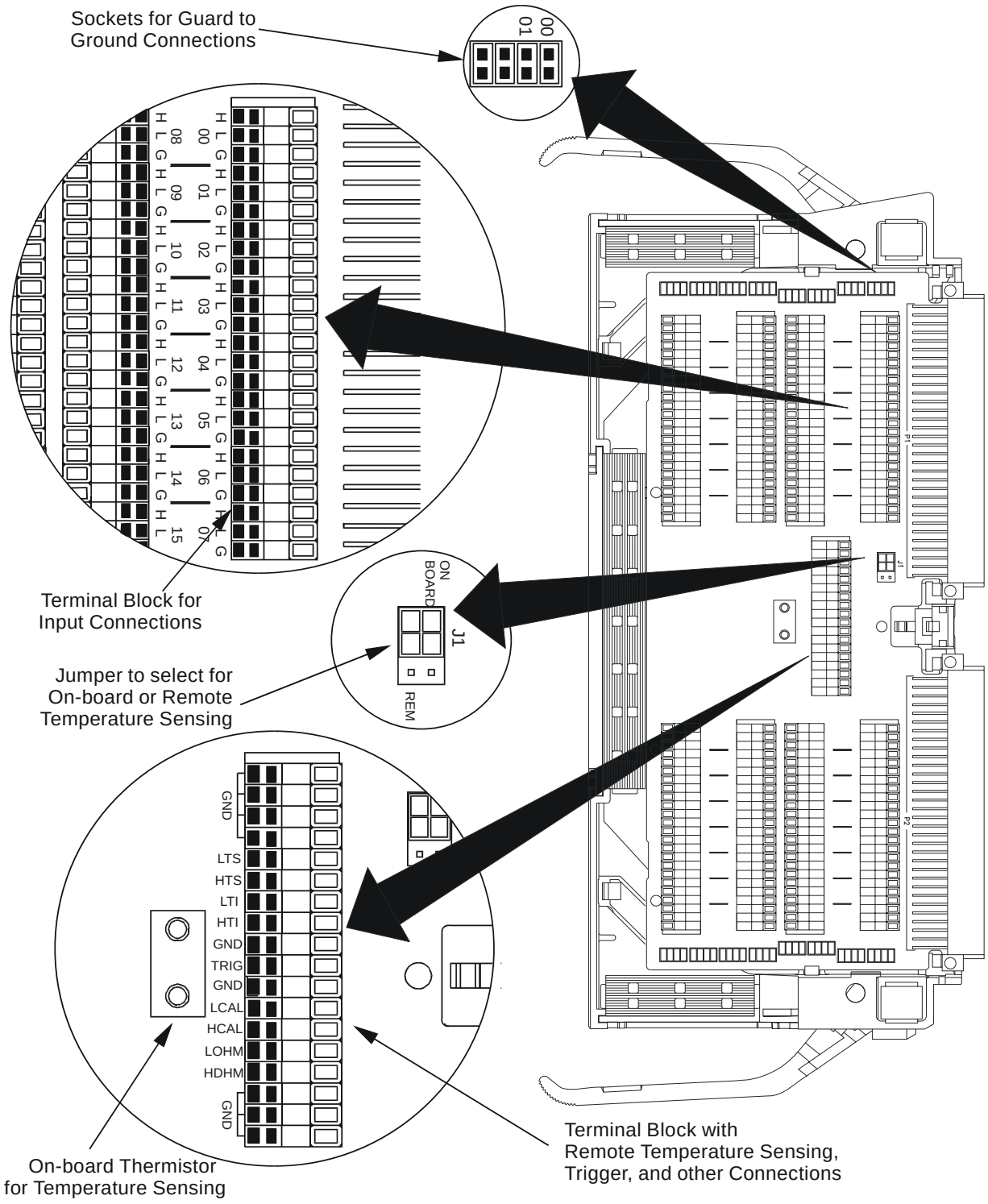


Figure 2-5. VT1422A Spring Terminal Module

Screw Terminal Module Layout

Figure 2-6 shows the VT1422A Option 011 Screw Terminal Module features and jumper locations.

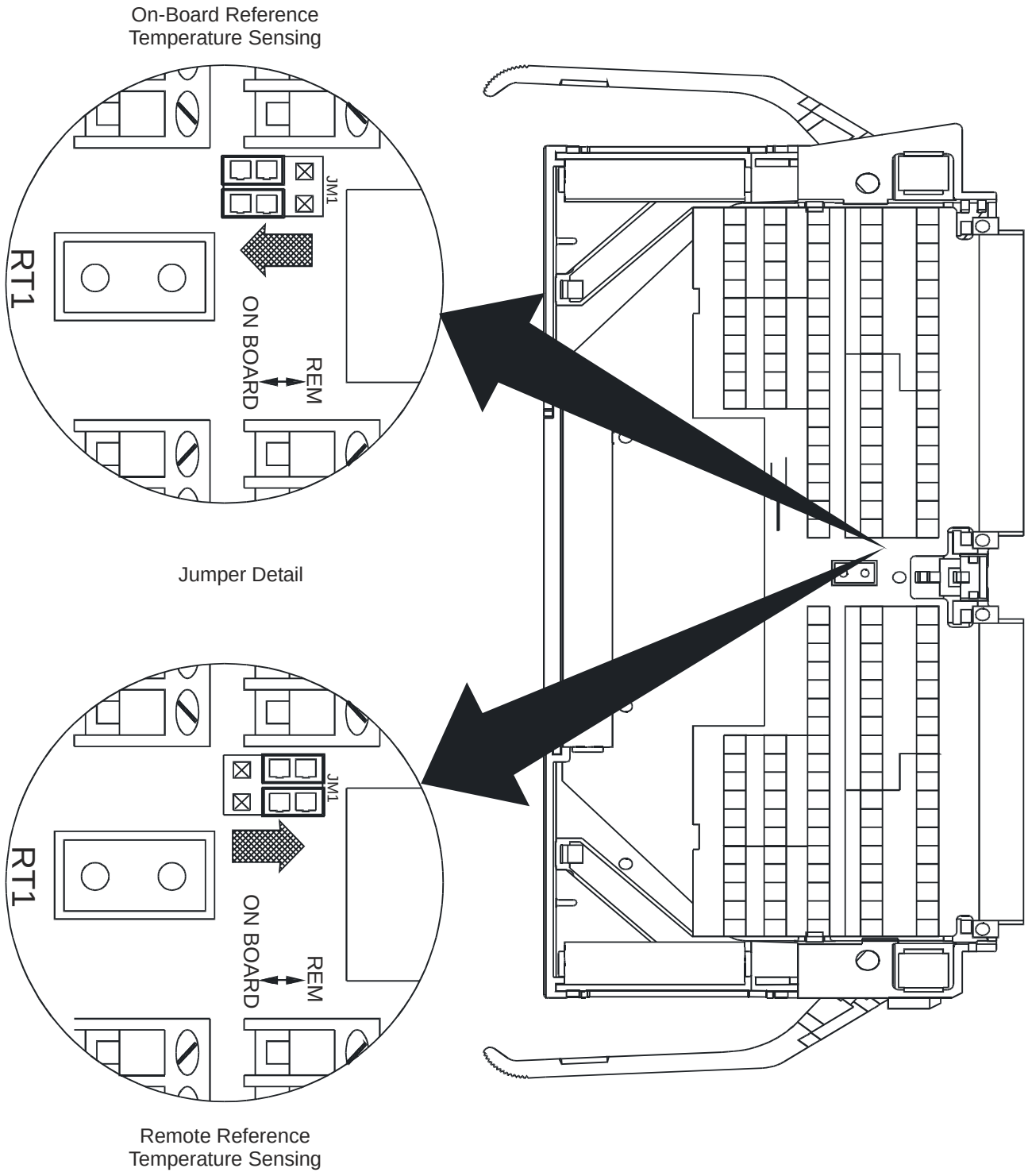


Figure 2-6. VT1422A Screw Terminal Module

Reference Temperature Sensing with the VT1422A

The Screw Terminal and Spring Terminal Modules provide 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 VT1422A'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 49 for location of J1.

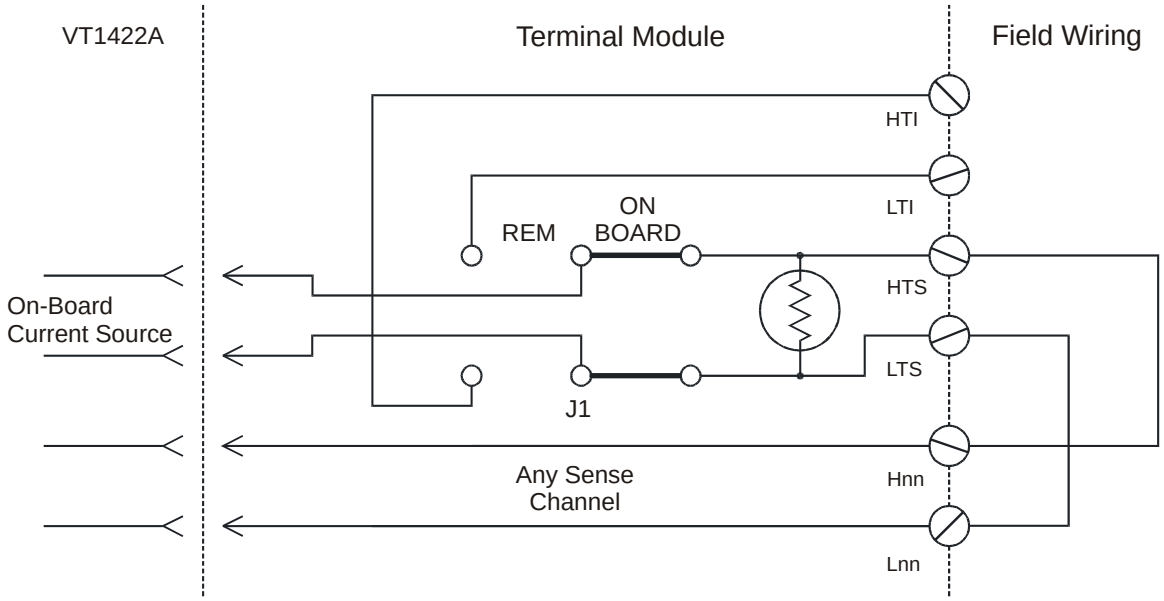


Figure 2-7. On-Board Thermistor Connection

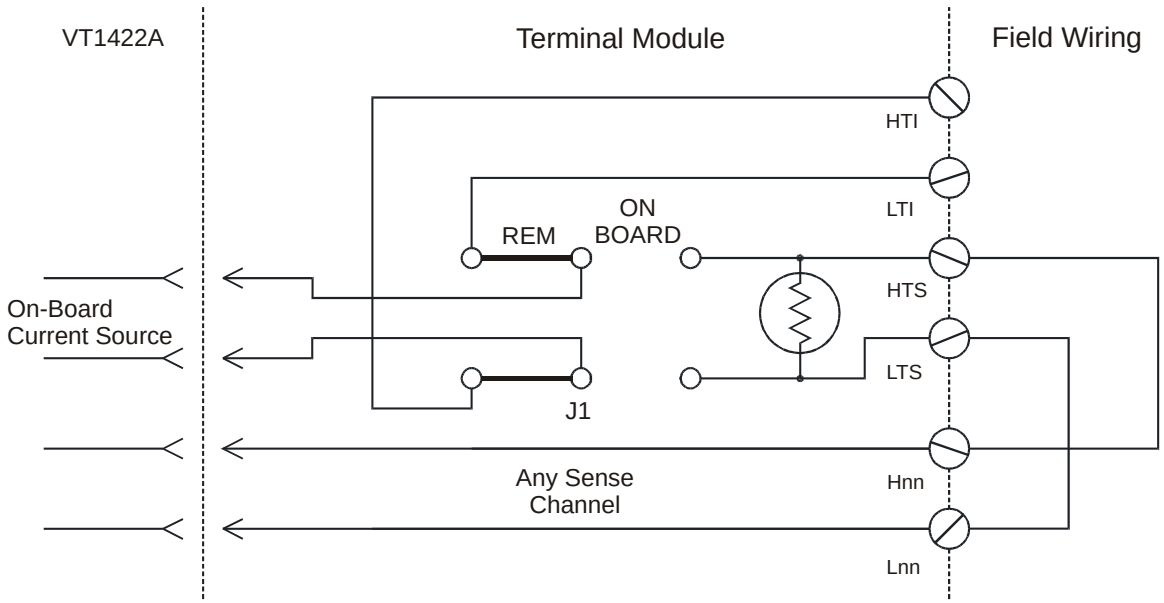


Figure 2-8. Remote Thermistor or RTD Connections

Terminal Module Considerations for TC Measurements

The isothermal characteristics of the VT1422A 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 VT1422A 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 VT1422A 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 VT1422A at a uniform temperature. If there is no front door, try opening the back door.
6. VXI Technology recommends that the cooling fan switch on the back of the of the CT-400 Mainframe be in the "High" position. The normal variable speed cooling fan control can make the internal VT1422A module temperature cycle up and down, which affects the amplifiers with these microvolt-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 (< 10 μs for the VT1422A). 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 E page 483 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 E.
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-to-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 50/60 Hz) 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).
5. 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.
6. Use good quality foil or braided shield signal cable.
7. Route signal leads as far as possible from the sources of greatest noise.
8. In general, don't connect Hi or Lo to Guard or Ground at the VT1422A.
9. It is best if there is a dc path somewhere in the system from Hi or Lo to Guard/Ground.
10. The impedance from Hi to Guard/Ground should be the same as from Lo to Guard/Ground (balanced).
11. Since each system is different, don't be afraid to experiment using the suggestions presented here until an acceptable noise level is found.

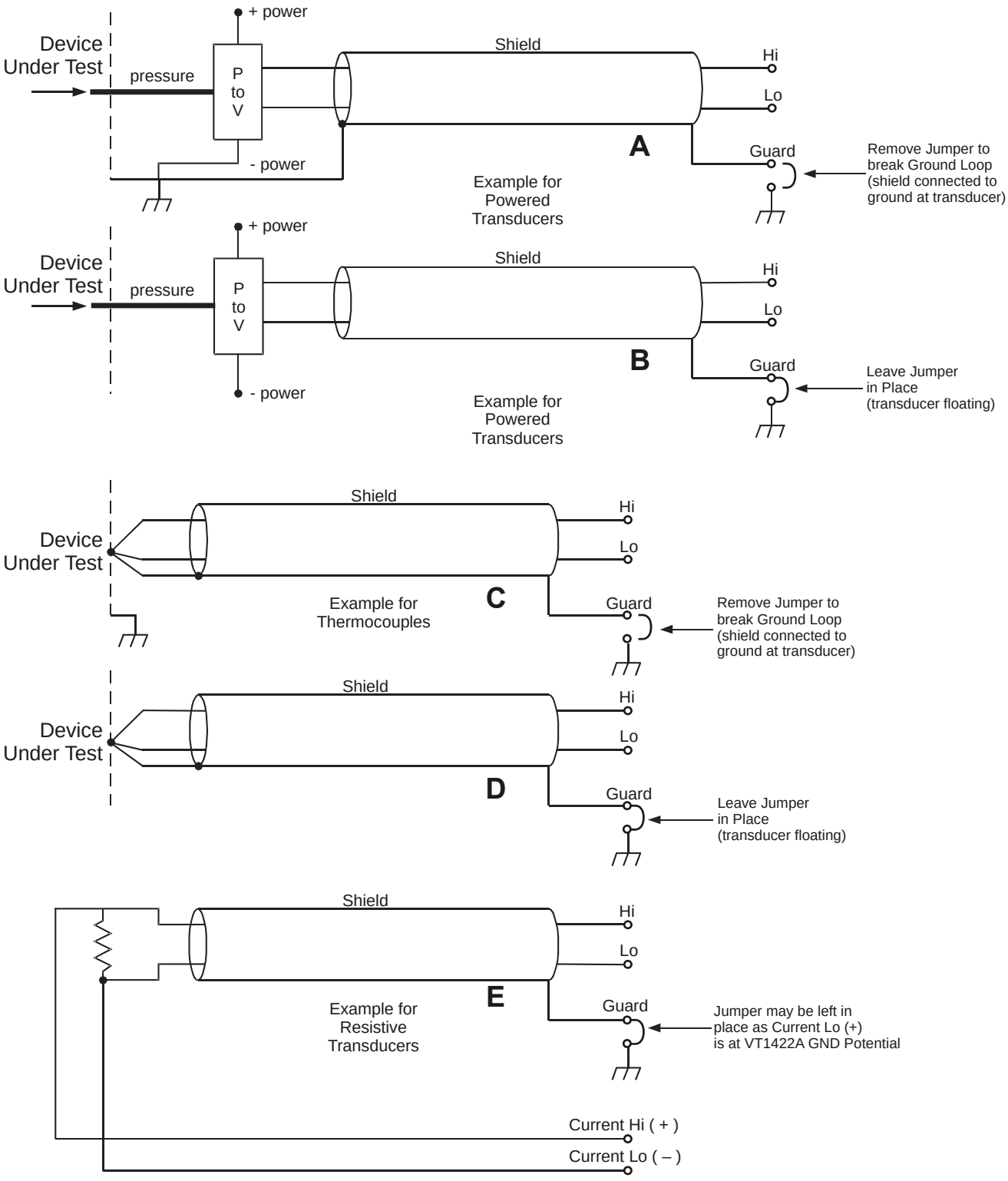


Figure 2-9. Preferred Signal Connections

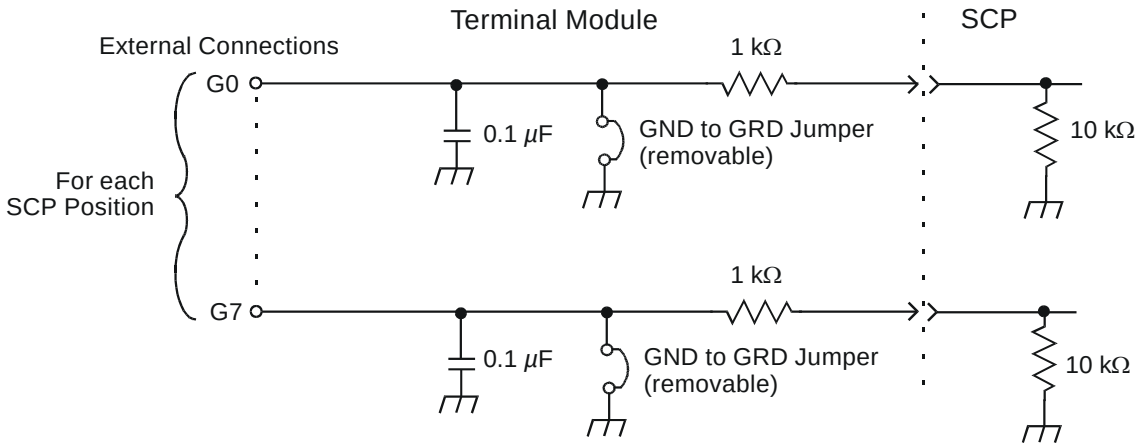


Figure 2-10. GRD/GND Circuitry on Terminal Module

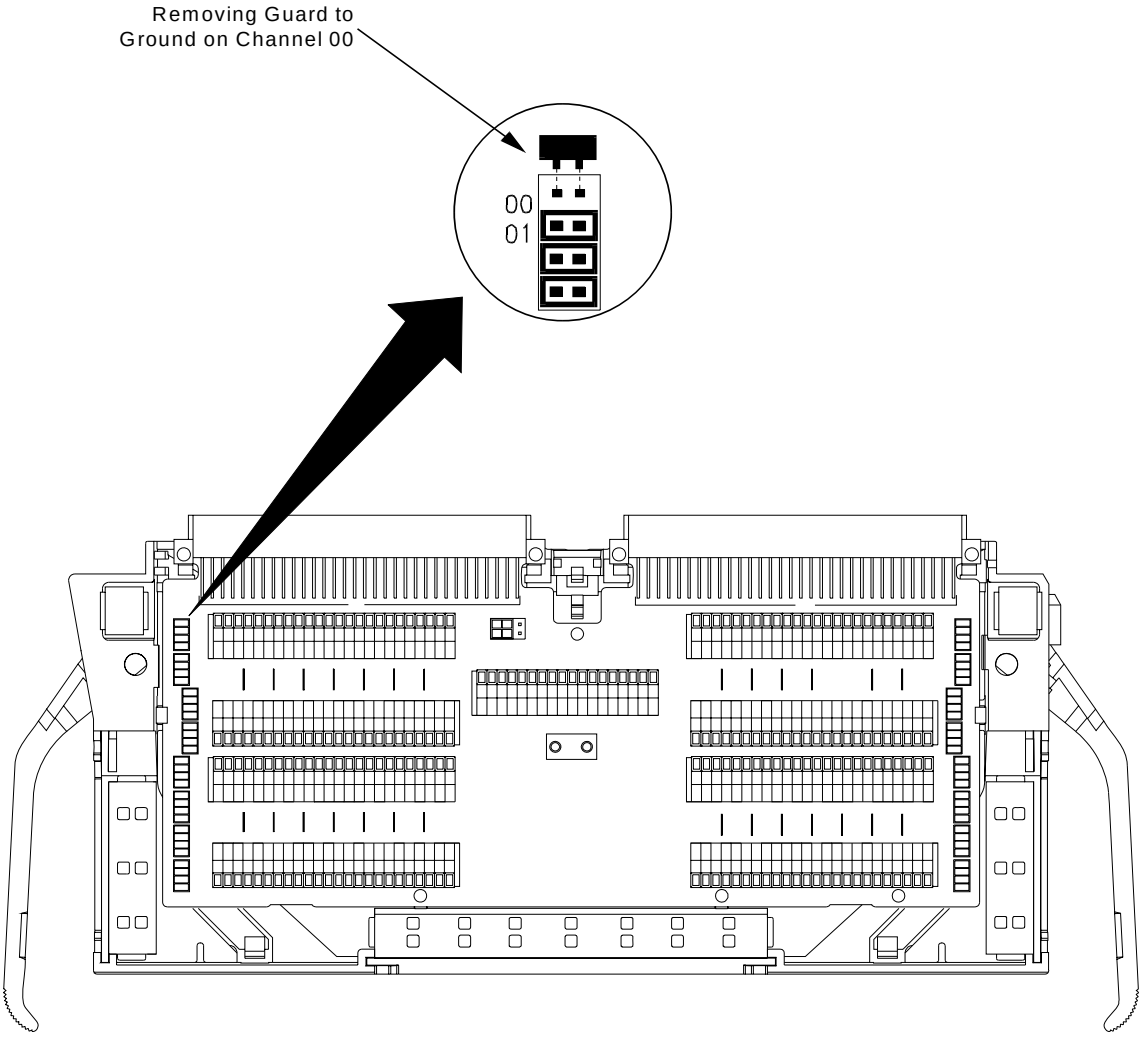


Figure 2-11. Grounding the Guard Terminals

Connecting the On-Board Thermistor

The following figures show how to use the VT1422A 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 VT1422A's Spring Terminal Module, Figure 2-6 shows the configuration for the Screw Terminal Module. See "Reference Temperature Sensing with the VT1422A" on page 44 for a schematic diagram of the reference connections.

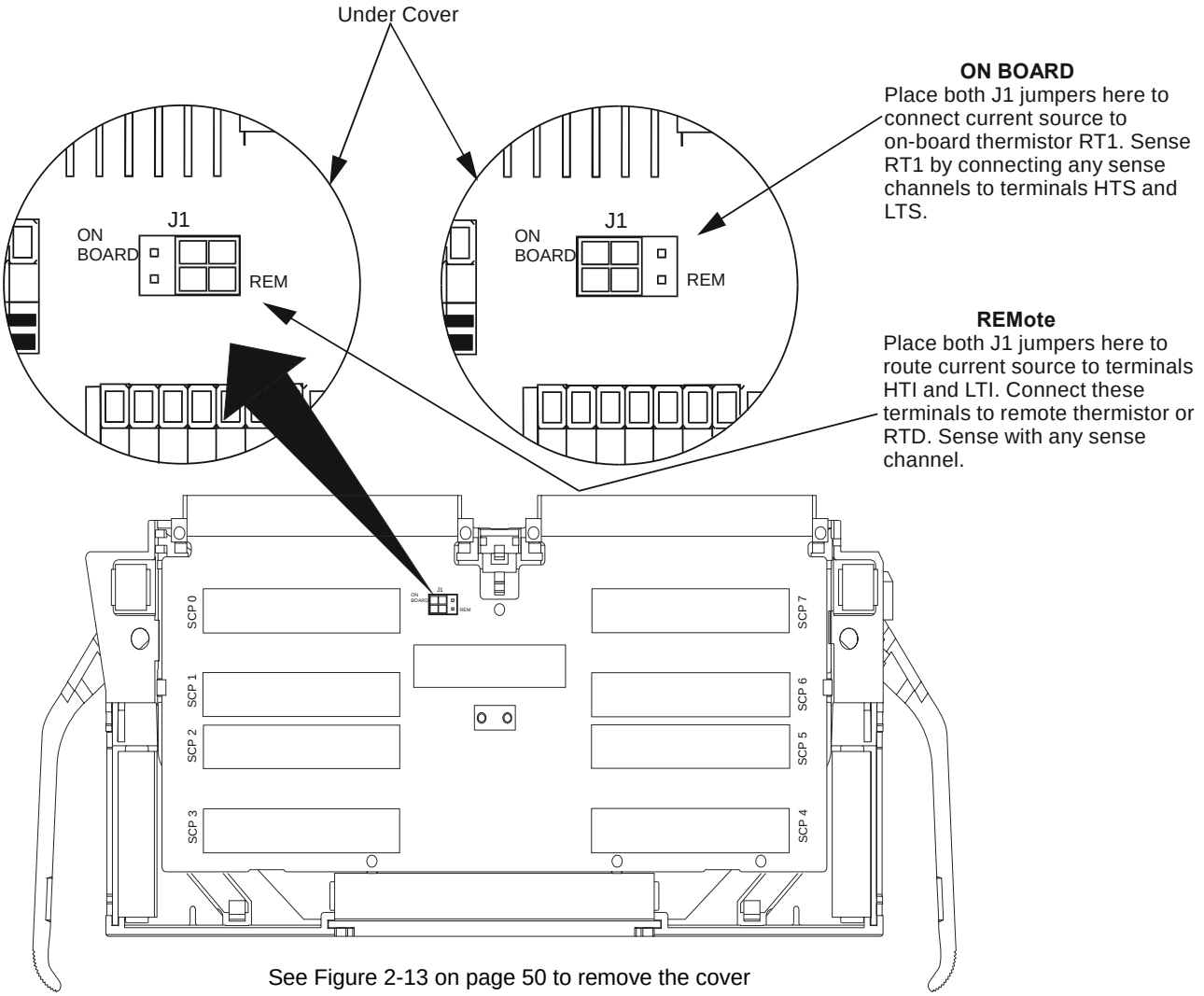


Figure 2-12. Temperature Sensing for the Terminal Module

Wiring and Attaching the Terminal Module

Figures 2-13 and 2-14 show how to open, wire and attach the terminal module to a VT1422A.

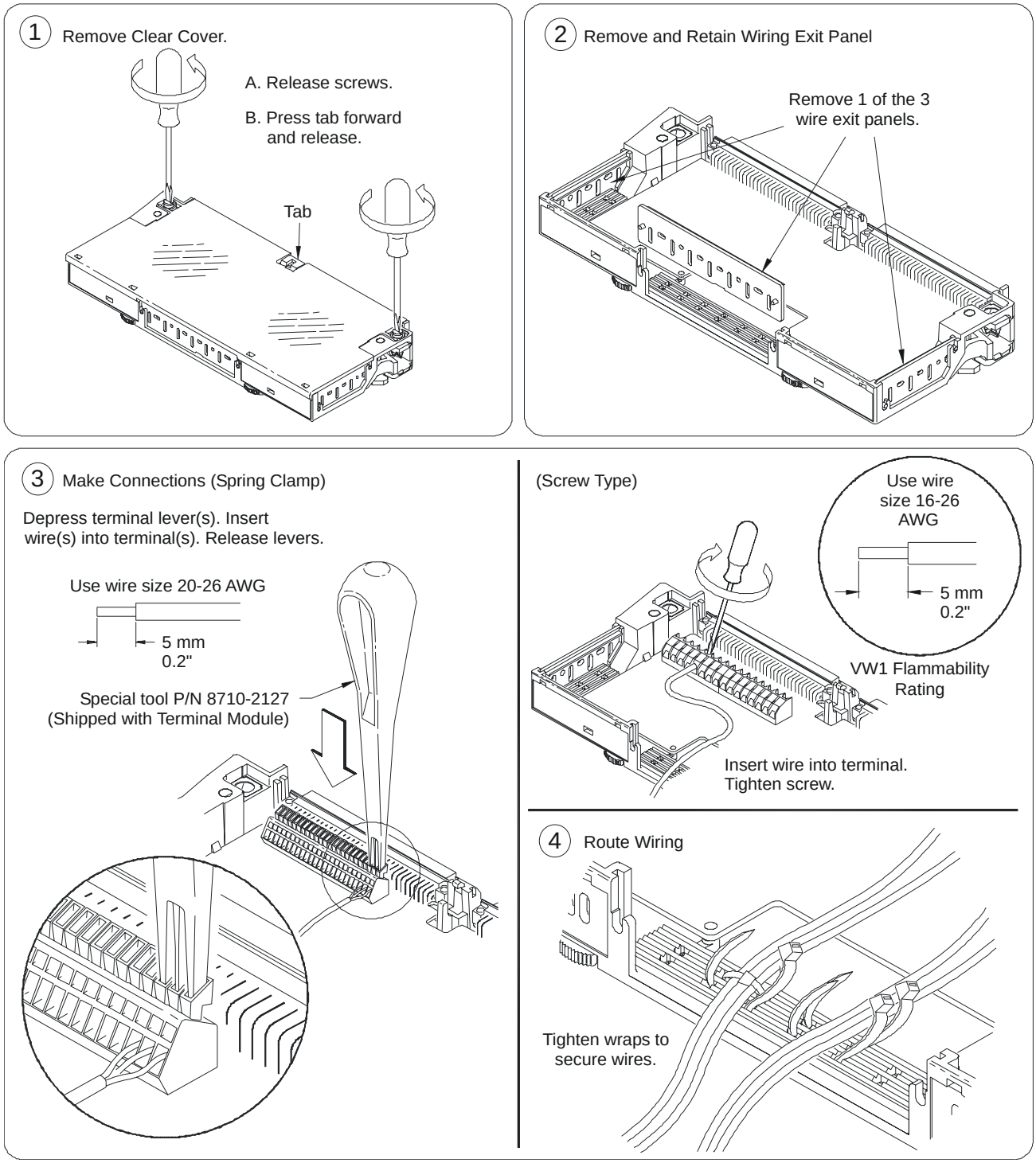


Figure 2-13. Opening and Wiring the VT1422A's Terminal Module

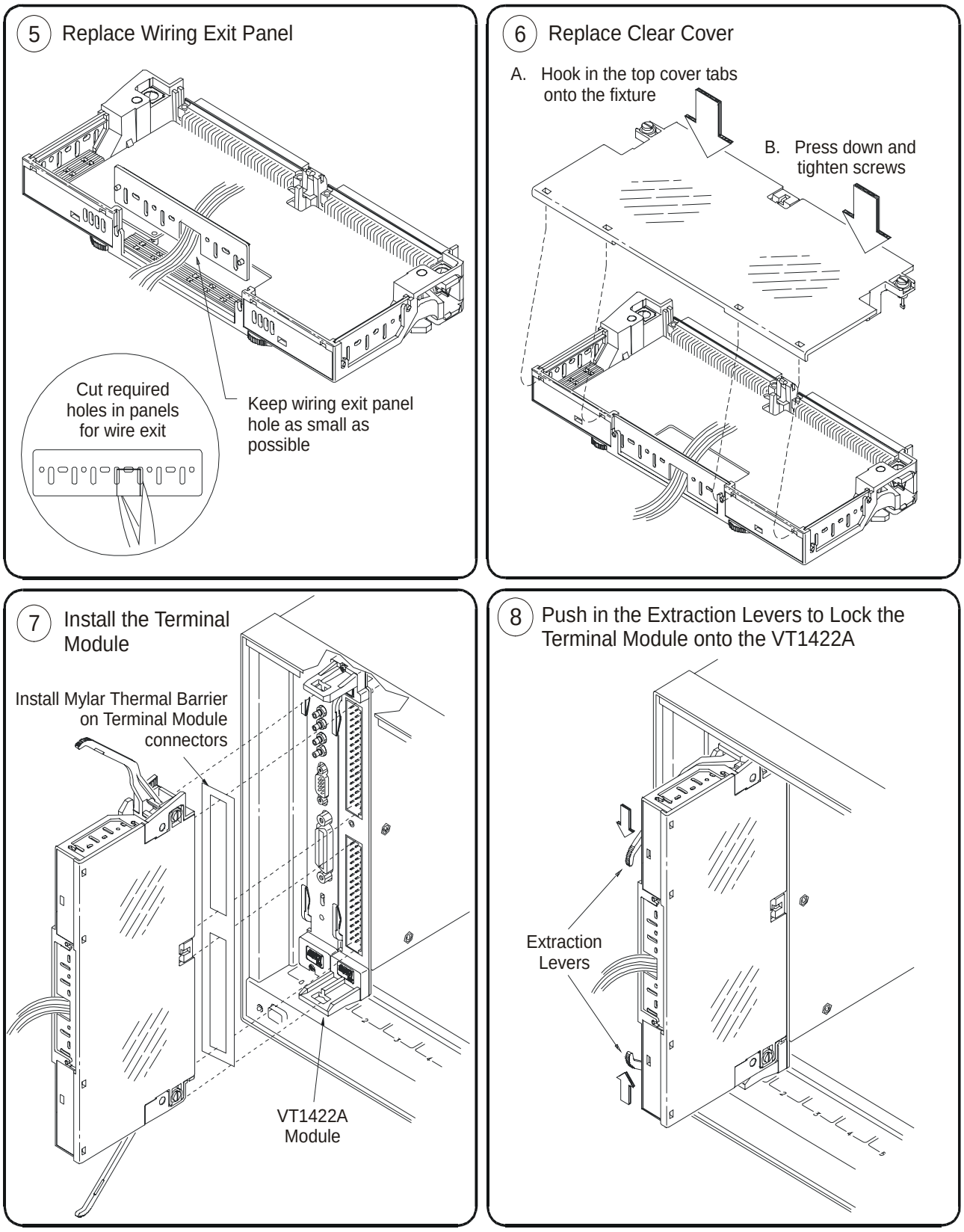


Figure 2-14. Closing and Attaching the VT1422A Terminal Module

Removing the VT1422A Terminal Modules

Figure 2-15 shows how to remove the Spring Terminal and Screw Terminal Modules from the VT1422A.

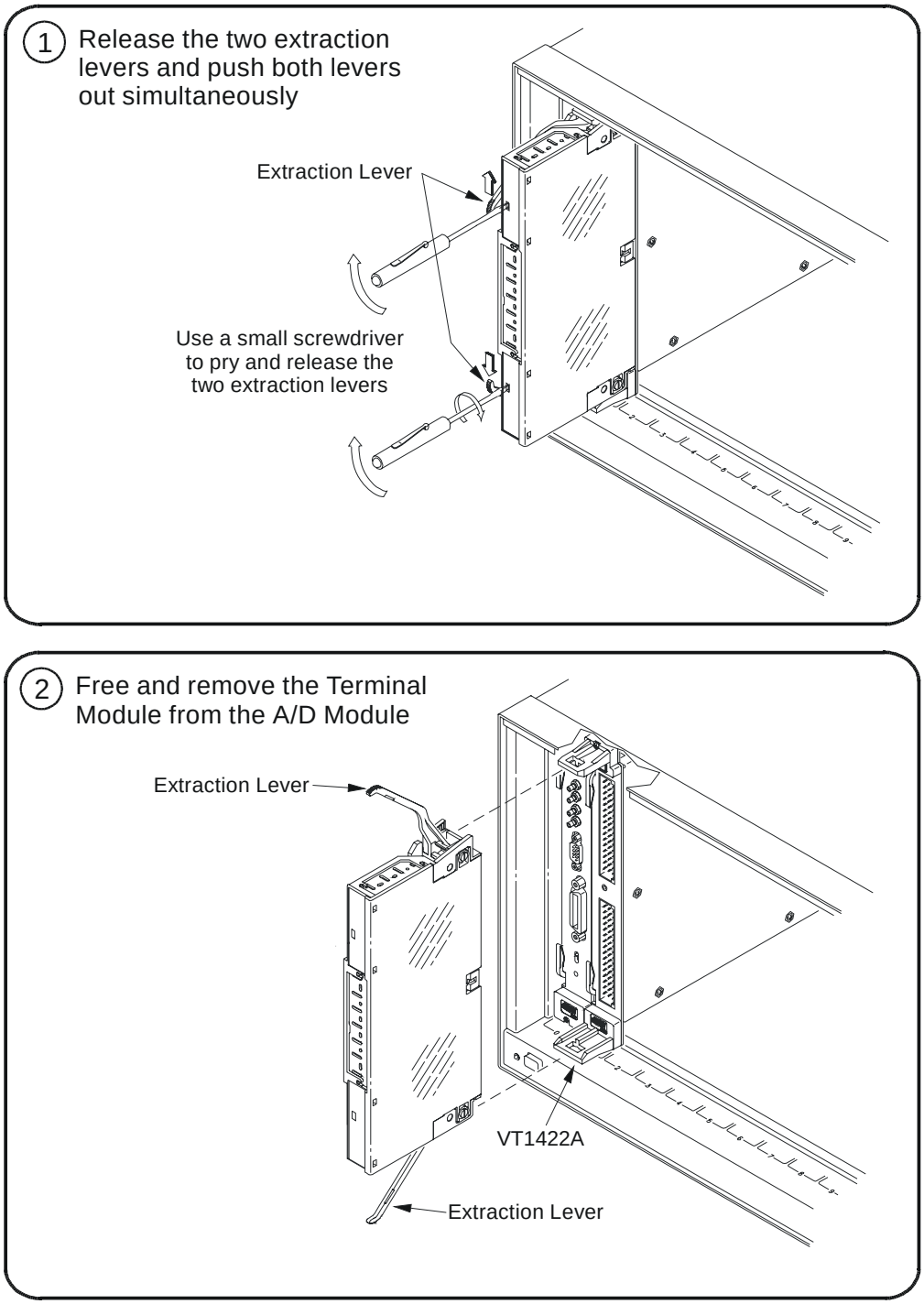


Figure 2-15. Removing the Screw and Spring Terminal Modules

Attaching and Removing the VT1422A RJ-45 Module

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

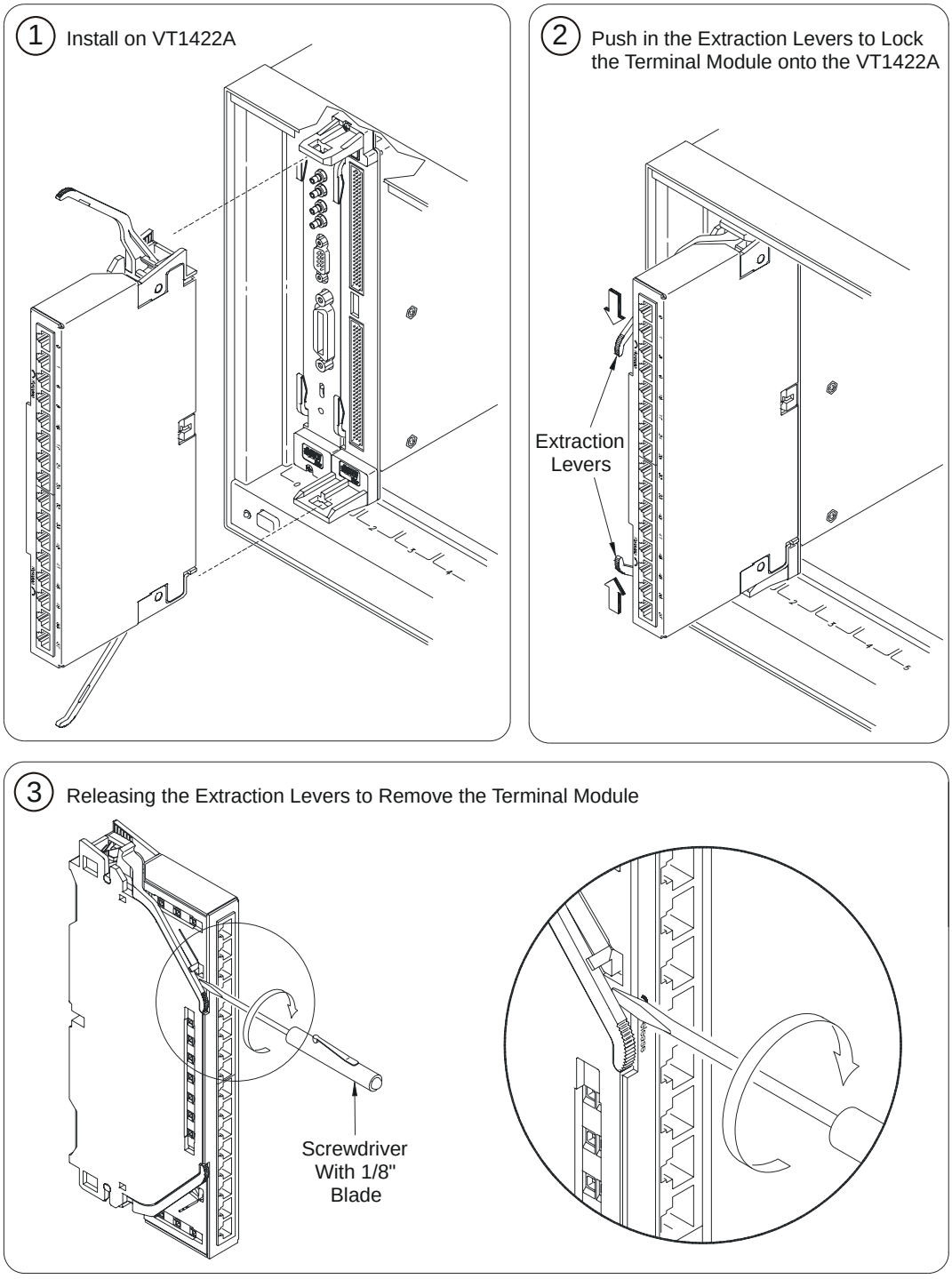


Figure 2-16. Removing the RJ-45 Terminal Module

Adding Components to the Terminal Module

The back of the terminal module PCB (printed circuit board) provides surface mount pads which can be used 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 terminal module PCB). Figure 2-18 shows some usage example schematics.

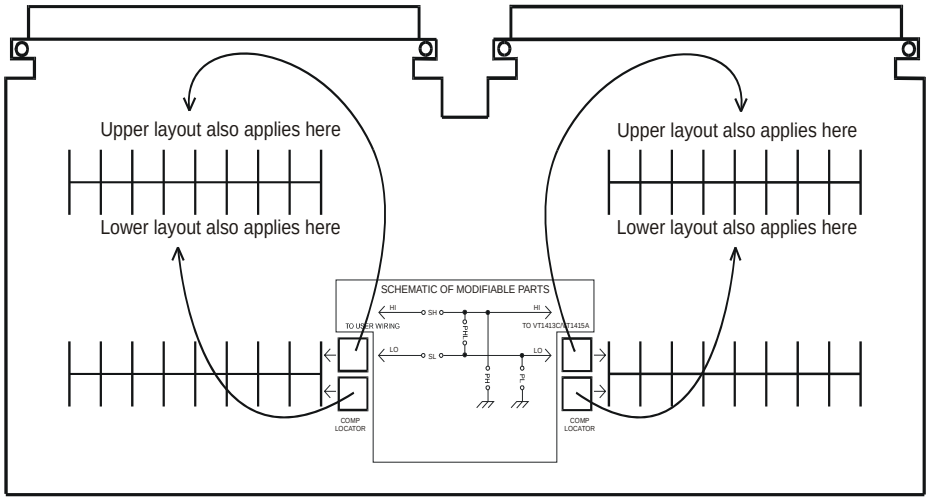
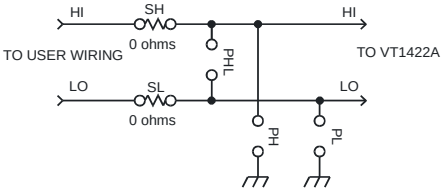
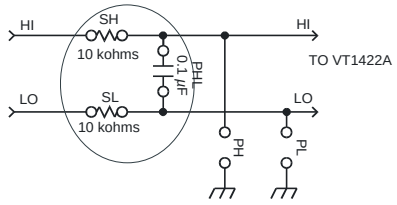


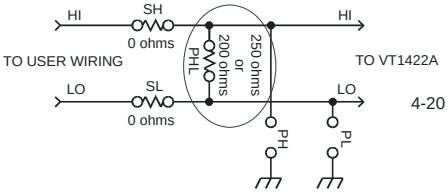
Figure 2-17. Additional Component Location



Default Circuit



Normal Mode Low-Pass Filter Circuit



4-20 mA NOTE: input must not exceed common mode limits (usually ±16 volts unless attenuated with a VT1513A SCP)

4 to 20 mA Sense
5 V full scale with 250 ohm (must use 16 volt range)
4 V full scale with 200 ohm (can use 4 volt range for better resolution)

Figure 2-18. Series & Parallel Component Examples

Spring and Screw Terminal Module Wiring Maps

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

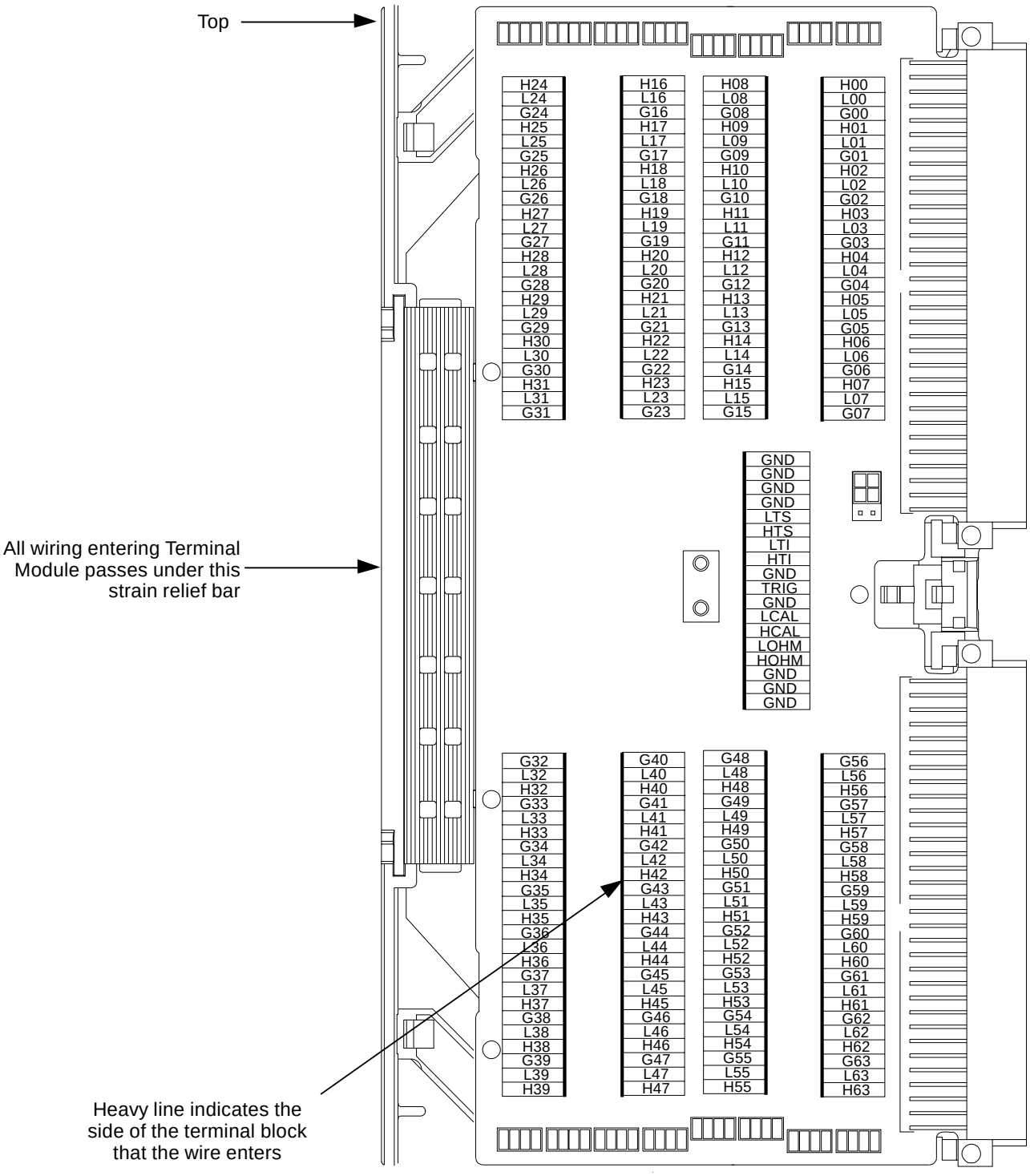


Figure 2-19. Spring Terminal Module Full-Size Wring Map

Figure 2-20 shows the Screw Terminal Module wiring map.

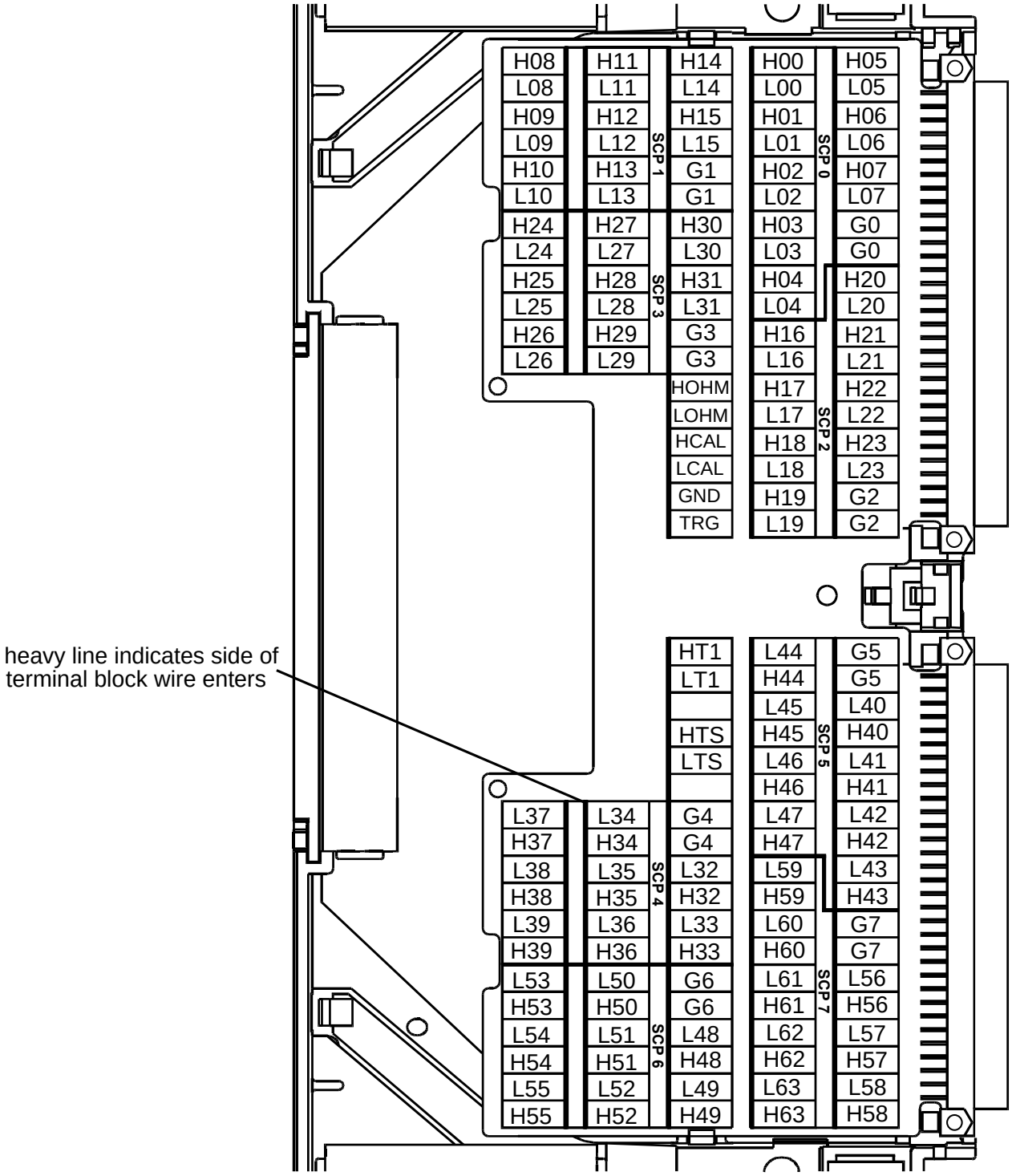


Figure 2-20. Screw Terminal Module Full-Size Wiring Map

Chapter 3
Programming the VT1422A & VT1529A/B for Remote Strain Measurement

About This Chapter

This chapter describes using the VT1422A in combination with the VT1539A Remote Channel SCP and VT1529A/B Remote Strain Conditioning Units to make large channel count strain measurements. The system is shown being used strictly in a data acquisition mode where, after configuration, it is driven by a user-defined channel list (the Scan List) and sends the measurements to the unit's FIFO buffer and Current Value Table (CVT) for transfer to the computer. Of course, control algorithms can also be created that execute concurrently with the Scan List driven data acquisition operation. Chapter 4 and Chapter 6 cover general data acquisition and control programming with algorithms. This chapter assumes that the user is an expert when it comes to making strain measurements, so this chapter simply illustrates how to make strain measurements with the VXI Technology Remote Strain Measuring System (VT1422A, VT1539A, and VT1529A/B). The chapter will cover:

Instrument Setup for Remote Strain Measurements

This section involves:

- Preparing the VT1422A for installation into a VXIbus Mainframe
- Preparing the VT1529A/B for use
- Connecting the VT1422A to VT1529A/B Remote Strain Completion units.
- Connecting Excitation power supplies to the VT1529A/B
- Connecting strain bridges to the VT1529A/B

Preparing the VT1422A for Installation

The VT1422A requires VT1539A SCPs to control Remote Signal Conditioning Units like the VT1529A/B Remote Strain Conditioning Unit. Chapter 1 “Getting Started” covers everything which is needs to be done before installing the VT1422A 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 getting into the specifics of configuring a Remote Strain Measuring System, it might be helpful to see the overall set up. Figure 3-1 shows the components and connections of a remote strain measuring system. The circled letters identify connections that will be referenced in later sections.

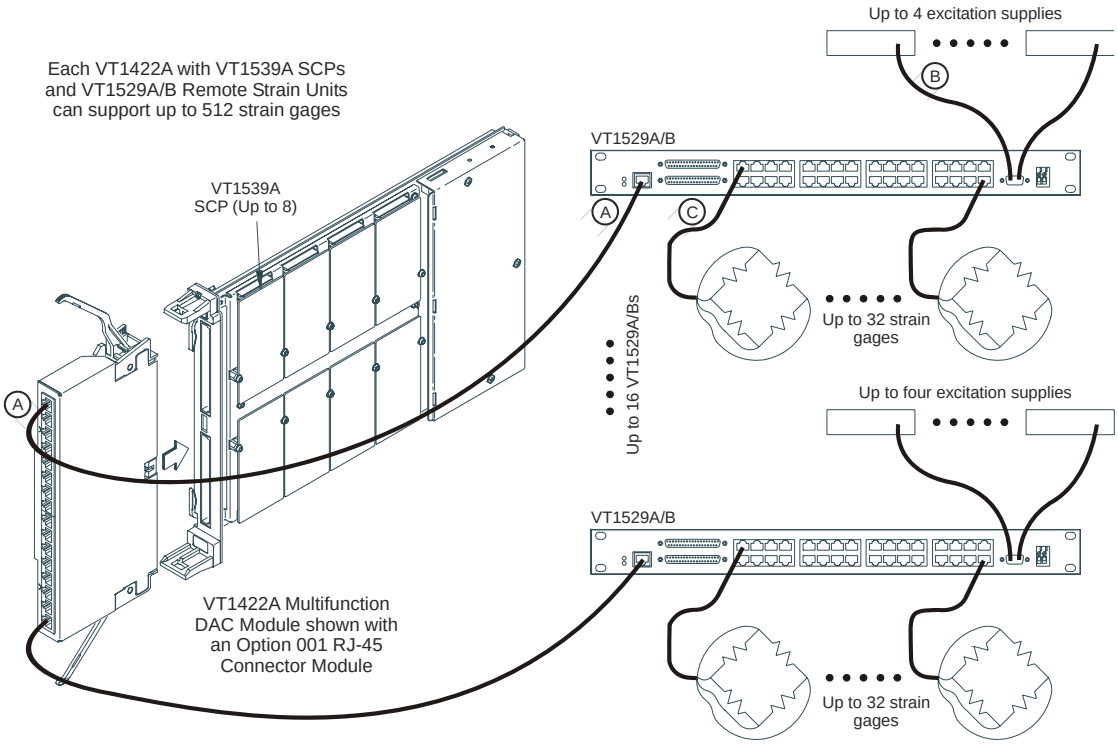


Figure 3-1. Components of the Remote Strain Measuring System

Preparing the VT1529A/B for Use

For most applications, the VT1529A/B 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 which may need to be considered is installation of the user supplied quarter-bridge completion resistors. If these are required by the application, see "Installing User Selected 1/4 Bridge Resistors (Optional)" in the following section.

WARNING **Ground the equipment: The safety Earth ground for the VT1529A/B is supplied through the ground conductor of the power cable. Make sure that the installation's ac line supply connectors provide a suitable Earth ground.**

WARNING **The power cord is the only way to disconnect the VT1529A/B from ac power. Therefore, the power cord must be accessible to the operator at all times. When the VT1529A/B 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 one or more VT1529A/B channels are required to provide 1/4 Bridge completion of other than 120 Ω or 350 Ω. Only those with experience in soldering components on printed circuit boards should attempt this installation. The VT1529A/B provides locations on its printed circuit board for the user's own 1/4 bridge completion resistors. Bridge configuration commands then can switch these resistors into the bridge completion circuits where the custom value resistors are installed.

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 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.

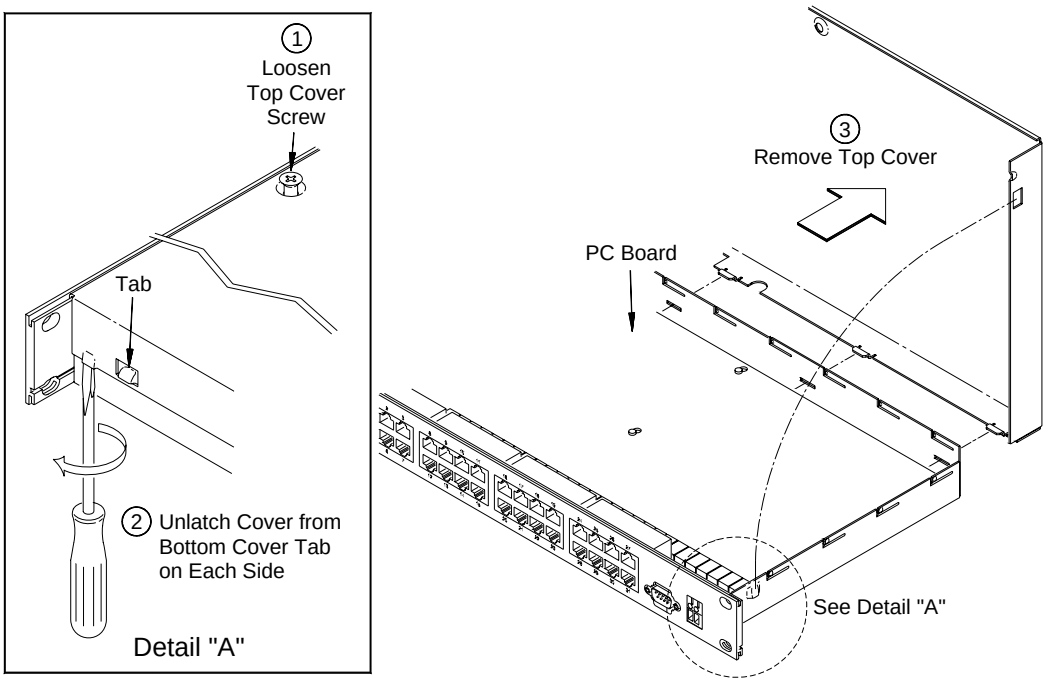


Figure 3-2. Removing the VT1529A/B Top Cover

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.

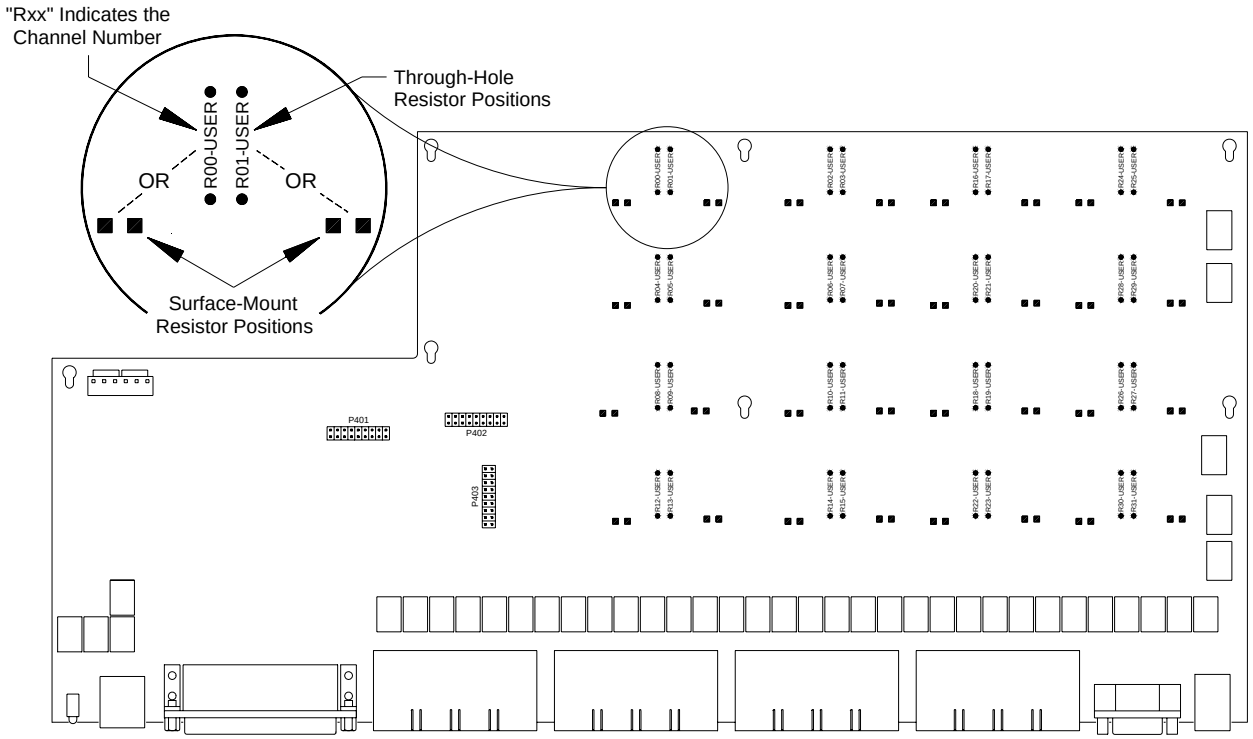


Figure 3-3. Locating User 1/4 Bridge Resistor Positions

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 installing through-hole resistors, be very careful to observe the specified maximum safe resistor lead length to avoid shorting the resistor to the chassis.

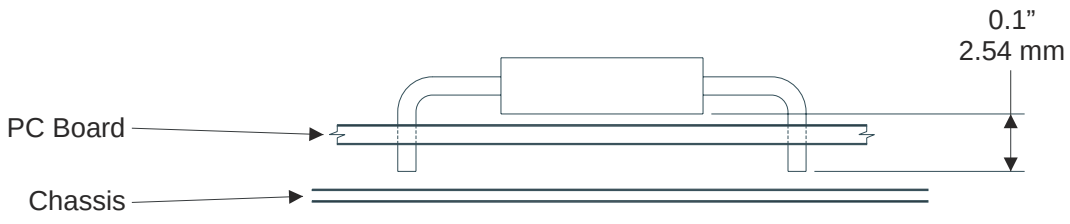


Figure 3-4. Installing User 1/4 Bridge Resistors

Connecting VT1529A/Bs to the VT1422A

The cable between a VT1422A and each VT1529A/B (connection "A" in Figure 3-1) is a standard type of cable used in computer Local Area Networks (LANs). The VT1529A/B can be any distance up to 1000 feet (304.8 m) from the VT1422A and the interconnect cable can be easily custom made to fit the installation. This type of cable assembly can typically be found in the IT (Information Technology) department of most companies.

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 VT1422A will be able to make accurate measurements from a VT1529A/B over the maximum cable length of 1000 feet (304.8 meters). Additionally, the cable and connectors must be shielded.

Cabling Supplies and Tools

Tables 3-1 and 3-2 show part numbers for supplies which can facilitate the production of custom high quality cables for the installation. If a third party builds the cables, make certain they supply cables that comply with the TIA/EIA-568 Category 5 standard and are they 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 the installation and that one should check applicable local codes and standards and select the proper type of cable accordingly (plenum vs. non-plenum types, etc.).

Table 3-1. Cable Part Numbers

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)	1624P
Non-Plenum Type: (PVC Jacket and polyolefin insulation)	1624R

‡DuPont trademark

Table 3-2. Connector Part Numbers

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 the Terminal Module ordered with the VT1422A, there are two methods of interconnecting a VT1529A/B to the VT1422A (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 VT1422A SCP positions contain a VT1539A Remote Channel SCP. For RSCUs, simply plug one end into the VT1422A and the other into the VT1529A/B'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 41 for on-board SCP channel connection through the RJ-45 connector module.

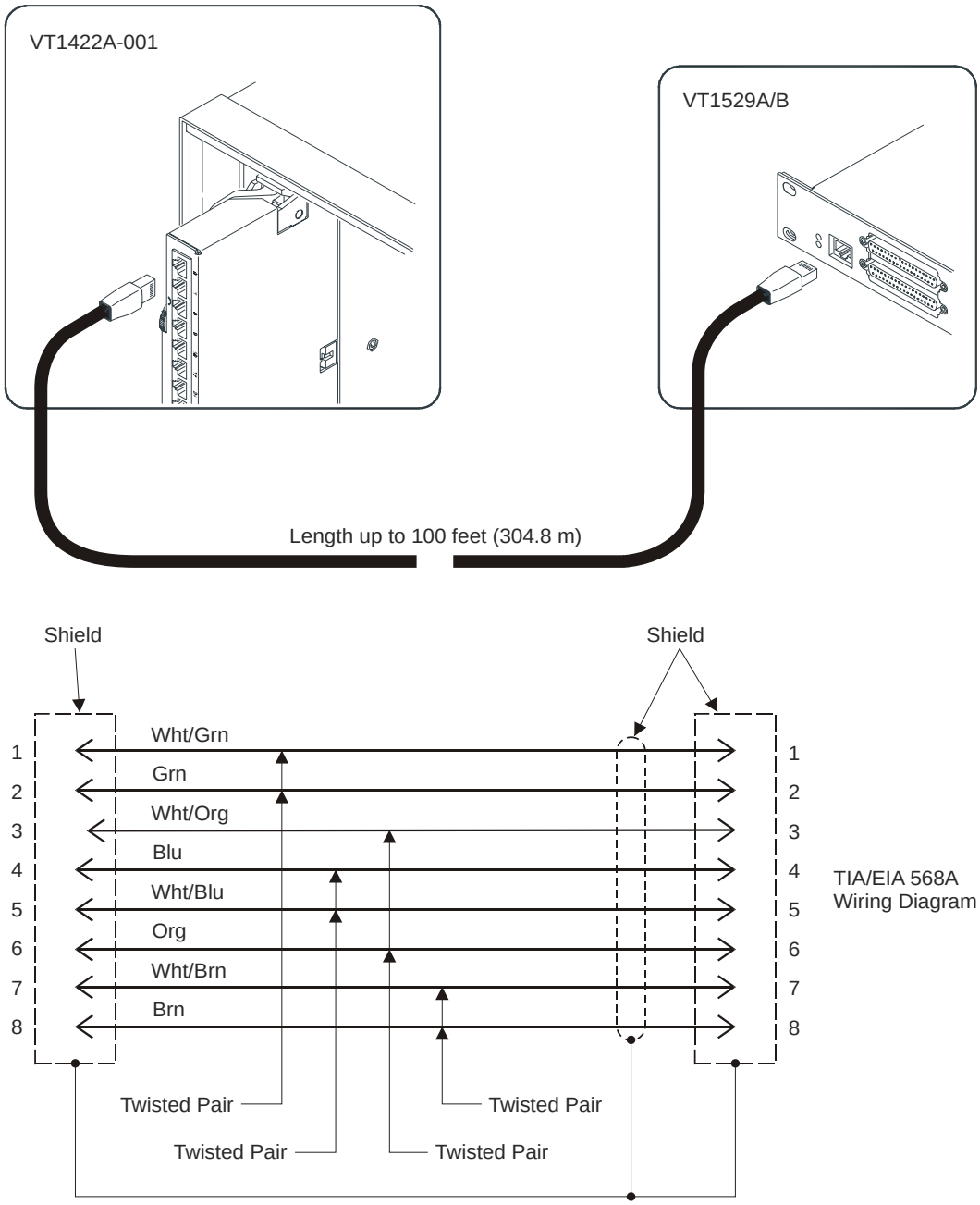


Figure 3-5. Connecting VT1529A/Bs to the RJ-45 Connector Module

Spring and Screw Terminal Modules

For mixed on-board SCP channels and RSCU operation, spring or screw type terminal modules can be used. For standard SCP channel connections, see Chapter 2 “Field Wiring” on page 35. For remote channels, connect the individual wires from each VT1529A/B’s data interface cable to the appropriate terminals for remote channel operation. The VT1539A 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 VT1539A signal name and associated terminal name as printed on the Terminal Module.

Note: In the table below, color combinations may vary.

Table 3-3. VT1539A Signal Names

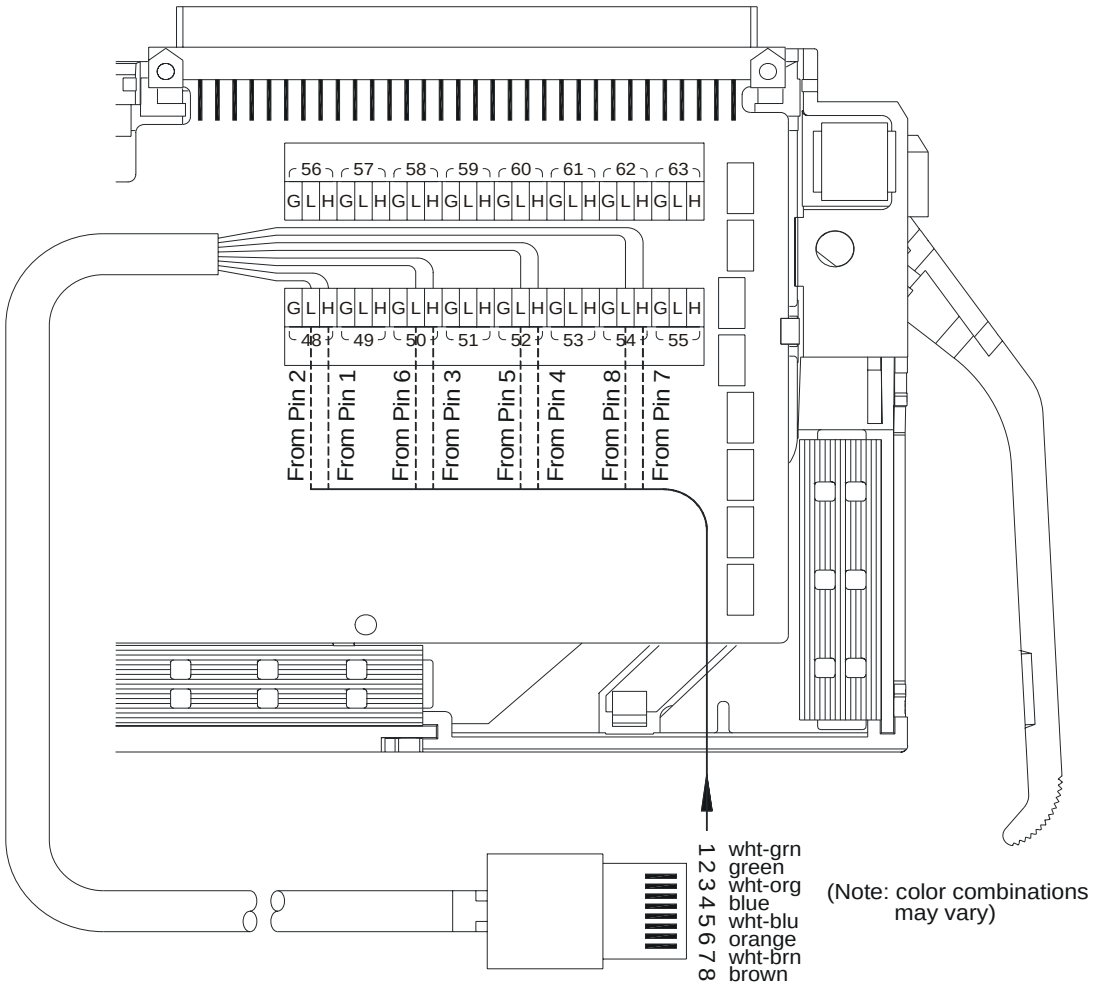
SCP Signal Names - to - Terminal Names				
SCP Position	**Plug Pin#**	**VT1539A Signal Name (with EIA/TIA-568A wire color-code)**	**Terminal Name on Terminal Module (SCP’s low channel)**	**Terminal Name on Terminal Module (SCP’s High Channel)**
SCP Position 0 Addresses 10000 to 10131	1	Analog+ (wht-green)	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

Table 3-3. VT1539A Signal Names

SCP Signal Names - to - Terminal Names				
SCP Position	**Plug Pin#**	**VT1539A 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 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
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 VT1529A/B Connection

Figure 3-6 shows a typical connection to a VT1529A/B through one of the optional terminal modules. In this case, the connection is to the low channel on the VT1539A 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 a VT1529A/B to an Optional Terminal Module

Connecting Excitation Supplies

This connection is shown as "B" in Figure 3-1. The VT1529A/B 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 power eight channels through a programmable switch. Multiple excitation inputs can be parallel-wired to a single power supply.

Notes

1. The excitation supply ***must*** have a balanced output with respect to ground. That is, the output must be centered about ground. For example, if the supply voltage is 5 V, then the positive lead should be +2.5 V and the negative lead at -2.5 V. Use of an unbalanced excitation supply causes longer than normal settling times in the VT1529A/B, which will cause errors during the measurement of the excitation voltage ***and the next few channels***.
2. The maximum excitation voltage the VT1422A can sense through the VT1529A/B's excitation sense path is 16 V (±8 V dc centered about the Gnd terminal). If a higher excitation voltage is supplied through the VT1529A/B, ***do not*** connect the excitation sense terminals.
3. Make sure that the power supply chosen can supply the current requirement of all of the bridges it can be switched to. It will be connected to all bridges to be measured before a measurement scan is started. The supply switches cannot be programmatically re-configured while a measurement scan is under way. The measurement scan must be halted to programmatically re-configure the excitation supply switches.

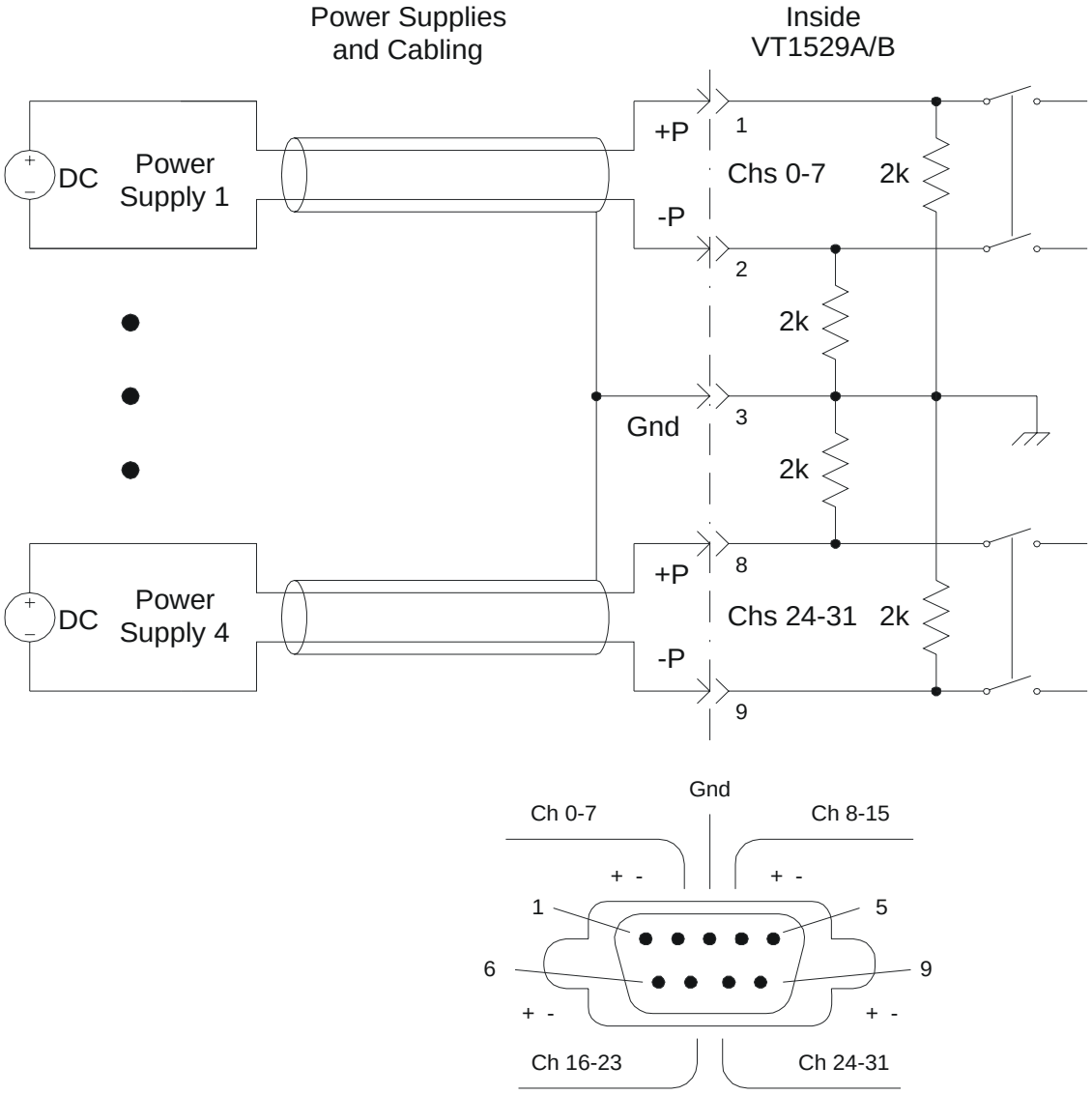


Figure 3-7. Excitation Supply Connections

Connecting the VT1529A/B to Strain Gages

The following discussion relates to the connection marked "C" in Figure 3-1 on page 58. Connecting the strain gages to the RJ-45 telecom connectors is illustrated below. 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 a gage connection example.

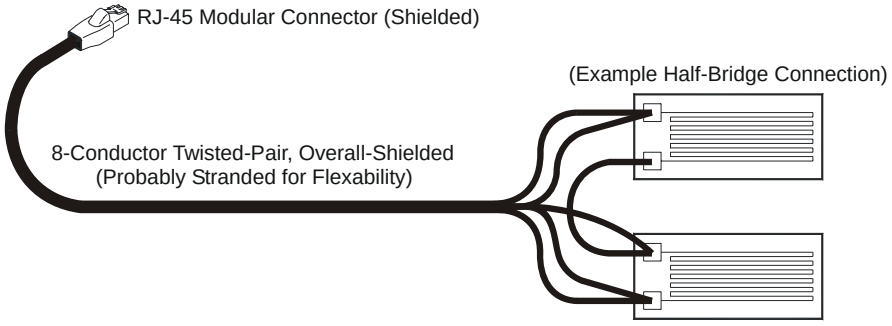


Figure 3-8. VT1422A to Strain Gage Connection

Channel Connector Pin-to-Signal Relationship

Figure 3-9 shows the pin-to-signal relationship for each VT1529A/B strain gage connector. The same signal names are found on the following strain bridge configuration illustrations as well.

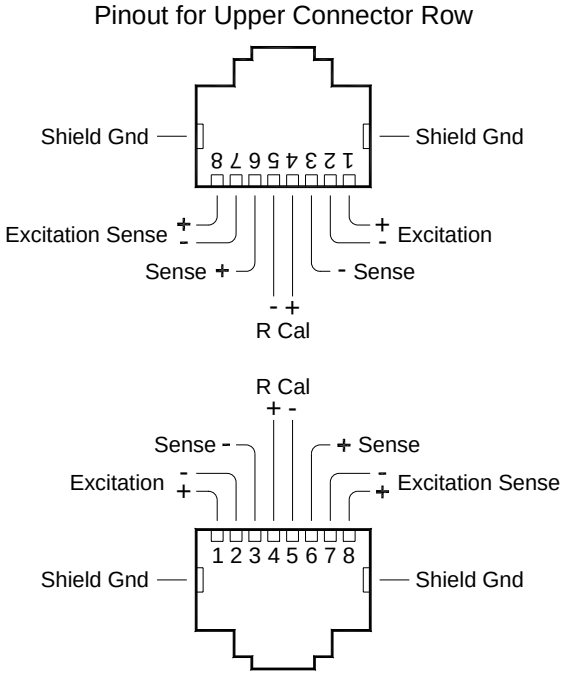


Figure 3-9. Pin-out for Strain Gage Connectors

VT1529A/B 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.

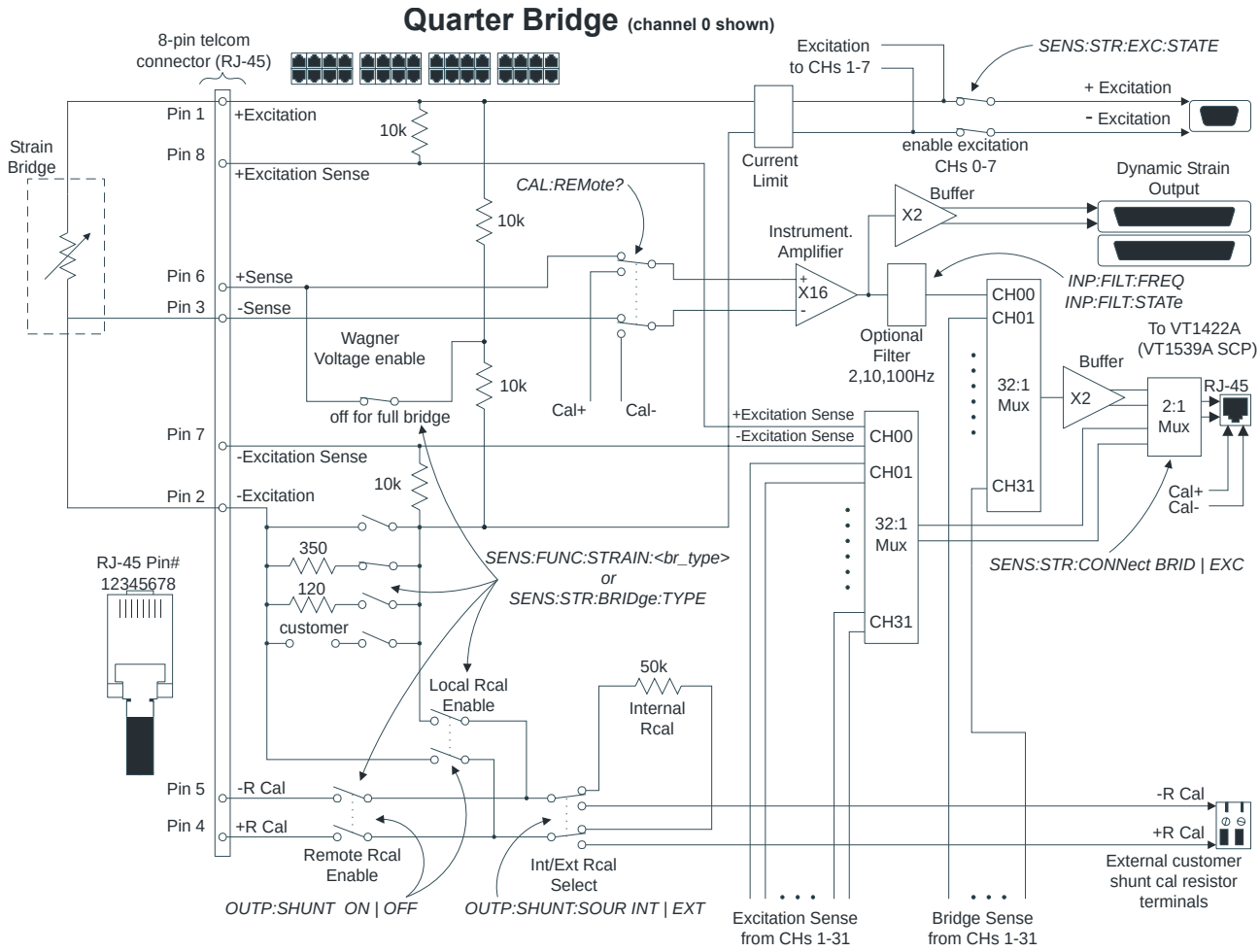


Figure 3-10. Bridge Completion for a Quarter Bridge Channel

Note

While the diagram above shows amplifier gain in the measurement path, the measurement values returned by these channels are corrected by the VT1422A's DSP (Digital Signal Processor) chip to reflect the actual value at the user input terminal. The only time gain must be considered 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 0.5 V would cause an overload reading even on the highest A/D range (16 V).

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.

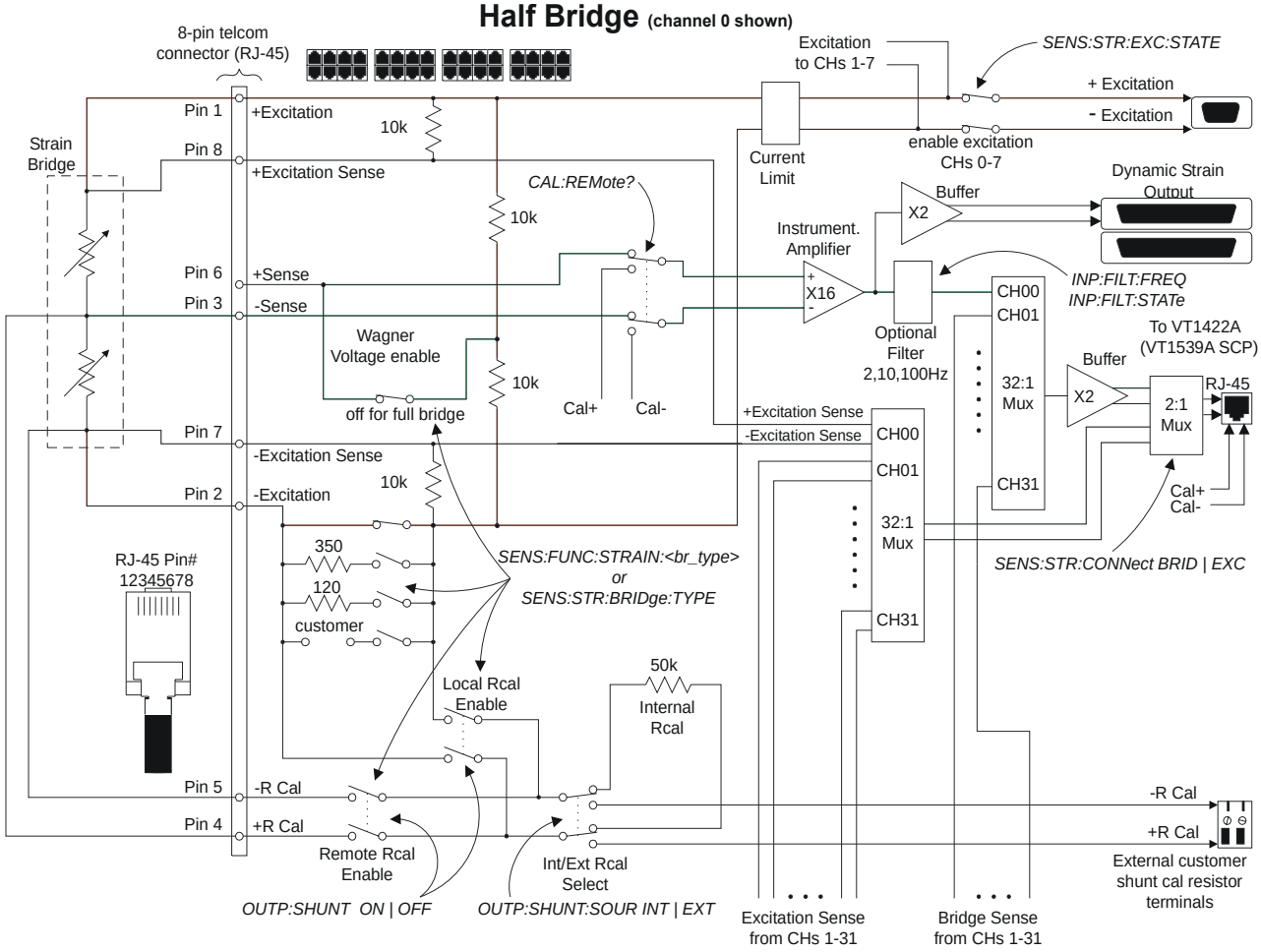


Figure 3-11. Bridge Completion for a Half Bridge Channel

Note While the diagram above shows amplifier gain in the measurement path, the measurement values returned by these channels are corrected by the VT1422A's DSP (Digital Signal Processor) chip to reflect the actual value at the user input terminal. The only time gain must be considered 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 0.5 V would cause an overload reading even on the highest A/D range (16 V).

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.

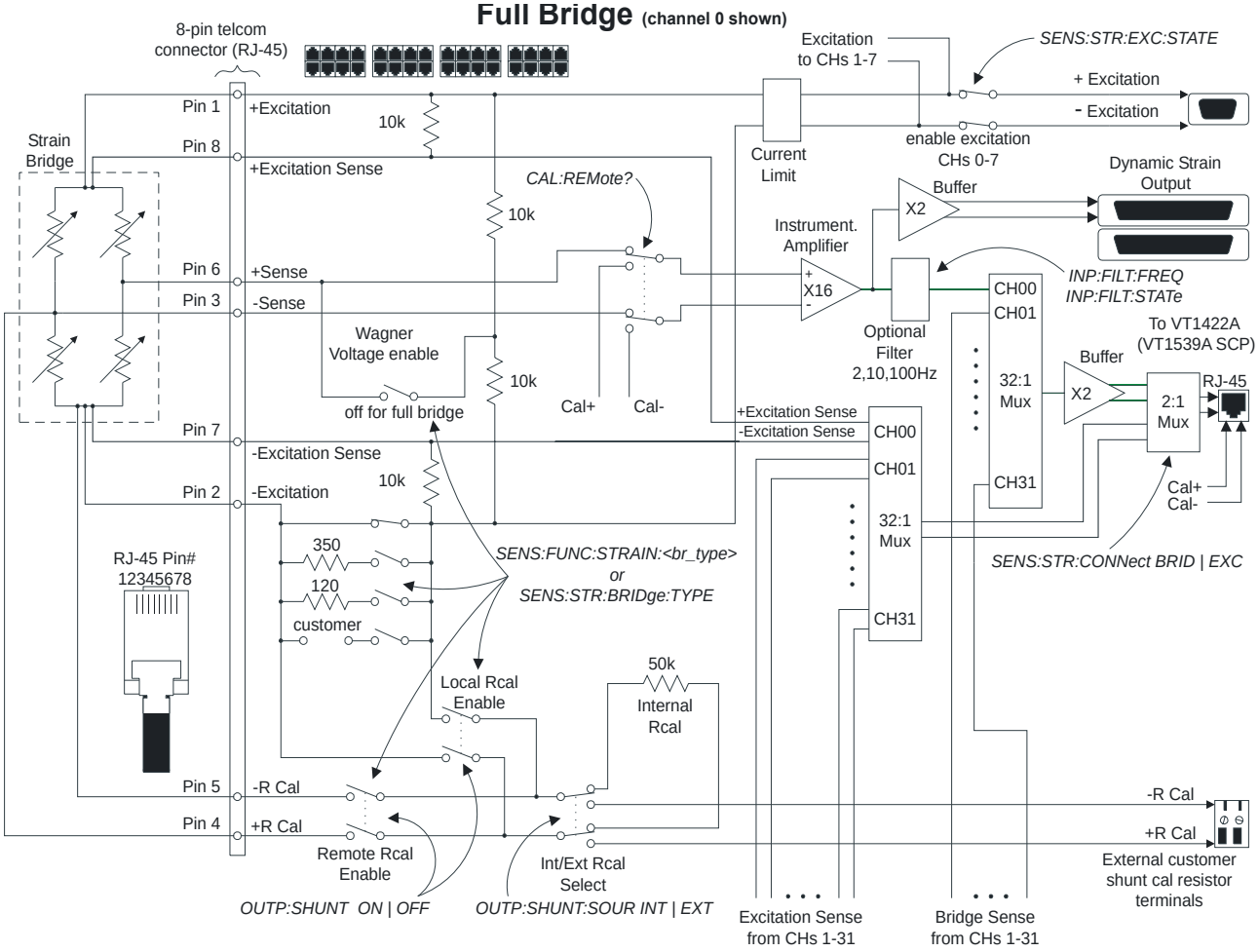


Figure 3-12. Bridge Completion for a Full Bridge Channel

Note

While the diagram above shows amplifier gain in the measurement path, the measurement values returned by these channels are corrected by the VT1422A's DSP (Digital Signal Processor) chip to reflect the actual value at the user input terminal. The only time gain must be considered 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 0.5 V would cause an overload reading even on the highest A/D range (16 V).

Connecting to the VT1529A/Bs Dynamic Strain Ports

The VT1529A/B has two, 37-pin connectors that provide wideband amplified outputs from each strain bridge signal. This allows for connection to a high-speed ADC-per-channel instrument like the VT1432A or VT1433B to capture dynamic strain events.

While an instrument like the VT1432A or VT1433B can measure signals from the VT1529A/B, a VT1422A is still required to control the VT1529A/B's bridge configuration, calibration and self-test functions.

One VT1422A can control up to sixteen VT1529A/Bs. Figure 3-13 shows the general interconnection layout for a VT1432A. The cable shown is the VT1529A/B Option 001. This cable is 10 feet (3.05 meters) long.

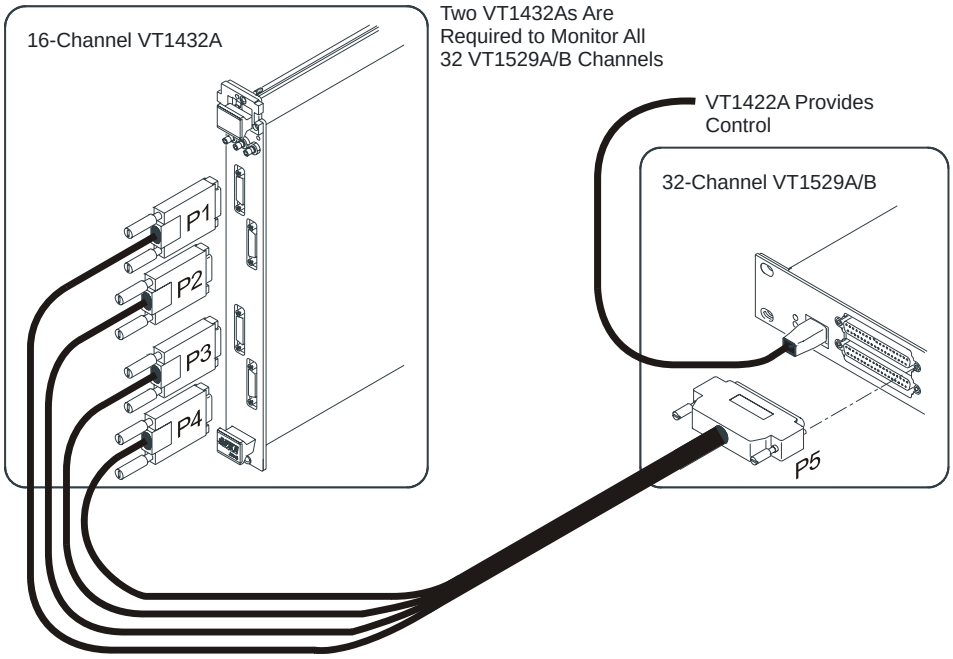


Figure 3-13. VT1432A to VT1529A/B Connection

Extending the Dynamic Strain Connection

If additional length is required, build or have an extender cable built with a male, 37-pin D-connector on one end and a female, 37-pin D-connector on the other. The extender cable must provide sixteen twisted pair conductors and be overall shielded. See "Dynamic Strain Extender Cable Pin-Out" on page 74.

Note

The spacing between the two "Buffered Output" connectors is narrow (0.625 in./15.875 mm) and requires narrow connector shells. The following two manufacturers' parts work well:

L-COM (distributor catalog Cat# SDRS37HOT)
Cinch DC24660 (Newark Cat# - 45F988)

Table 3-4. Dynamic Strain Extender Cable Pin-Out

Female 37-Pin Connector Pin Number	Signal Name	Male 37-Pin Connector Pin Number
1	Buffered Output 0+/16+	1
20	Buffered Output 0-/16-	20
2	Buffered Output 1+/17+	2
21	Buffered Output 1-/17-	21
3	Buffered Output 2+/18+	3
22	Buffered Output 2-/18-	22
4	Buffered Output 3+/19+	4
23	Buffered Output 3-/19-	23
5	Buffered Output 4+/20+	5
24	Buffered Output 4-/20-	24
6	Buffered Output 5+/21+	6
25	Buffered Output 5-/21-	25
7	Buffered Output 6+/22+	7
26	Buffered Output 6-/22-	26
8	Buffered Output 7+/23+	8
27	Buffered Output 7-/23-	27
9	Buffered Output 8+/24+	9
28	Buffered Output 8-/24-	28
10	Buffered Output 9+/25+	10
29	Buffered Output 9-/25-	29
11	Buffered Output 10+/26+	11
30	Buffered Output 10-/26-	30
12	Buffered Output 11+/27+	12
31	Buffered Output 11-/27-	31
13	Buffered Output 12+/28+	13
32	Buffered Output 12-/28-	32
14	Buffered Output 13+/29+	14
33	Buffered Output 13-/29-	33
15	Buffered Output 14+/30+	15
34	Buffered Output 14-/30-	34
16	Buffered Output 15+/31+	16
35	Buffered Output 15-/31-	35
17, 18, 19, 36, 37	Shield (drain wire)	17, 18, 19, 36, 37

Dynamic Strain Port Offset Control

Each buffered dynamic strain channel includes an offset adjusting DAC controlled by the command
SOURce:VOLTage[:AMPLitude] <-*offset_v*>,(@<*ch_list*>)
Reducing the unstrained bridge offset voltage at the dynamic strain port channel can allow the VT1432A to measure the channel using a more sensitive range. See Figure 3-14 for the offset DAC arrangement.

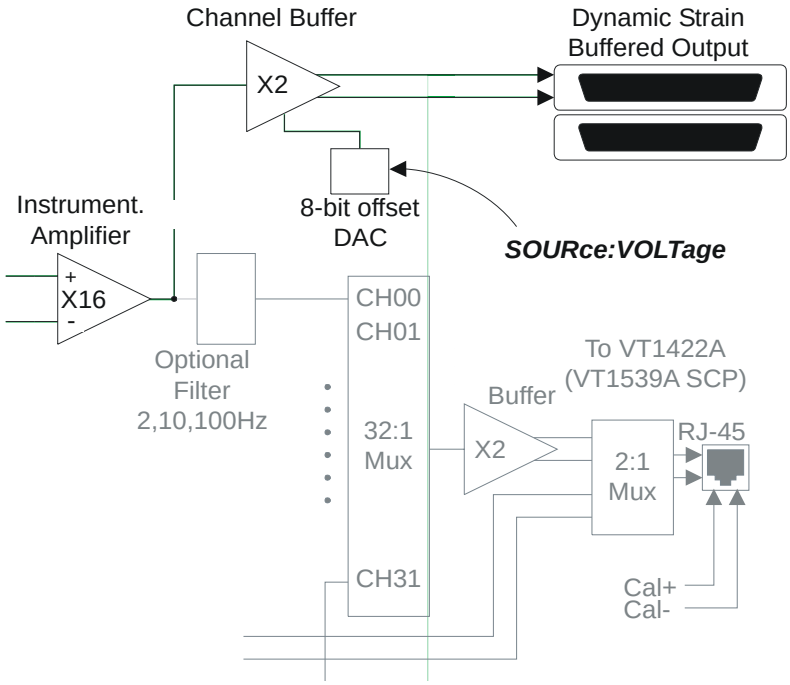


Figure 3-14. Dynamic Strain Offset DAC

To reduce the offset voltage at each dynamic strain "Buffered Output" channel:

1. Measure an unstrained Buffered Output channel with a VT1432A/33B and place the value in a variable arbitrarily called <*offset_v*>.

2. Send minus <*offset_v*> to that channel with the SOUR:VOLT command. For example: SOUR:VOLT <-*offset_v*>,(@10000)

Note With a 13 mV resolution, the offset DAC can reduce the Buffered Output channel offset to within a few millivolts of zero.

Remote Strain Channel Addressing

Figure 3-15 shows the relationship between SCP positions and Remote Channel Addressing through the VT1539A SCP (see Figure 2-1 on page 36 to compare with On-Board Channel Addressing). Not all SCP positions need to contain VT1539As. If needed, VT1539As can be mixed with other analog sense, source, and digital I/O SCPs.

Channels measured through Remote Signal Conditioning Units like the VT1529A/B Remote Strain Conditioning Unit are addressed with five digit channels specifiers rather than the traditional on-board channel's three digit specifier. Both three and five digit specifier start with a "1." This is the SCPI "card number" digit and is retained in the VT1422A for SCPI compatibility. The next two digits complete the specification of an on-board channel. When used in a five digit remote multiplexed channel specifier, the first three digits mean the same as in the on-board specifier. Digits two and three specify the VT1539A 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 VT1539A Remote Channel SCP. So, digits two and three will specify channels 00, 01, 08, 09, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, or 57. This allows the VT1422A to address up to sixteen VT1529A/Bs. Digits four and five 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 3-15 also:

Ch 0 on VT1529A/B connected to on-board chan 0 (VT1539A in SCP position 0).
(@10000)

Ch 0 on VT1529A/B connected to on-board chan 1 (VT1539A in SCP position 0).
(@10100)

Ch 24 on VT1529A/B connected to on-board chan 48 (VT1539A in SCP position 6).
(@14824)

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

channels 0 to 31 on each of the two VT1529A/Bs connected to on-board channels 0 and 1 (VT1539A in SCP position 0). This is 64 Chs.
(@10000:10131)

channels 0 to 15 on the VT1529A/B connected to on-board channel 24 (VT1539A in SCP position 3).
(@12400:12415)

combined previous two examples into a single scan list to show combining ranges.
(@10000:10131,12400:12415)

Runtime Remote Scan Verification

The VT1422A provides a method to verify that remote channels in the scan list defined in algorithms or with the ROUTe:SEQuence DEFine command are successfully scanned in each RSCU. See "Runtime Remote Scan Verification" on page 98, "The Operating Sequence" on page 133 and "Runtime Remote Scan Verification" on page 187.

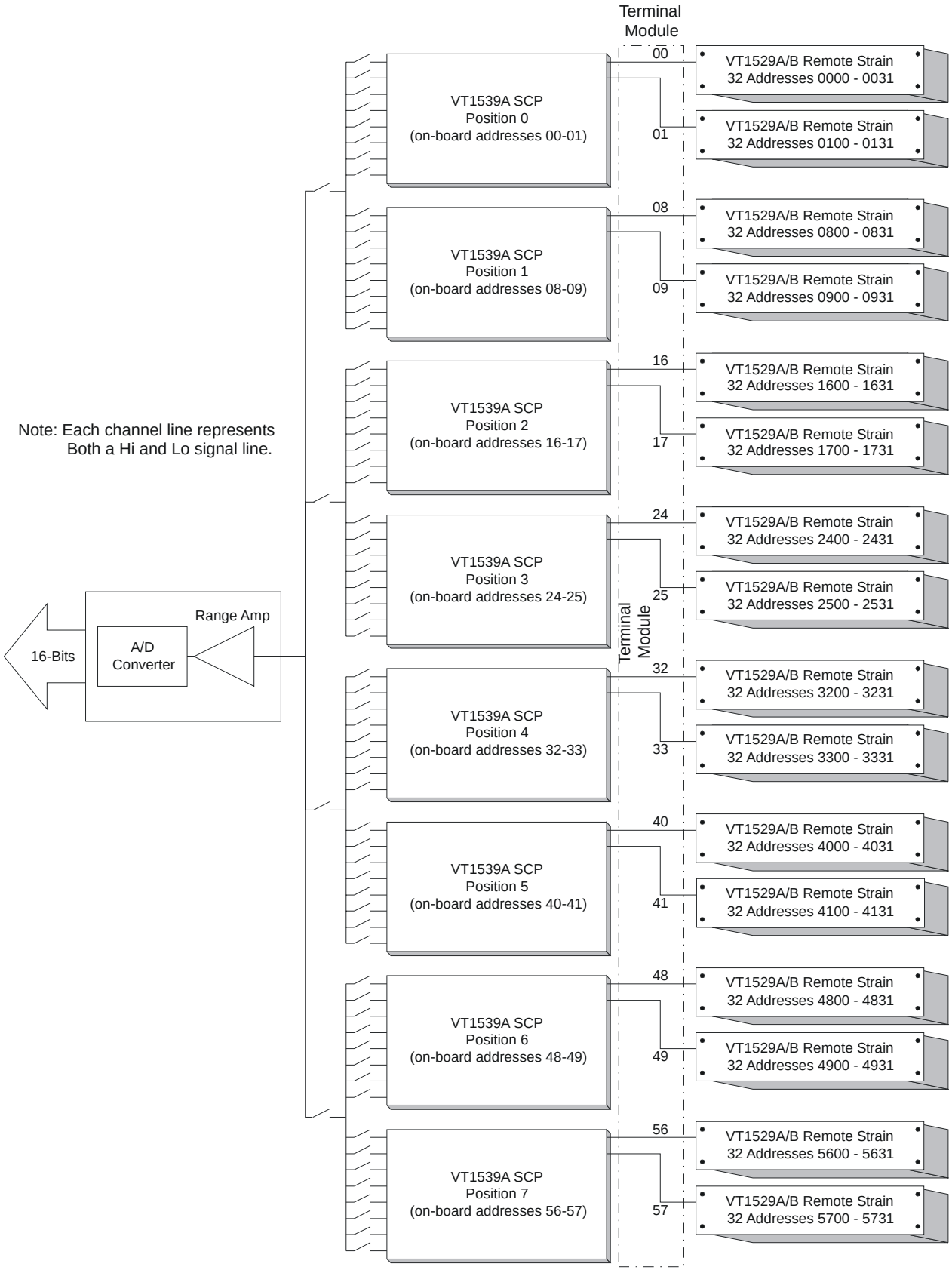


Figure 3-15. Remote Strain Channel Addressing

Programming for Remote Strain Measurement

This programming section is focused exclusively on programming the VT1422A and VT1529A/B for remote strain measurement. For more general VT1422A programing see Chapter 4 “Programming the VT1422A 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 an *RST command. Where these default settings coincide with the configuration settings required, there is no 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.
- VT1529A/B input filters:
  -- INPut:FILTer:FREQuency 10,(@<*all channels*>)
  -- INPut:FILTer:STATe ON,(@<*all channels*>)
- All analog input channels linked to the EU conversion for voltage.
- ARM:SOURce IMMediate
- TRIGger:SOURce TIMer
- TRIGger:COUNt 1
- TRIGer:TIMer .010 (10 ms)
- 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 actual value to make reasonable measurements).

- The default for the VT1529A/B 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 VT1529A/B. 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 419 depend on recent *CAL? and CAL:REMote? operations.

Measure Strain Using Built-in Strain EU Conversion

This method lets the VT1422A convert the strain bridge readings to units of strain (∈) 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 VT1422A to make the Engineering Unit conversion to strain. In fact, this is considered the "normal" VT1422A measurement method.

When the command SENSe:FUNC:STRain:<*bridge_type*> is sent, the specified bridge type is configured by switches in each VT1529A/B, the channel inputs are connected to the bridge outputs (see Figure 3-10 through Figure 3-12 starting on page 70) 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 VT1422A 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.

The user provides the above information to the VT1422A and 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? query (recommended) which will take an average of 32 voltage readings on each specified channel and save the values internally for use later by the strain EU conversion process. When using this method, loaded algorithms are not executed to avoid putting extraneous readings into the FIFO buffer. The voltage readings are also sent to the FIFO buffer for review.

   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 above must be sent back to the VT1422A'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 VT1422A.

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. This value MUST be changed to obtain reasonable reading values. There are two ways to change any channel's value:

   a. Use the MEAS:VOLTage:EXCitation? query (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 for review later if desire.

   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 VT1422A'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 VT1422A.

Figure 3-16 shows the sequence of commands to measure remote strain channels using the built-in strain Engineering Unit Conversion routines.

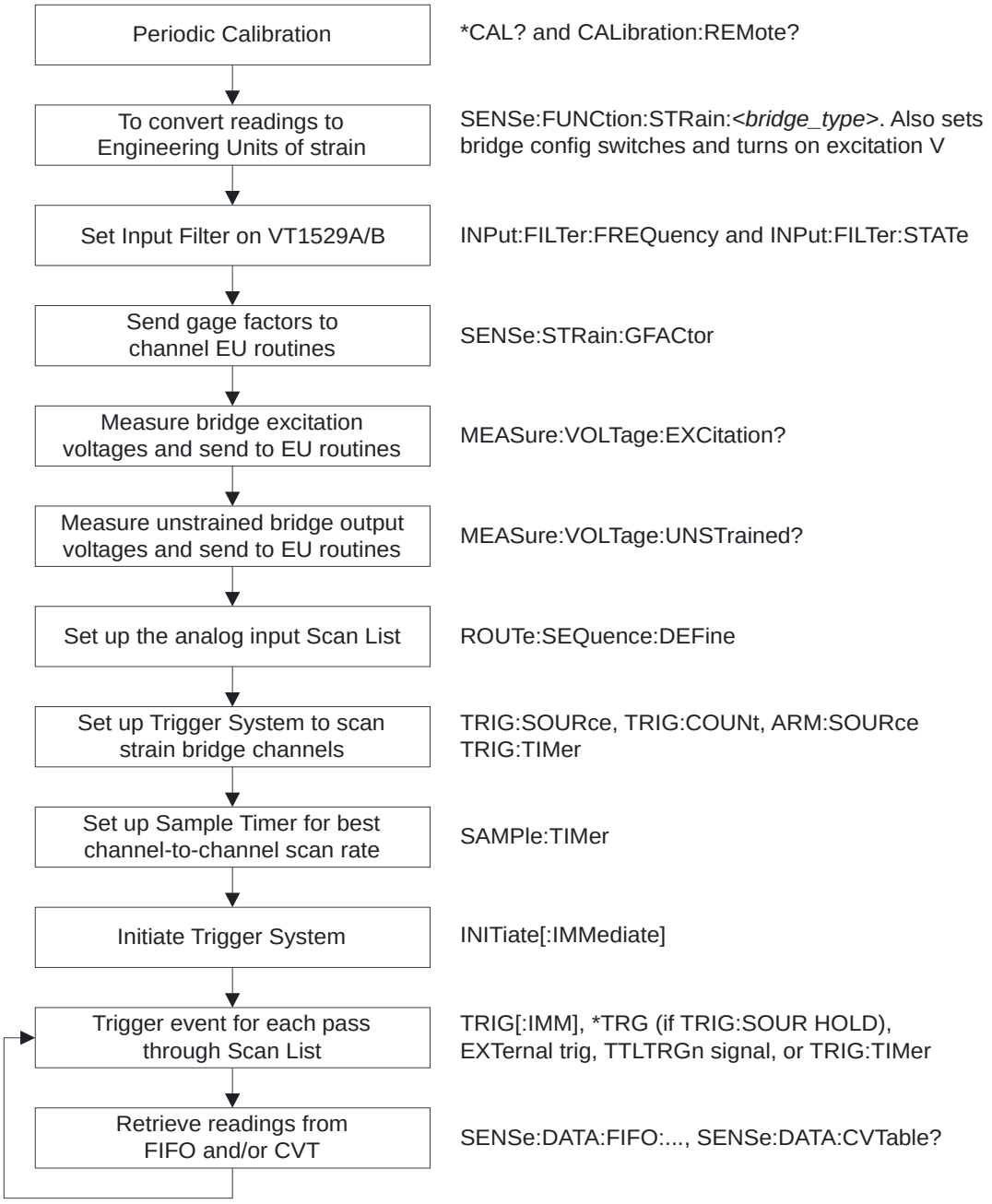


Figure 3-16. Sequence for Built-in Strain EU Conversion

Built-in EU Conversion Command Sequence

Below is an example VXI*plug&play* command sequence. Note that this is not executable; it has been simplified for easier reading. The C++ example source file (*euseq.cpp*) is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&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 VT1529A/B input filters (2, 10 or 100Hz) */
errStatus = hpe1422_cmd(sessn,"input:filter:frequency 10,(@10000:10007)");
/* optionally enable VT1529A/B input filters (approx 100 kHz when OFF) */
errStatus = hpe1422_cmd(sessn,"input:filter:state ON,(@10000:10007)");

/* send gage factors to channel EU conversion 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 acquisition, 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 VT1422A measures voltage and then applies a conversion routine (linear) supplied by the user. The user must supply the slope (M) and offset (B) 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 Ω), Q350 (quarter bridge, 350 Ω) 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 [SENSe:]STRain:CONNect BRIDge,(@<*ch_list*>) command. 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, both of which are functions of the excitation voltage, the unstrained reference and the gage factor.

5. The VT1422A must be told to use the custom conversion when taking measurements. This is done by sending the SENSe:FUNC:CUSTom [<*range*>,](@<*ch_list*>) command.

Figure 3-17 shows the sequence of commands to convert remote measurements according to the user's own downloaded EU conversion method.

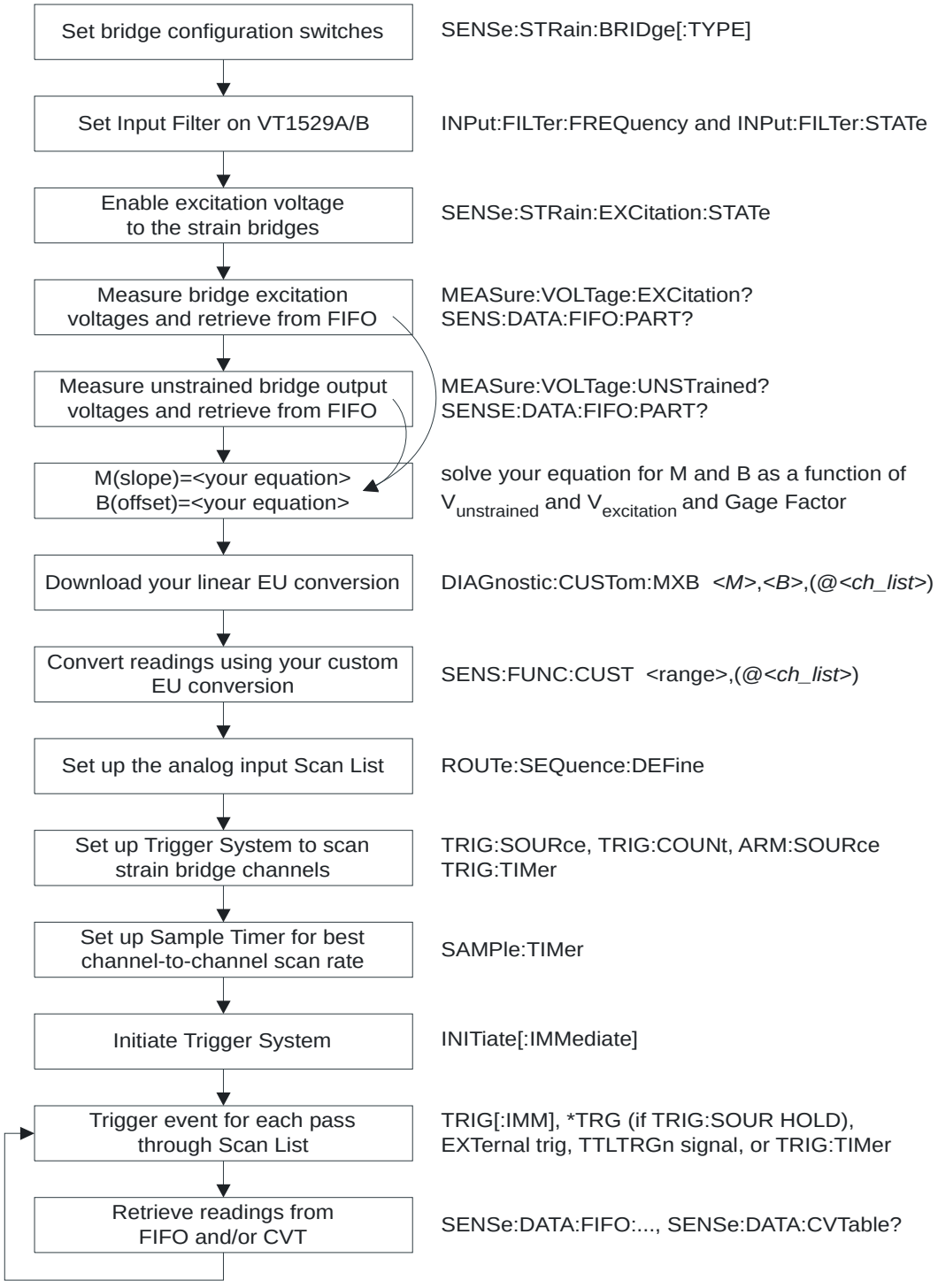


Figure 3-17. Sequence for User's Custom EU Conversion

Custom EU Conversion Command Sequence

Below is an example VXI*plug&play* command sequence. Note that this is not executable; it has been simplified for easier reading. The C++ example source file (*mxbseq.cpp*) is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

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

/* optionally set VT1529A/B input filters (2, 10 or 100 Hz) */
errStatus = hpe1422_cmd(sessn,"input:filter:frequency 10,(@10000:10007)");
/* optionally enable VT1529A/B input filters (approx 100 kHz when OFF) */
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 "VT1529A/B 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 acquisition, 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 this method is desired, voltage measurements will be made at the strain bridges while unstrained, then again while under strain. The excitation voltage will also be measured at each bridge. Using this data as well as the gage factor, strain conversion equations can be calculated in the computer.

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

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 Ω), Q350 (quarter bridge, 350 Ω) 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 [SENSe:]STRain:CONNect BRIDge,(@<*ch_list*>) command.
   The power on and reset setting is BRIDge.

4. Use the MEAS:STR:UNSTrained? (@<*ch_list*>) query 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, loaded algorithms are not executed to avoid putting extraneous readings into the FIFO buffer.

5. Use the MEAS:STR:EXCitation? (@<*ch_list*>) query 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, 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. The readings are then retrieved and used to calculate the strain for each channel using the excitation, unstrained, and strained voltage values.

Figure 3-18 shows the sequence of commands to convert bridge voltage measurements to strain by post-processing.

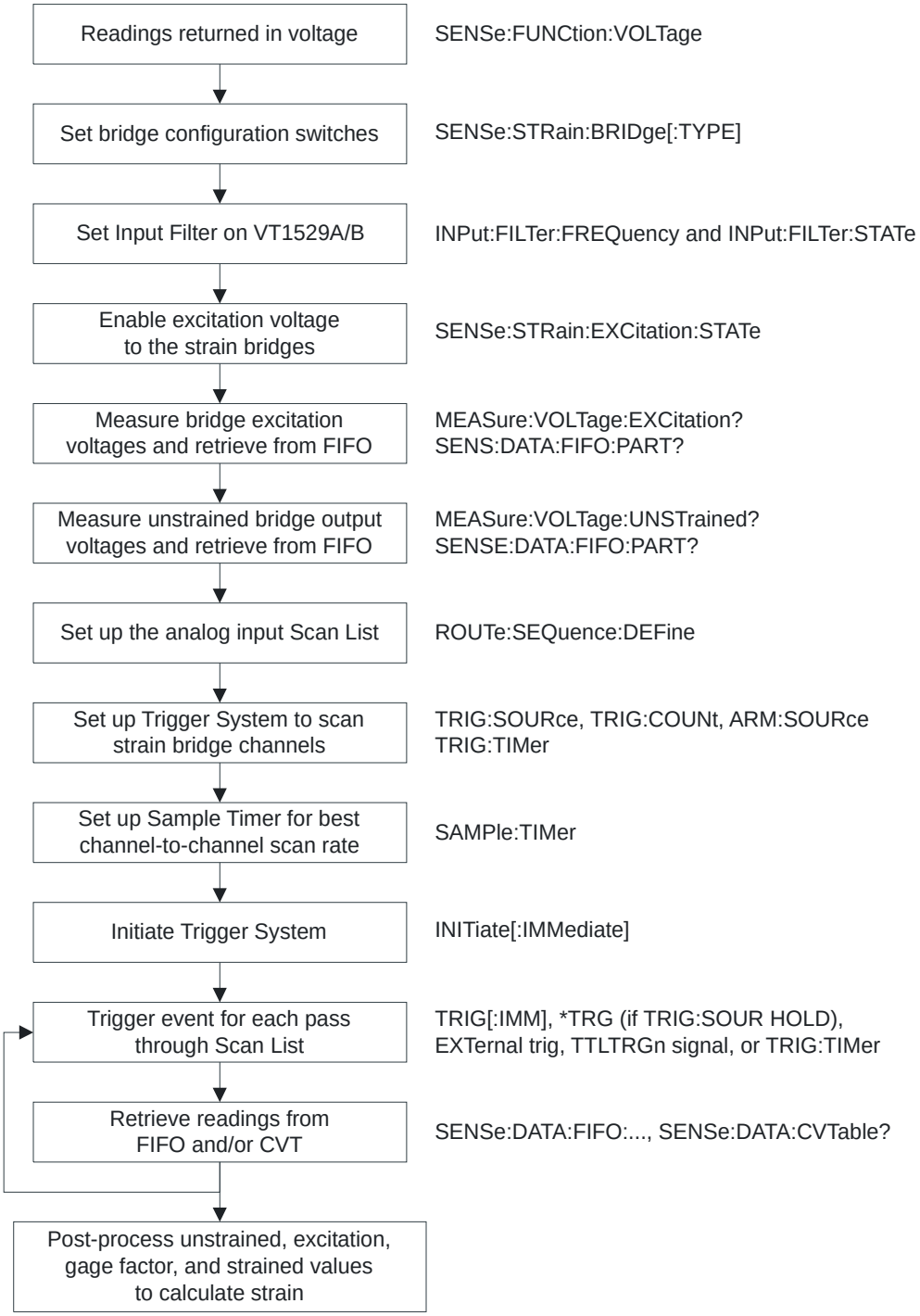


Figure 3-18. Converting Bridge Voltage Measurements to Strain

Voltage Conversion Command Sequence

Below is an example VXI*plug&play* command sequence. Note that this is not executable; it has been simplified for easier reading. The C++ example source file (*voltseq.cpp*) is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&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 VT1529A/B input filters (2, 10 or 100 Hz) */
errStatus = hpe1422_cmd(sessn,"input:filter:frequency 10,(@10000:10007)");
/* optionally enable VT1529A/B input filters (approx 100 kHz when OFF) */
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 "VT1529A/B 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 acquisition, 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 factor *
 *  and calculate individual strain values for each channel using    *
 *  your own equations.                                             *
 *                                                                  *
 ********************************************************************/
```

Verifying Correct Bridge Completion (Shunt Cal)

Verifying bridge configurations and connections is accomplished by inserting a known resistance (shunt cal resistor) in parallel with one leg of the bridge to imbalance it by a predictable amount. The VT1529A/B provides a single, internal 50 kΩ shunt cal resistor that can be programmatically connected to each of the 32 channels, one channel at a time. The VT1529A/B also provides the same connection capability for an optional external user supplied shunt cal resistor. The user's shunt cal resistor can be connected via the front panel "Shunt Cal Resistor" terminal block. See Figure 3-19.

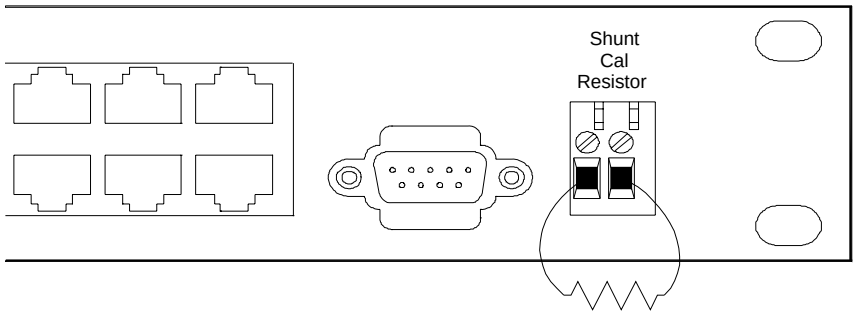


Figure 3-19. User Shunt Cal Resistor Connection

For the following discussion, refer to Figure 3-10 through Figure 3-12. The OUTPut:SHUNt:SOURce INT | EXT,(@<*ch_list*>) command selects either the INTernal (built-in) or EXTernal (user supplied) shunt cal resistor. Use the OUTput:SHUNt ON | OFF,(@<*ch_list*>) command to actually connect the shunt cal resistor to the bridge to be tested. For OUTP:SHUN, <*ch_list*> may specify only a single channel on any one VT1529A/B. This is because a single resistor is used to shunt each of a VT1529A/B's 32 channels. When the command is sent to connect another channel, the previously closed channel is opened.

To perform shunt cal on multiple channels, the program will have to enter a loop to connect the shunt cal resistor to sequential channels and read the result from the shunted channel. Generally, only OUTP:SHUN OFF needs to be sent to open the last channel closed on a particular VT1529A/B. For quarter bridge completion, the shunt cal resistor is connected locally (on-board the VT1529A/B). For both half and full bridge completion, the shunt cal resistor is connected remotely via the -Real and +Real terminals. The switches that route Real are automatically controlled by the bridge configuration commands [SENSe:]FUNCtion:Q120, [SENSe:]FUNCtion:Q350, [SENSe:]FUNCtion:USER, [SENSe:]FUNCtion:HBEN, [SENSe:]FUNCtion:FBEN, and [SENSe:]STRain:BRIDge[:TYPE].

See Figure 3-20 for a general shunt cal programming sequence. A C++ example source file (*shuntcal.cpp*) is available in the VXI*plug&play* help file and on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

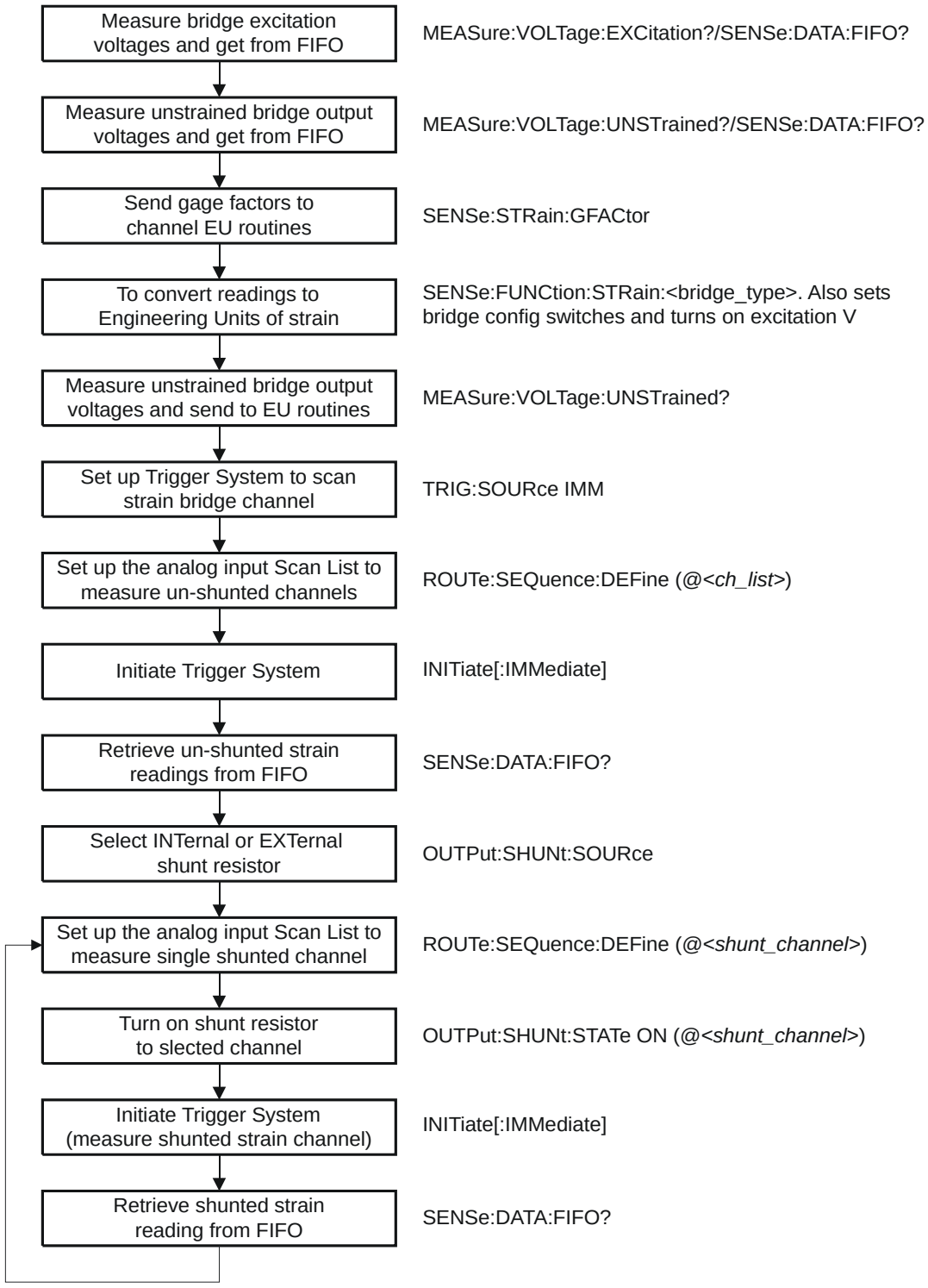


Figure 3-20. Performing Shunt Calibration

Built-in Strain Conversion Equations

When using the VT1422A'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 VT1422A to convert bridge measurements to Engineering Units of Strain for channels specified in the command SENSe:FUNCtion:STRain:FBEN <*range*>,(@<*ch_list*>).

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

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 VT1422A to convert bridge measurements to Engineering Units of Strain for channels specified in the command SENSe:FUNCtion:STRain:HBEN <*range*>,(@<*ch_list*>).

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

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 used by the VT1422A to convert bridge measurements to Engineering Units of Strain for on-board strain SCP channels only. Because VT1529A/Bs can expand the VT1422As strain channel count to 512, use of this non-linear strain conversion equation would require too much VT1422A memory. Instead, a quadratic 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

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

Quarter Bridge Equation for VT1529A/B only

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

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

Error Analysis

Figure 3-21 compares the non-linear quarter bridge equation used for strain SCPs with the linear approximation used with the VT1529A/B. Notice that while the error is independent of excitation voltage and unstrained voltage, error is quite sensitive to gage factor.

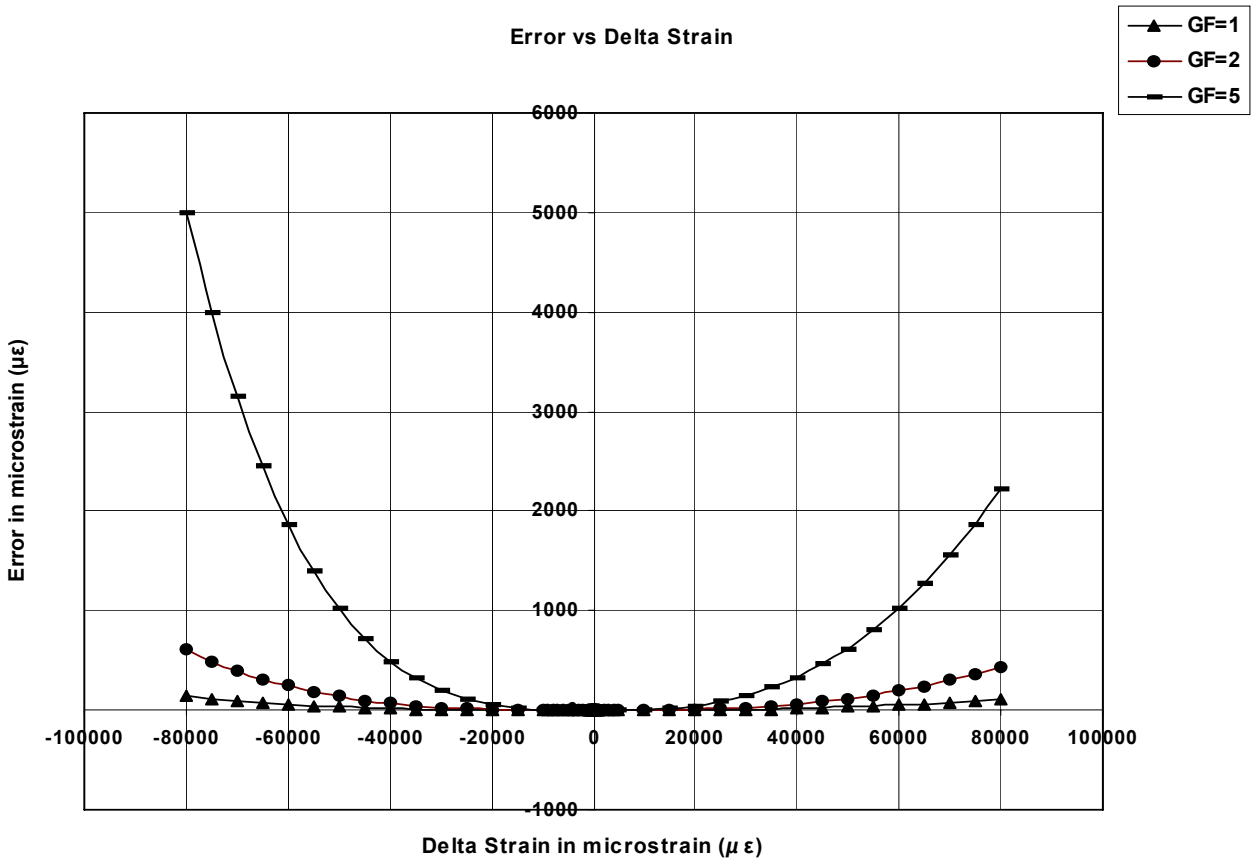


Figure 3-21. Error of Quarter Bridge Linear Approximation

Chapter 4
Programming the VT1422A for Data Acquisition and Control

About This Chapter

The focus in this chapter is to show the VT1422A's general programming model. The programming model is basically the sequence of SCPI commands the application program will send to the VT1422A to configure it to execute the defined Scan List and/or algorithms. This chapter contains:

Overview of the VT1422A Multifunction DAC Module

This section describes how the VT1422A gathers input data, executes its 'C' algorithms and sends its output data. Figure 4-1 shows a simplified functional block diagram.

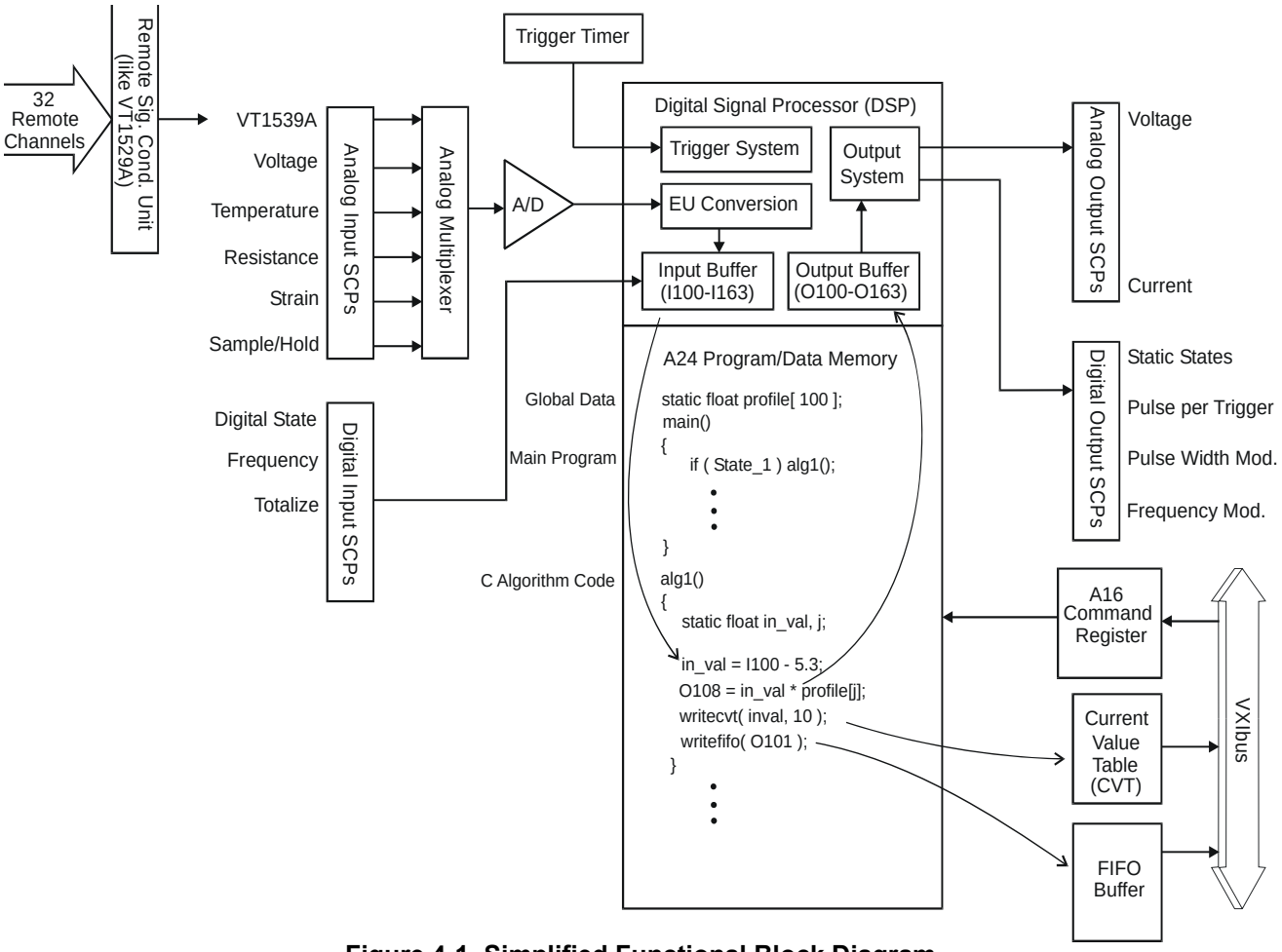


Figure 4-1. Simplified Functional Block Diagram

Multifunction DAC?

The VT1422A is a complete data acquisition and control system on a single VXI card. It is multifunction because it uses the Signal Conditioning Plug-on (SCP) concept whereby analog input/output and digital input/output channels can be mixed and matched 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 VT1422A has eight SCP positions with each SCP position capable of addressing up to eight channels of input or eight channels of output for a total of 64 channels. These 64 channels are known as the on-board channels. The eight SCP slots can be used for any of the analog/digital SCPs available for the VT1422A which cover most data acquisition and control needs.

Remote Multiplexing and Signal Conditioning

In addition, each SCP slot that contains a VT1539A SCP can operate two Remote Signal Conditioning Units (RSCUs) that externally multiplex up to 32 channels each. These channels are known as remote channels because they are multiplexed remotely to the VT1422A. So, with 32 channels per RSCU and 16 RSCUs, the VT1422A can make analog measurements on 512 remote channels. The upper left corner of Figure 4-1 shows how Remote Signal Conditioning Units fit in.

Programmable Signal Conditioning and EU Conversion

The input and output SCPs are configured with the SCPI and/or VXI*plug&play* programming. Analog SCPs are measured with the VT1422A's A/D. Configuring the analog SCPs include specifying what type of Engineering Unit (EU) conversions are desired 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 SCPs perform their own conversions as configured by the SCPI language.

Scan List and/or C Language Control Programming

The VT1422A 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 VT1422A can execute its own internal 'C' language algorithms which can perform data acquisition and control and pass values to the supervisory computer 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.

Runtime Remote Scan Verification

The VT1422A provides a method to verify that remote channels in the scan list defined in algorithms or with the ROUTe:SEQuence DEFine command are successfully scanned in each RSCU. Special algorithm variables are available to check the operating status of each VT1539A main channel. This provides protection against an RSCU failing to scan remote channels due to a signal cable disconnect or a power failure at the RSCU.

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 the 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. from Scan List channels (ROUT:SEQ:DEF) are 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 the 'C' algorithms executing on the VT1422A card.

Analog input values from channels in the Scan List, stored in the FIFO and/or CVT can be read from the VT1422A without creating or running any 'C' algorithms. This makes for easy traditional analog data acquisition where no control aspect is required.

Start Algorithms

Approximately 2000 lines of user-written 'C' code can be downloaded into the VT1422A's memory and can be split among up to 32 algorithms. VXI Technology 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, they can be thought of in traditional terms, with 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 4-1, 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

The 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 VT1422A's FIFO and Current Value Table (CVT). As seen in Figure 4-1, any algorithm can write any expression or constant to the FIFO/CVT. The application can then read the FIFO/CVT to characterize what's happening inside the VT1422A 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.

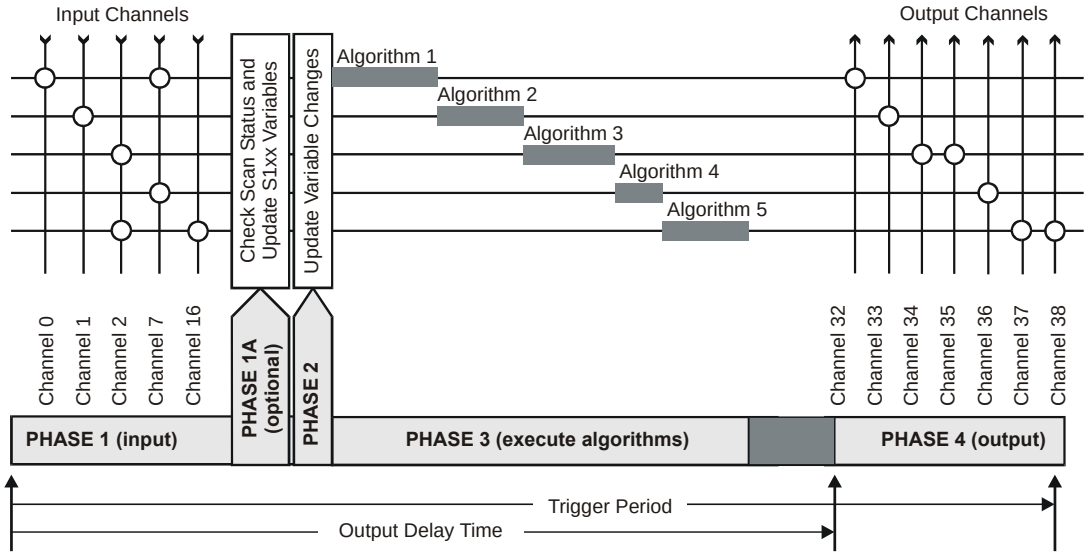


Figure 4-2. Instrument Operation Cycle Phases

Detailed Instrument Operation Cycle

Figure 4-2 illustrates the timing of all these operations and describes the VT1422A's input-update-execute algorithms-output phases. This cycle-based design is desirable because it results in deterministic operation of the VT1422A. 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.

In Phase 1, all input channels specified in the ROUTe:SEQuence:DEFine command and/or referenced in downloaded algorithms are scanned.

Phase 1A is for Runtime Remote Scan Verification and is optional. When one or more special scan status variables (S1xx) are included in an algorithm, this time is required to evaluate the scan status of each VT1539A SCP channel reference by a status variable. The time required is 230 μs + 40 μs * (number of S1xx variables referenced). If no status variables are referenced in any algorithms, then Phase 1A is not executed.

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 several 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 kept 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 VT1422A's ability to execute programs directly on the card and its fast execution speed give the programmer real-time response to changing conditions. Plus, programming the card has been made very easy to understand. The C language was chosen for writing user programs as this language is already considered the industry standard. Choosing C allows the user to write algorithms on PCs or UNIX workstations that have C compilers, so they can debug algorithms before execution on the card. The VT1422A also provides good debugging tools that permit users to determine worst-case execution speed, monitor variables while running, and selectively enabling/disabling any of the VT1422A's 32 algorithms.

A limited and simplified version of C was created 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 VT1422A'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 and vary considerably. This can cause undesirable output jitter when the beginning of the output phase starts immediately after the last user algorithm executes. The VT1422A'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 simplified 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 VT1422A'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. This allows for more time to solve the application in terms of input and output value variables rather than dealing with the individual differences of each SCP.

Programming Model

SCPI commands and/or VXI*plug&play* driver functions are used to configure, start, stop, and communicate with the VT1422A. 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. These two states will be called "before INIT" and "after INIT," respectively. 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, commands will be sent 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 VT1422A's driver disallows execution of most SCPI commands and VXI*plug&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 acquisition values from the CVT and FIFO and return Status System values. The Command Reference Section (Chapter 7) 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 VT1422A to make measurement scans and/or run algorithms.

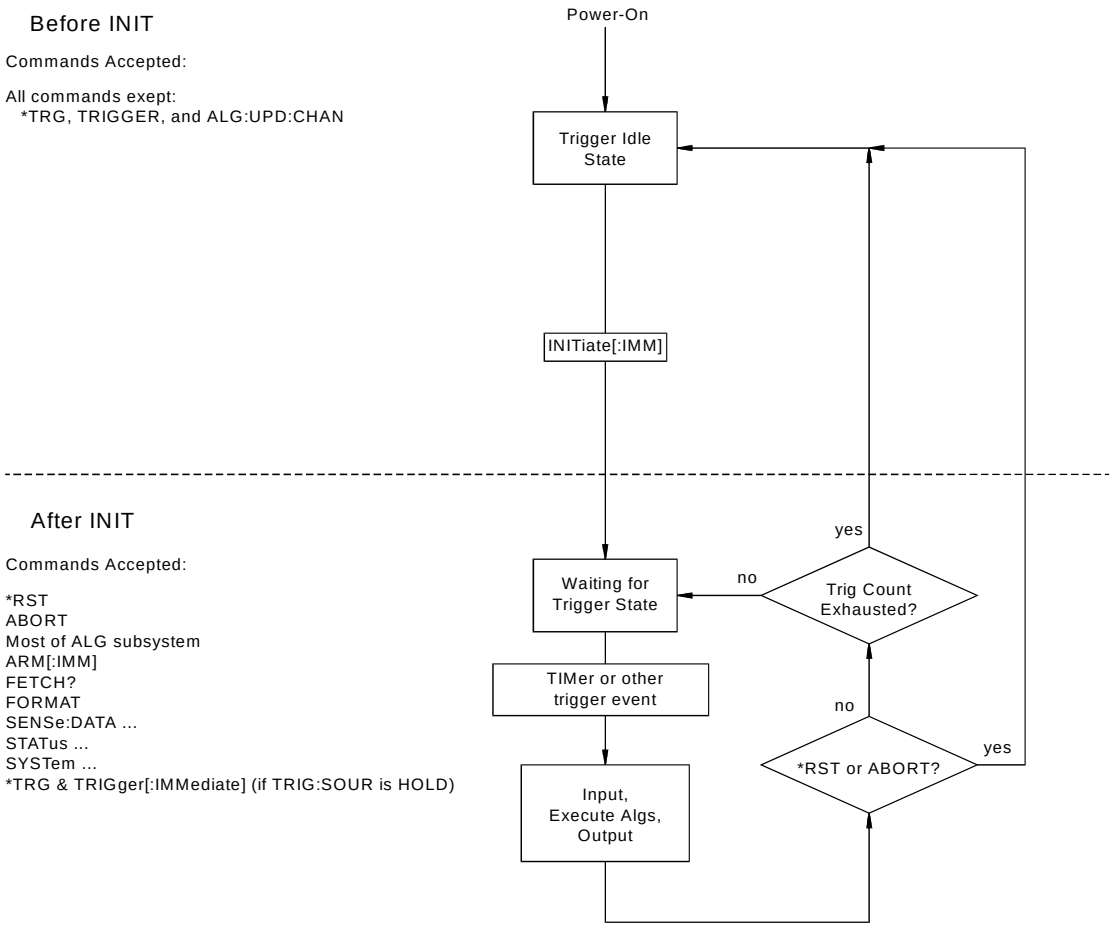


Figure 4-3. Module States

Executing the Programming Model

This section shows the sequence of programming steps that should be used for the VT1422A. Within each step, most of the available choices are shown using command sequence examples, with further details available in the Command Reference Chapter 7.

IMPORTANT! It is very important that, while developing an application, the SYSTem:ERRor? query be executed after each programming command. This is the only way to determine if a programming error has occurred. 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 required, there is no 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 (40 μs)
- TRIGger:SOURce TIMer
- TRIGger:COUNt 1
  (Note that this default was chosen to make testing data acquisition scan list easier. For algorithm operation, changing the count to INFinite is probably desired.)
- TRIGer:TIMer .010 (10 ms)
- FORMat ASC,7 (ASCII)
- SENSe:DATA:FIFO:MODE BLOCking

Figure 4-4 provides a quick reference to the Programming model. Refer to this along with the "Programming Overview Diagram" to keep an overview of the VT1422A SCPI programming sequence. Again, when default settings match the settings desired, that configuration step can be skipped.

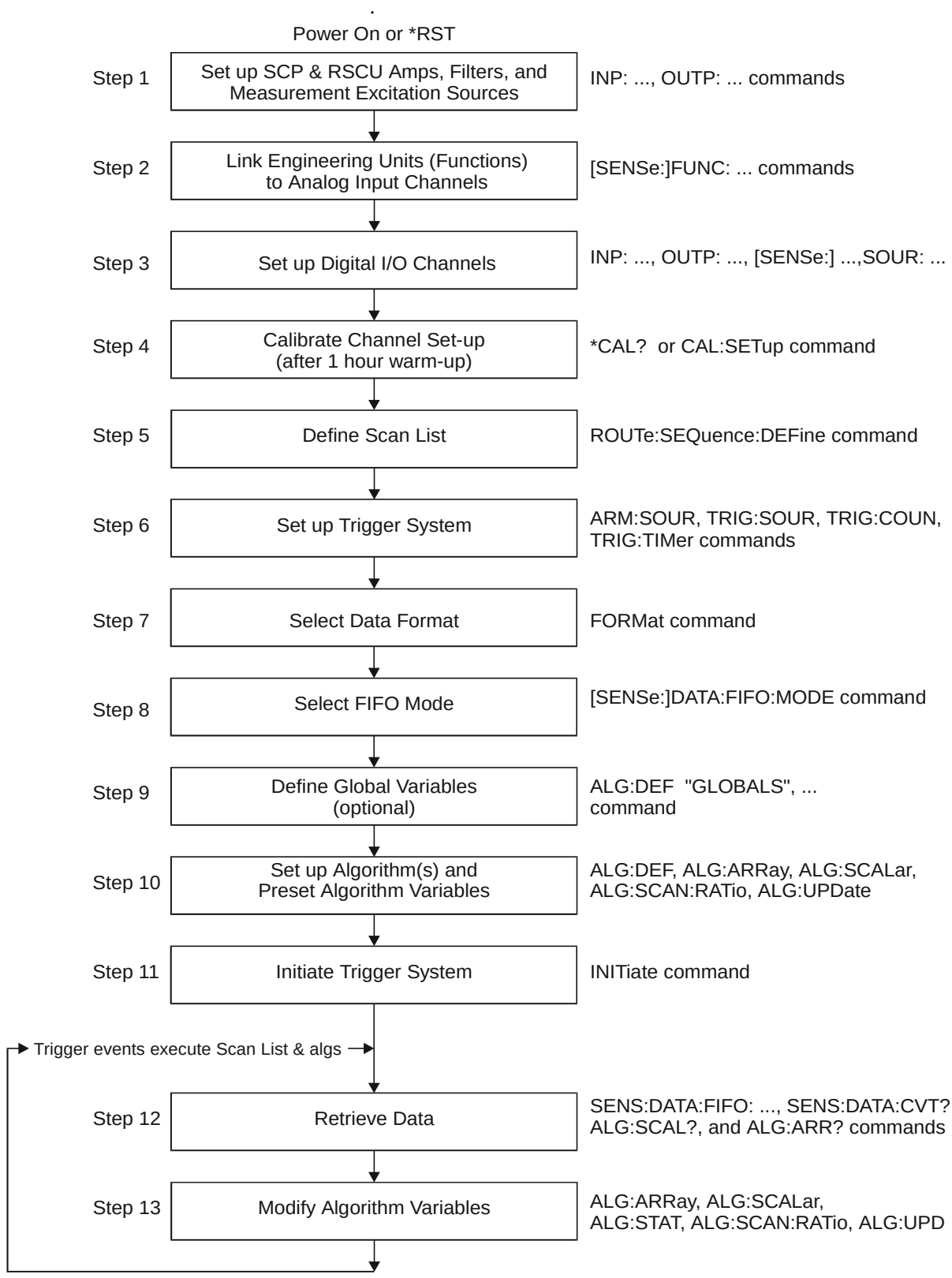
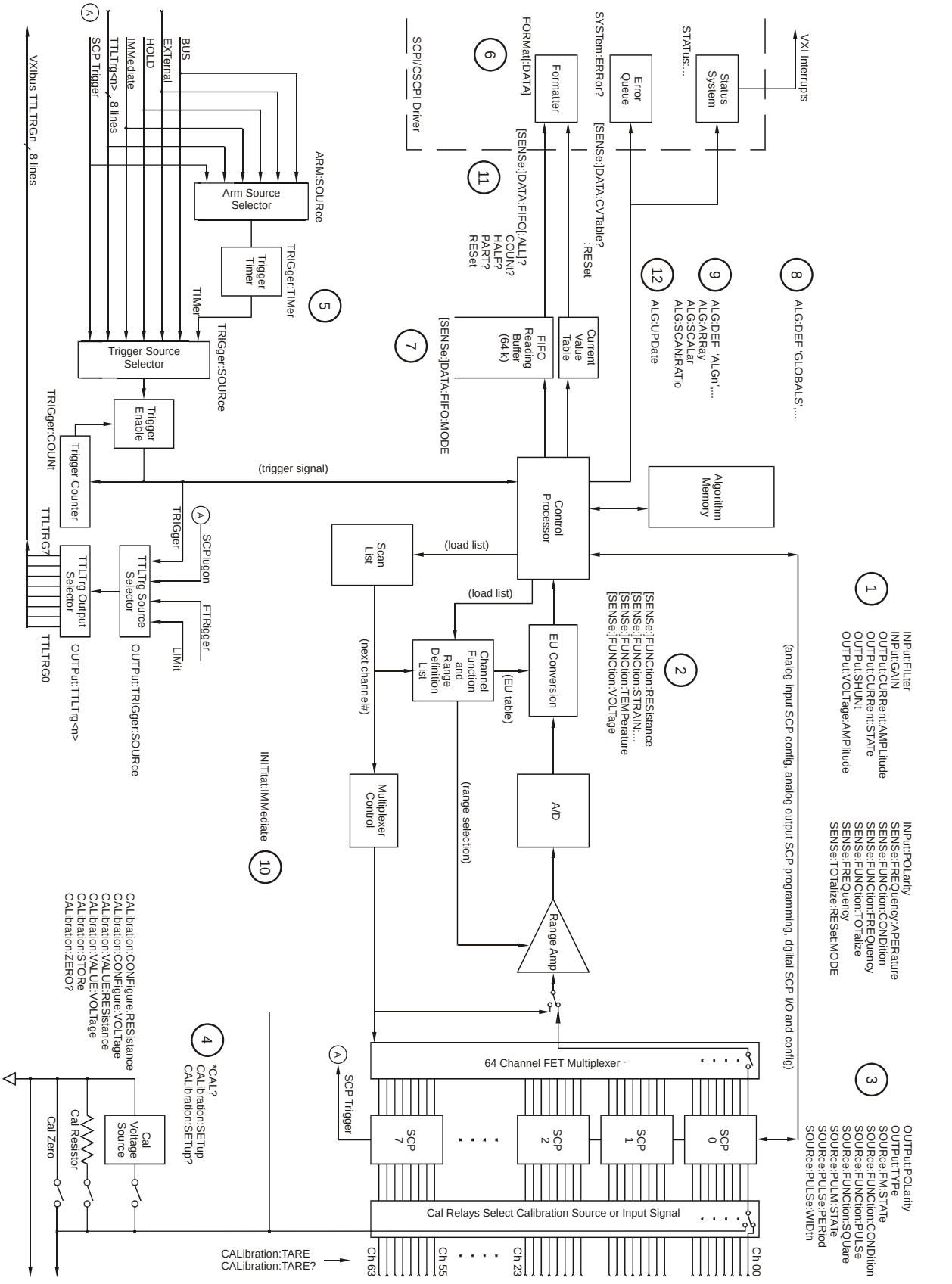


Figure 4-4. Programming Sequence

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 the algorithms need to operate.

Configuring Programmable Analog SCP Parameters

This step applies only to programmable Signal Conditioning Plug-ons such as the VT1503A Programmable Amplifier/Filter SCP, the VT1505A Current Source SCP, the VT1510A Sample and Hold SCP, the VT1511A Transient Strain SCP and Remote Signal Conditioning Units (RSCUs such as the VT1529A/B Remote Strain Conditioning Units). 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 VT1422A & VT1529A/B for Remote Strain Measurement" on page 57. for information on the VT1529A/B'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 (Digital Signal Processor) chip 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*>)

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 VT1503A 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 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

Note: *For the VT1529A/B, these commands affect an 8-channel bank.*

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 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 VT1505A Current Source SCP and VT1518A Resistance Measurement SCP

The Current Source and Resistance Measurement SCPs supply 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*>)
OUTPut:CURRent[:STATe] <*enable*>

- The <*amplitude*> parameter sets the current output level. It is specified in units of amps dc and for the VT1505A/18A SCP can take on the values 30e-6 (or MIN) and 488e-6 (or MAX). Select 488 μA for measuring resistances of less than 8,000 Ω. Select 30 μA for resistances of 8,000 Ω 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 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 VT1532A.

Setting the VT1511A Strain Bridge SCP Excitation Voltage

The VT1511A 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 VT1511A'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 VT1511A.

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 VT1531A.

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: 0.0625, 0.25, 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, send:

```
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 0.0625 volt range, send:

```
SENS:FUNC:VOLT  16,(@116,124)
SENS:FUNC:VOLT  .0625,(@132:147)          must send a command per range
```

or send both commands in a single command message:

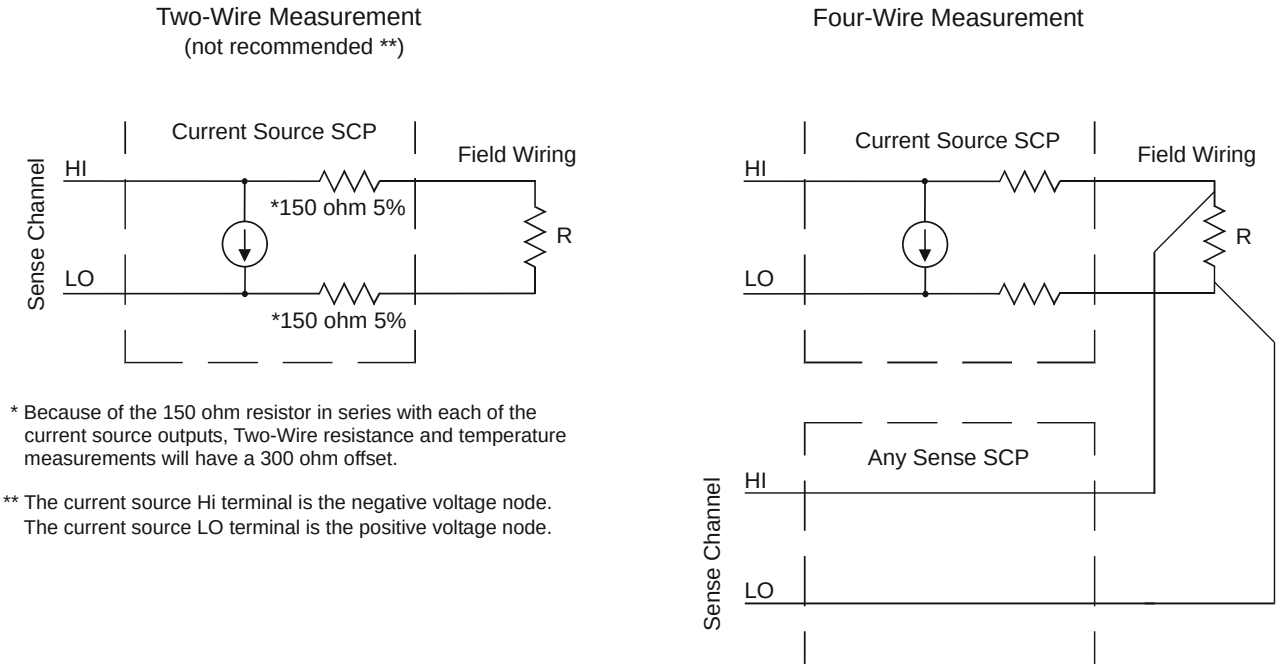
```
SENS:FUNC:VOLT  16,(@116,124);VOLT  .0625,(@123:147)
```

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 the user 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.



* Because of the 150 ohm resistor in series with each of the current source outputs, Two-Wire resistance and temperature measurements will have a 300 ohm offset.

** The current source Hi terminal is the negative voltage node. The current source LO terminal is the positive voltage node.

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 VT1505A SCP are 30e-6 (or MIN) and 488e-6 (or MAX). Select 488 μA for measuring resistances of less than 8,000 Ω. Select 30 μA for resistances of 8,000 Ω 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 Ω 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 30 μA for 8 kΩ or greater resistances
SENS:FUNC:RES 30e-6, (@100:115)
link channels 0 through 15 to resistance EU conversion (8 kΩ or greater)
OUTP:CURR:AMPL 488e-6, (@148,149,150:157)
set 10 channels to output 488 μA for less than 8 kΩ resistances
SENS:FUNC:RES 488e-6, (@116,120,124:132)
link channels 16, 20, and 24 through 32 to resistance EU conversion (less than 8 kΩ)

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 Thermo-Couple)
- 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/°C temperature coefficients respectively).
- For Thermistors; 2250, 5000, or 10000 (the nominal value of these devices at 25 °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 (488 μA)	RTD,85 | 92 and THER,2250
MIN (30 μA)	THER,5000 | 10000

2. The <*sub_type*> parameter 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 Ω thermistors (in this case, paired to current source SCP channels 16 through 31):

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

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

```
OUTP:CURR:AMPL  30e-6,(@148:163)
```
set excite current to 30 μA 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 kΩ thermistor

To set channels 48 through 63 to measure temperature using 100 Ω RTDs with a TC of 0.00385 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 488 μA on current SCP channels 32 through 47
```
SENS:FUNC:TEMP  RTD, 85, (@148:163)
```
link channels 48 through 63 to temperature EU conversion for 100 Ω RTDs with 0.00385 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 are 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/°C temperature coefficients respectively)

  -- For Thermistors; only "5000" (See previous note on page 113)

  -- 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, the concept of the VT1422A Scan List will be introduced. As each algorithm is defined, the VT1422A 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 123). When the algorithms are run, the scan list tells the VT1422A 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 VT1422A 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 VT1422A 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 (VT1506A, VT1507A, VT1511A) or VT1529A/B 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 part of the command syntax. The following table relates the command syntax to bridge type. See the VT1506A, VT1507A, and VT1511A 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 VT1529A/B's internal 120 Ω resistor
:Q350 *	Quarter using VT1529A/B's internal 350 Ω resistor
:User **	Quarter using VT1529A/B's user supplied resistor value

* These choices are only available with the VT1529A/B.

** This choice is only available with VT1529A/B 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. This parameter does not specify channels on the VT1506A/07A Strain Bridge Completion SCPs. It can specify any of the lower four channels of a VT1511A SCP since these channels are the sense channels used to measure the SCPs four bridge completion channels.

- VT1529A/B 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 VT1529A/B 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 VT1529A/B based command sequences, see "Programming the VT1422A & VT1529A/B for Remote Strain Measurement" on page 57.

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 one minute.

See the Command Reference (Chapter 7) and the VT1506A/07A and VT1511A User's Manuals for more information on strain measurements.

Custom EU Conversions

See "Creating and Loading Custom EU Conversion Tables" on page 150.

Linking Output Channels to Functions

Analog outputs are implemented either by a VT1531A Voltage Output SCP or a VT1532A Current Output SCP. Channels where these SCPs are installed are automatically considered outputs. No SOURce:FUNCtion command is required since the VT1531A can only output voltage, while the VT1532A can only output current. The only way to control the output amplitude of these SCPs is through the VT1422A'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 is 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*> parameter can be either NORMal or INVerted. When set to NORM, an input channel with 3 V applied will return a logical "1." When set to INV, a channel with 3 V applied will return a logic "0."

- The <*ch_list*> parameter specifies the channels to configure. The VT1533A has 2 channels of 8 bits each. All 8 bits in a channel take on the configuration specified for the channel. The VT1534A has 8 I/O bits that are individually configured as channels.

To configure the lower 8 bit channel of a VT1533A for inverted polarity:

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

To configure the lower 4 bits of a VT1534A for inverted polarity:

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

Setting Input Function

The VT1533A Digital I/O SCP and the VT1534A Frequency/Totalizer SCP can both input static digital states. The VT1534A 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 a VT1533A in SCP position 4 to input

```
SENS:FUNC:COND  (@132)
```

To set the upper 4 channels (bits) of a VT1534A in SCP pos 2 to input states

```
SENS:FUNC:COND  (@120:123)
```

Frequency Function

The frequency function uses two commands. For more on this VT1534A and VT1538A capability, see the appropriate SCP's User's Manual.

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

To set 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 VT1534A and VT1538A capability see the appropriate SCP's User's Manual.

To set the VT1534A's totalize reset mode
[SENSe:]TOTalize:RESet:MODE INIT | TRIG,(@<*ch_list*>)

To configure VT1534A 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*> parameter 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 VT1533A has 2 channels of 8 bits each. All 8 bits in a channel take on the configuration specified for the channel. The VT1534A and VT1538A have 8 I/O bits that are individually configured as channels.

To configure the higher 8 bit channel of a VT1533A for inverted polarity:

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

To configure the upper 4 bits of a VT1534A for inverted polarity:

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

Setting Output Drive Type

The VT1533A and VT1534A 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 VT1533A has two channels of 8 bits each. All 8 bits in a channel take on the configuration specified for the channel. The VT1534A has 8 I/O bits that are individually configured as channels.

To configure the higher 8-bit channel of a VT1533A for passive pull-up:

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

To configure the upper 4 bits of a VT1534A for active pull-up:

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

Setting Output Functions

The VT1533A Digital I/O SCP, VT1534A, and VT1538A 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 a VT1533A in SCP position 4 to output

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

To set the lower 4 channels (bits) of a VT1534A in SCP pos 2 to output states

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

Variable Width Pulse Per Trigger

This function sets up one or more VT1534A or VT1538A 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 VT1534A/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 ms

```
O134 = 0.001;
```

Variable Width Pulses at Fixed Frequency (PWM)

This function sets up one or more VT1534A/38A 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 VT1534A's first channel at SCP position 4

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

To set pulse period to 0.5 ms (which sets the signal frequency 2 kHz)

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

To set function of VT1534A's first channel in SCP position 4 to PULSE

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

Example algorithm statement to control pulse width to 0.1 ms (20% duty-cycle)

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

Fixed Width Pulses at Variable Frequency (FM)

This function sets up one or more VT1534A/38A 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:

Enable frequency modulation for VT1534A's second channel at SCP position 4

```
SOUR:FM:STATE  ON,(@133)
```

To set pulse width to 0.3333 ms

```
SOUR:PULSE:WIDTH  0.3333e-3,(@133)
```

To set function of VT1534A'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 VT1534A/38A'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 20 kHz

```
O134 = 20e3;
```

Note: *For complete VT1534/38A capabilities, see the SCP's User's Manual.*

Performing Channel Calibration (Important!)

Calibrating the VT1422A

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 VT1422A. *CAL? must be performed in order for the VT1422A to deliver its specified accuracy.

Operation and Restrictions

*CAL? generates calibration correction constants for each analog input channel for offset and gain at all five 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 channel gain or filter settings (INP:FILT or INP:GAIN) are changed, but is still valid for changes of channel function or range (using SENS:FUNC...). Calibration becomes invalid if the SCPs are moved 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. Calibration becomes invalid if the SCPs are moved to different SCP locations.

How to Use *CAL?

When power is turned on to the VT1422A after the SCPs are first installed (and after they are moved), 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 VT1422A'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 one full hour. After it has warmed up, execute the *CAL? query.

What *CAL? Does

The *CAL? query causes the module to calibrate A/D offset, gain, and all channel offsets. This may require several minutes to complete. The actual time it takes the VT1422A to complete *CAL? depends on the mix of SCPs installed. *CAL? literally performs hundreds of internal calibration source measurements for each channel and must allow seventeen 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 are always ready to use. If the configuration just calibrated is to be fairly long-term, the CAL:STORE ADC command should be executed 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 the module warms up, these constants will be relatively accurate.

Re-Execute *CAL? When:

- When channel gain and/or filter cut-off frequency on programmable SCPs are changed (using INPut:GAIN or INPut:FILTer...).
- When SCPs are re-configured to different locations. This is true even if an SCP is replaced 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 419 "Specifications."

NOTE To save time when performing channel calibration on multiple VT1422As in the same mainframe, use the CAL:SETup and CAL:SETup? queries (See "CALibration:SETup" on page 268. for details).

Calibrating Remote Signal Conditioning Units

RSCUs have a local calibration source that the VT1422A can measure directly. This source voltage along with a local short can be fed to each channel on the RSCU. The VT1422A reads the output value of each remote channel when connected to the short and then the calibration voltage source. Using this method, the VT1422A 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, the contents of the analog input Scan List will be defined using the ROUTe:SEQuence:DEFine command. This allows measurements to be made that will be stored to the Current Value Table (CVT) and/or the FIFO buffer without programming or executing any algorithms. While the VT1422A can be used exclusively in this way, these modes of operation can also be combined (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 231 and “ROUTe:SEQuence:DEFine” on page 323.

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 interaction, the scan list should be order in such a way that 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 four VT1529A/Bs connected to two VT1539A 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, the FIFO, or even neither (no reading stored). For more on controlling data destination, See “ROUTe:SEQuence:DEFine” on page 323.

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 VT1422A's memory. It is assumed that the user has some programming experience in C, but since the VT1422A's version of C is limited, just about any experience with a programming language will suffice. See “Creating and Running Algorithms” on page 181 for a complete discussion of the VT1422A'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 VXI*plug&play* users

While the following discussion of algorithm definition is useful for *plug&play* users as regards the coding of an algorithm or global variable definition, the method of generating an algorithm code and actually downloading it to the VT1422A becomes much easier because of *plug&play* VT1422A.exe Soft Front Panel program and hpe1422_downloadAlg(...) *plug&play* driver function.

Using the SFP "Algorithm Panel," an algorithm can be created and tested and then stored to a file. The hpe1422_downloadAlg(...) *plug&play* driver function was created specifically to download algorithms from files into the VT1422A as part of the application program.

Global Variable Definition

Global variables are used when it is necessary 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, an indefinite block header and definitions must be downloaded and it must be terminated by an 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 VEE. It is necessary to edit the VT1422A 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 an algorithm. If less than 256 characters, simply place an algorithm code within string quotes:

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

If the global definition exceeds 256 characters, an indefinite block header and definitions must be downloaded and it must be terminated by an LF/EOI sequence:

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

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

Pre-setting Algorithm Variables

As is evident above, a variable can be initialized to a particular value. However, that value is a one-time initialization. Later program execution may alter the variable and re-issuing an INIT command to re-start program execution will NOT re-initialize that variable. Instead, any scalar or array can be altered using SCPI commands prior to issuing the INIT command or the intrinsic variable, First_loop, can relied upon 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 issuing 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, the following commands can be used:

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

The ALG:SCAL command designates the name of an algorithm, indicates 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 an algorithm, the name of the local array, and a definite length block for assigning the four real number values. As it can be seen, 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. 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 VXI*plug&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 the VT1422A *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 fifteen 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 VT1422A's FIFO can operate in two modes. One mode is for reading FIFO values while the VT1422A 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. The application program must read FIFO values often enough to keep it from overflowing (See "Continuously Reading the FIFO (FIFO mode BLOCK)" on page 135.). 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? query).

- OVERwrite: When the VT1422A 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 137.). This mode is very useful when viewing an algorithm's response to a disturbance is desired.

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 VT1422A. Note that when the Trigger Source selected is TIMer (the default), the remaining sources become Arm Sources. Using ARM:SOUR allows an event to be specified that must occur in order to start the Trigger Timer. The default Arm source is IMMediate (always armed).

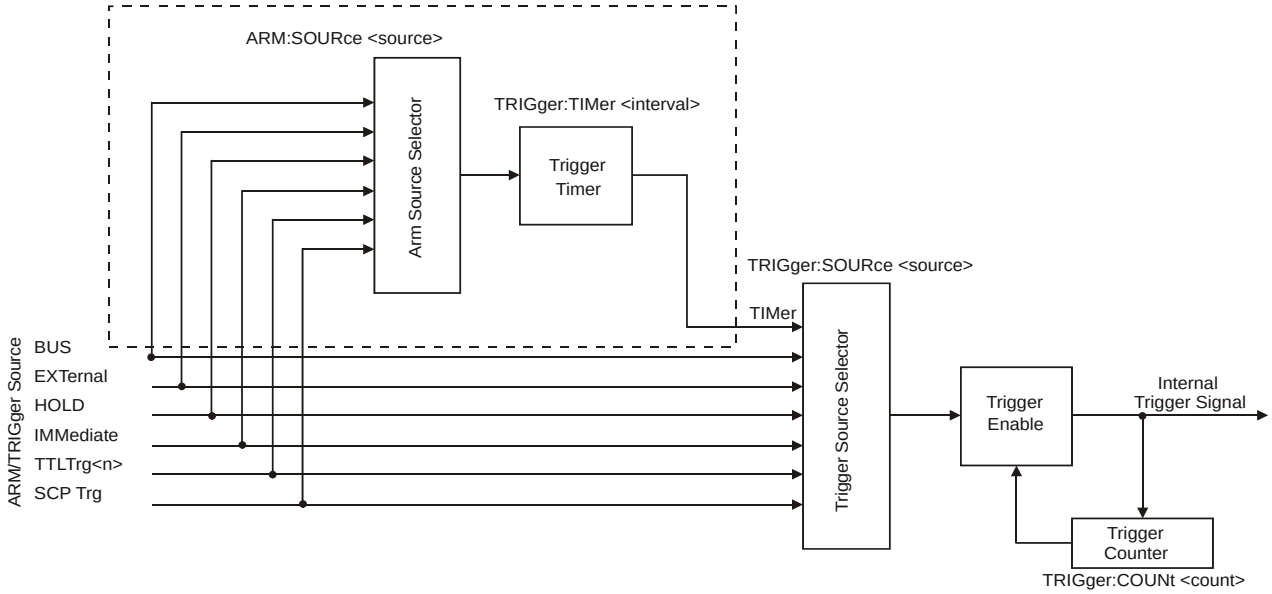


Figure 4-6. Logical Arm and Trigger Model

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 GPIB)
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 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 397 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 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 VT1422A 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. How can this minimum time be determined? After all algorithms are defined, send the ALG:TIME? query with its <*alg_name*> parameter set to 'MAIN'. This causes the VT1422A'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, 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 acquisition scan lists easier (only one scan list of data in the FIFO). For algorithm operation, changing the count to INFinite (can be triggered an unlimited number of times) is probably desired. This setting will be used most often for uninterrupted execution of control algorithms.

To set the trigger count to fifty (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 VT1422A 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 319.

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 VT1422A builds the input Scan List from the input channels referenced when an algorithm is defined with the ALG:DEF command above and from the channels 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? query). 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.

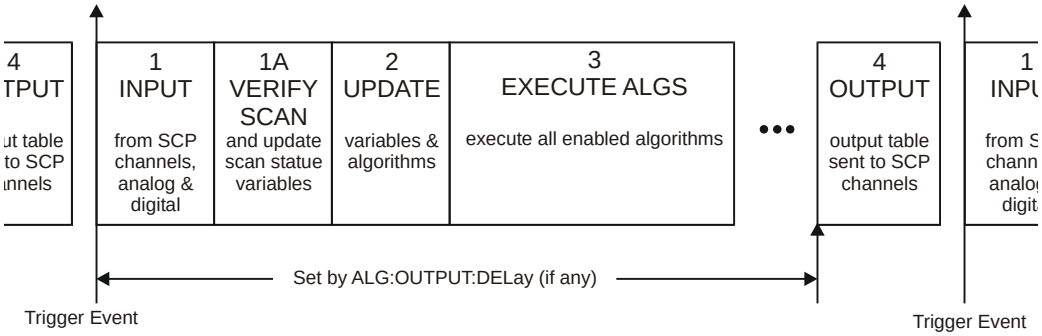


Figure 4-7. Sequence of Loop Operations

The Operating Sequence

The VT1422A has four major operating phases plus one optional phase. 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.

1A. (Remote Runtime Scan Verification): If a Scan Status Variable (S1xx) is referenced in any algorithm, this time is used to check the scan list execution of the Remote Signal Conditioning Unit (RSCU) connected to the channel xx. The S1xx variable will then take on one of three values; 0=normal operation, 1=the RSCU is disconnected and 3= the RSCU scan list was out of synchronization. Each VT1539A SCP has 2 main channels so there are 16 possible scan status variables; S100, S101, S108, S109, S116, S117, S124, S125, S132, S133, S140, S141, S148, S149, S156, and S157. If no S1xx variable is referenced in any algorithm, then phase 1A is not executed.

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 the algorithms at this time.

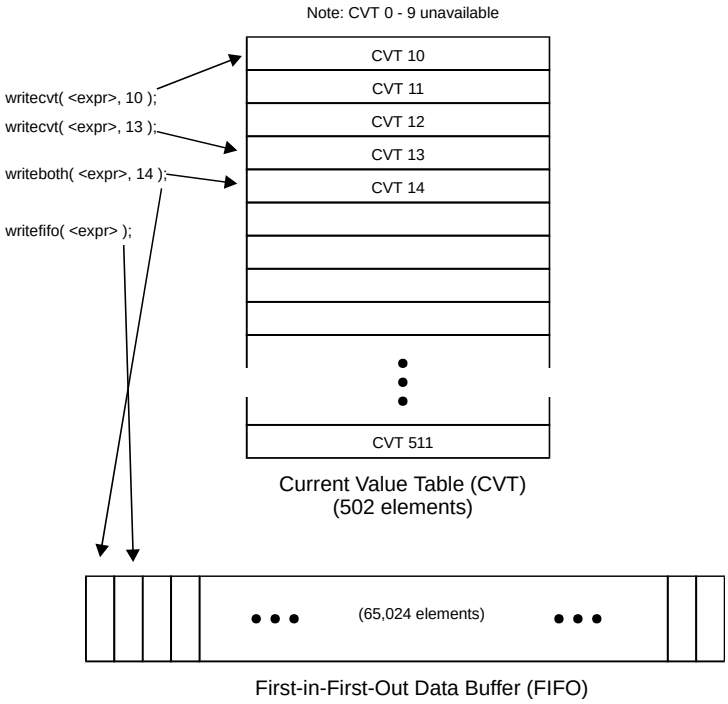
Note Changing algorithm variables requires VT1422A hardware resources that can only be provided during the INPUT and UPDATE phases of the operating cycle. The VT1422A does not update variables during the time between the CALCULATE and OUTPUT phases. Therefore, applications that are intensive in the update 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 VT1422A 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 shown below.



Reading CVT Data

Note that the first ten 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 VXI*plug&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 the VT1422A *plug&play* driver Help file.

The format of data coming 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 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 323. 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 293 for the NaN, +INF, and -INF values for each data format.

Reading FIFO Data

The FIFO can store up to 65,024 real numbers. Each writefifo() or writeboth() cause that expression to be placed into the FIFO. With a FIFO this large, many seconds worth of data can be stored, depending upon the volume of writes and the trigger cycle time. The FIFO's most valuable service is to keep the computer from having to spend too much time acquiring data from the VT1422A. Data is retrieved from the FIFO with the SCPI query DATA:FIFO:PART?<*count*>.

The <*count*> parameter can be a number larger than the FIFO (up to 2.1 billion) if a continuous data read is desired. A query can also be made of how much data is in the FIFO with the DATA:FIFO:COUNT? query.

For VXI*plug&play* Users: See the functions hpe1422_readFifo32_Q andhpe1422_readFifo_Q in the VT1422A *plug&play* driver Help file.

Which FIFO Mode?

Reading the FIFO depends on how the FIFO mode was set in programming step 7 of the “Programming Sequence” on page 105.

Continuously Reading the FIFO (FIFO mode BLOCK)

If the FIFO is going to read while algorithms are running, the FIFO mode must be set 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, the application needs to read the FIFO often enough to keep it from overflowing. A flow diagram is included on the following page to indicate where and when to use the FIFO commands.

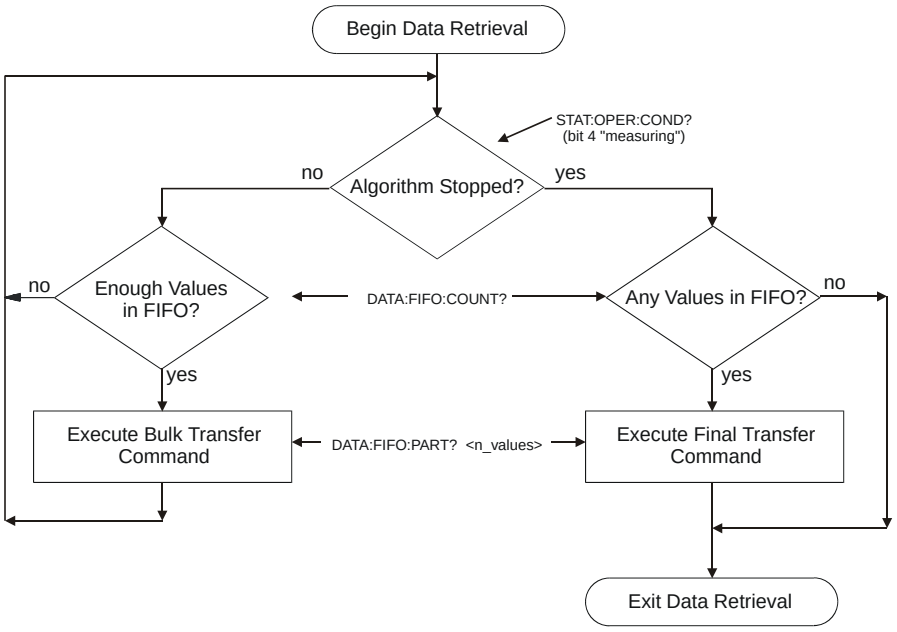


Figure 4-8. Controlling Reading Count

Below is 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                            Sets the minimum block size 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 cannot be read while it is accepting readings from algorithms. Algorithm execution must be stopped before the application program reads the FIFO.

Here is an example command sequence that can be used 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 VXI*plug&play* Users: See the functions hpe1422_sensDataFifoCoun_Q, hpe1422_readFifo32_Q in the VT1422A *plug&play* driver Help file.

Reading Algorithm Variables Directly

To directly read algorithm variables that are not stored in the FIFO or CVT, simply 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, use the ALG:SCALar? query. To read an entire array, use the ALG:ARRay? query. 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'             read global variable
ALG:SCAL? 'alg1','my_array[6]'           read single element of array
ALG:SCAL? 'alg1','S108'                  read scan status variable
ALG:ARR? 'alg2','my_other_array'         read all elements of array
```

The ALG:ARR? response data will consist of a block header and real-64 data bytes. For example, if my_other_array was ten 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 VT1422A is terminated with the GPIB EOI signal.

For VXI*plug&play* Users: See the functions hpe1422_algScal_Q (for scalar variables) and hpe1422_algArray_Q (for array variables) in the VT1422A *plug&play* driver Help file.

Modifying Running Algorithm Variables

Updating 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 132). The UPDate:WINDow <*num_updates*> command can be used to specify how many updates need to be performed during phase 2 (UPDATE phase) and assigns a constant window of time to accomplish all of the updates that will be made. The default value for <*num_updates*> is 20. Fewer updates (shorter window) means slightly faster loop execution times. Each update takes approximately 1.4 μs.

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 VT1533A at channel 133
                           changes state
```

For VXI*plug&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 the VT1422A *plug&play* driver Help file.

Enabling and Disabling Algorithms

An algorithm is enabled by default when it is defined. To enable or disable an algorithm, use the ALG:STATe <*alg_name*>, ON | OFF command. When an individual algorithm is enabled, it executes when the module is triggered. When disabled, the algorithm does not execute.

NOTE

The command ALG:STATE <*alg_name*>, ON | OFF does not take effect until an ALG:UPDATE command is received. This allows multiple ALG:STATE commands to be sent with a synchronized 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
```

VXI*plug&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*>. To execute 'ALG3' only every twenty triggers, an ALG:SCAN:RATIO 'ALG3',20 command can be sent 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 VT1539A 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 VXI*plug&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 VT1539A 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 previous 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 returned
```

Using the Status System

The VT1422A's Status System allows for the quick polling a single register (the Status Byte) to see if any internal conditions require 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.

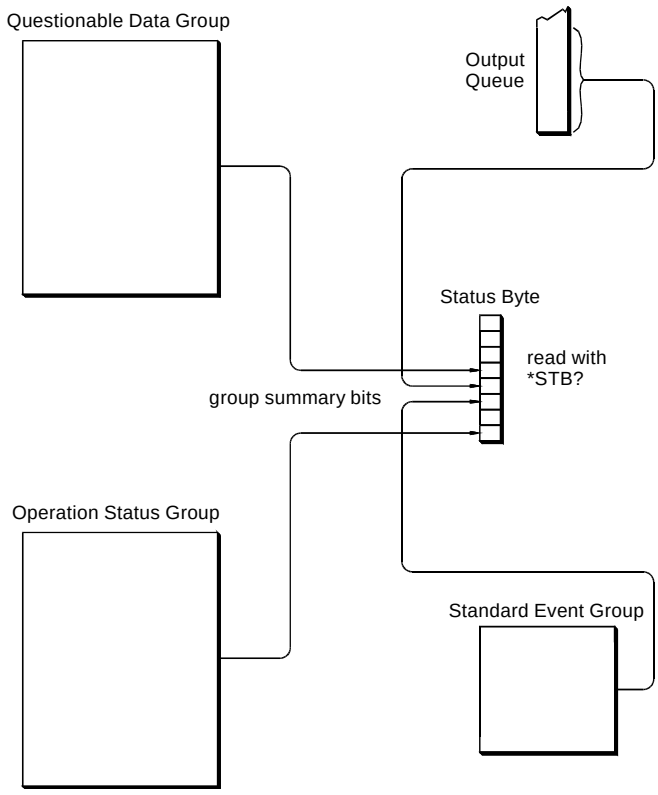


Figure 4-9. Simplified Status System Diagram

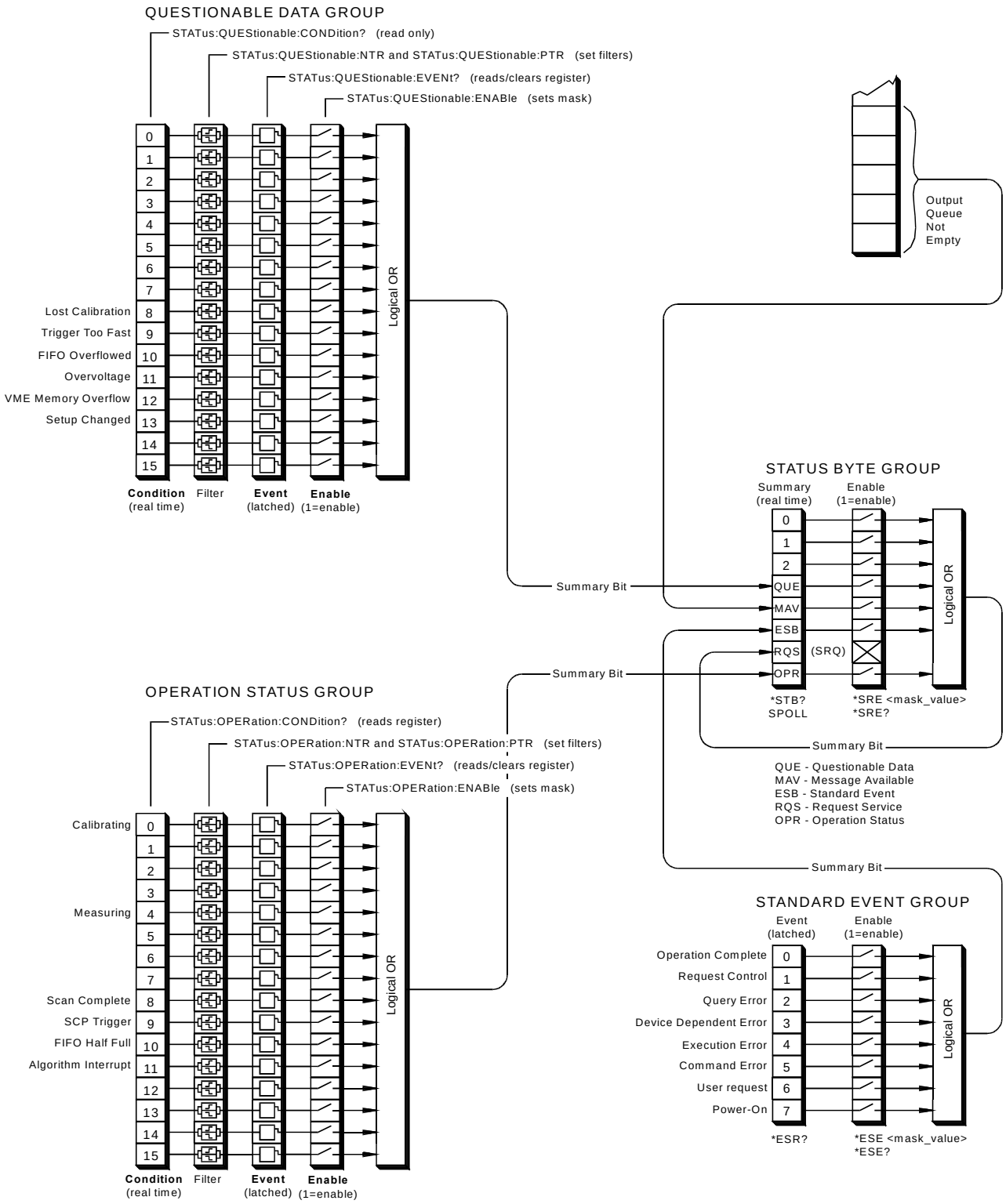


Figure 4-10. VT1422A Status System

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? query 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 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 VT1422A
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 453.
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 VT1422A.
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 having the module report via the Status System when it has completed executing *CAL? is desired. 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, 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

In Figure 4-10, 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

To have only the "FIFO Overflowed" condition reported by the QUE bit (bit 3) of the Status Byte, execute:

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

To have only the "FIFO Overflowed" and "Setup Changed" conditions reported, execute:

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

Operation Status Group Examples

To have only the "FIFO Half Full" condition reported by the OPR bit (bit 7) of the Status Byte, execute:

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

To have only the "FIFO Half Full" and "Scan Complete" conditions reported, execute:

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

Standard Event Group Examples

To have only the "Query Error", "Execution Error," and "Command Error" conditions reported by the ESB bit (bit 5) of the Status Byte, 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, first read the Status Byte using the *STB? query. 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. The Status Byte bit values are interpreted and act as follows:

Bit 3 (QUE)
bit value 8_{10} Read the Questionable Data Group's Event Register using the STAT:QUES:EVENT? query. 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? query to read the error(s).

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

Bit 5 (ESB)
bit value 32_{10} Read the Standard Event Group's Event Register using the *ESR? query. 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? query to read these.

Bit 7 (OPR)
bit value 128_{10} Read the Operation Status Group's Event Register using the STAT:OPER:EVENT? query. 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) the Status Byte can polled and the RQS bit checked to determine if there are any status conditions which need attention. In this way, the RQS bit is like the GPIB's SRQ (Service Request) line. The difference is that while executing a GPIB serial poll (SPOLL) releases the SRQ line, executing the *STB? query does not clear the RQS bit in the Status Byte. The Event Register of the group whose summary bit is causing the RQS must be read.

Reading Status Groups Directly

It may be desirable 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*

VT1422A Background Operation

The VT1422A 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 VT1422A 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 VT1422A 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 VT1422A 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 VT1422A 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 VT1422A 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 VT1422A, the VT1422A interrupts can be used.

The VT1422A 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 VT1422A.
- Scan complete. The VT1422A 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 VT1422A 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 by 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 VT1422A 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 VT1422A.

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 previous 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 VT1422A provides for loading custom EU conversion tables. This allows for the on-board conversion of transducers not otherwise supported by the VT1422A.

Standard EU Operation

The EU conversion tables built into the VT1422A are stored in a "library" in the module's non-volatile Flash Memory. When a specific channel is linked 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, 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 [SENSe:]FUNC:CUST is executed, but the command allows the A/D range for these channels to be specified.

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 VT1422A uses two types of EU conversion tables: linear and piecewise. The linear table describes the transducer's response slope and offset (y=m*x*+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 VT1422A 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 114.

Custom Reference Temperature EU Conversions

The VT1422A 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 114.

Creating Conversion Tables

Contact a VXI Technology System Engineer for more information on creating Custom Engineering Unit Conversion.

Loading Custom EU Tables

There is a specific location in the VT1422A’s memory for each channel’s EU Conversion table. When standard EU conversions are specified, the VT1422A loads these locations with EU conversion tables copied from its non-volatile flash memory. For Custom EU conversions, these table values must be loaded using either of two SCPI commands.

Loading Tables for Linear Conversions

The DIAGnostic:CUSTom:MXB <*slope*>,<*offset*>, (@<*ch_list*>) command sends the <*slope*> and <*offset*> parameters that allow the driver to calculate and download a custom linear Engineering Unit Conversion table to the VT1422A 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:

```
The program puts table constants into array <table_block>
DIAG:CUST:MXB  2.2,19,(@132:163)      send table for chs 32-63
                                      to VT1422A
SENS:FUNC:CUST  1,(@132:163)          link custom EU with chs 32-63
                                      and set the 1 V 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 VT1422A for each channel specified.

- The <*table_block*> parameter 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 VXI*plug&play* function hpe1422_sendBlockInt16(...) adds the header automatically. Contact a VXI Technology System Engineer for more information on creating piecewise custom EU tables.

- The <*table_range*> parameter specifies the range of input voltage that the table covers (from -<*table_range*> to +<*table_range*>). The value specified must be within 5% of: 0.015625 | 0.03125 | 0.0625 | 0.125 | 0.25 | 0.5 | 1 | 2 | 4 | 8 | 16 | 32 | 64.

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

Usage Example:

```
The program puts table constants into array table_block
DIAG:CUST:PIEC  <table_block>,1,(@124:131)   send table for chs 24-31 to
                                             VT1422A
SENS:FUNC:CUST  1,(@124:131)                 link custom EU with chs 24-31
                                             and set the 1 V 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 a VT1422A 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 VT1422A can compensate for offsets in the system's field wiring. Apply shorts to channels at the Unit-Under-Test (UUT) end of the 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

- CAL:TARE must not used on field wiring that is made up of thermocouple wire. The voltage that a thermocouple wire pair generates cannot be removed by introducing a short anywhere between its junction and its connection to an isothermal panel (either the VT1422A'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 a thermocouple wire pair is CAL:TARE'd inadvertently, see "Resetting CAL:TARE" on page 154.
- CAL:TARE should be used to compensate wiring offsets (copper wire, not thermocouple wire) between the VT1422A 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:TARE'd" 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 154).

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 several minutes to execute.

The tare calibration constants created during CAL:TARE are stored in and are usable from the instrument's RAM. To store the tare constants in non-volatile flash memory, execute the CAL:STORE TARE command.

NOTE The VT1422A's Flash Memory has a finite lifetime of approximately 10,000 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

To "undo" the CAL:TARE operation, 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	0.40104	0.20009	0.04970
4	0.82101	0.10101	0.05007	0.01220
1	0.23061	0.02721	0.01317	0.00297
0.25	0.07581	0.00786	0.00349	0.00055
0.0625	0.03792	0.00312	0.00112	n/a

Changing Gains or Filters

To change a channel's SCP setup after a CAL:TARE operation, a *CAL? operation must be performed to generate new DAC constants and reset the "range floor" for the stored tare value. The tare capability of the range/gain setup that the channel is being changed to must also be considered. 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 7-1 on page 294 for more on values returned for Overload readings.

Unexpected Channel Offsets or Overloads

This can occur when the VT1422A'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, then the tare constants in Flash memory by executing CAL:STORE TARE can be reset.

Detecting Open Transducers

Most of the VT1422A'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.

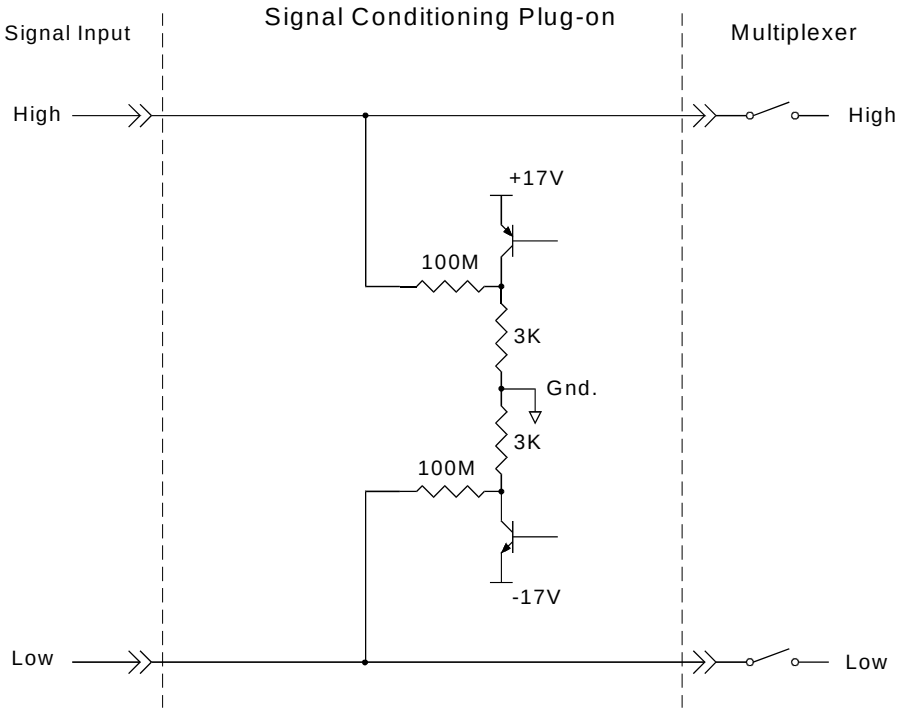


Figure 4-11. Simplified Open Transducer Detect Circuit

NOTES

1. When OTD is enabled, the inputs have up to 0.2 μA injected into them. If this current will adversely affect the measurement, but checking for open transducers is still desired, enable OTD, run the algorithms, check analog input variables for measurement values that indicate an open transducer, then disable OTD and run the algorithms without it. The VT1422A's accuracy specifications apply only when OTD is off.
2. When a channel's SCP filtering is enabled, allow fifteen 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 an input signal can be difficult for the VT1422A to auto range correctly. The module completes the range selection based on the input signal about 6 μs before the actual measurement is made on that channel. If, during that period, the signal becomes greater than the selected range can handle, the module will return an overflow reading (±INFinity).

To fix this problem, use the DIAGnostic:FLOor <*range*>,(@<*ch_list*>) command. Include the problem channel(s) in <*ch_list*> and specify the lowest range desired for 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 more than one range floor needs to specified 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 determine if a system presents any of these problems and how best to eliminate them or reduce their effect.

Background

While the VT1422A 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 VT1422A's Analog-to-Digital converter and its Range Amplifier, this situation is the most difficult to measure. For example, let's 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 0.0625 volt range. On this range, the resolution is now 1.91 μV 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, it has been 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 a VT1501A 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 gigaohm 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 VT1422A's Range Amp and A/D. The VT1503A, 8A, 9A, 10A, 12A, and 14A through 17A SCPs all provide this capability.

Checking for Problems

This method can be used to quickly determine if any of the system's channels need more settling time by simply applying some settling time to every channel. Use this procedure:

1. First, run the 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 the system and look primarily for low level channel measurements (like thermocouples) whose dc value changes somewhat. If channels are found that respond to this increase in sample period, it may also be observed that these channels 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 some improvement is seen, increase the sample period again and perform another test. When the sample period is increased and no improvement is seen, the maximum settling delay that any single channel requires has been found.
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 the system scans fast enough with the increased sample period, the problem is solved. The system is only running as fast as the slowest channel allows, but, if its fast enough, that's okay. If, on the other hand, getting quality readings has slowed the scan rate too much, there are two other methods that will, either separately or in combination, have the 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 four 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 VT1422A are available with gains of 0.5, 8, 16, 64, and 512. Now return to the 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 0.0625 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 μV 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, a gain of 64 can be used and set the Range Amplifier to the 4 volt range. At this setting, the A/D resolution for one LSB drops to about 122 μV 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 problems are seen on only a few channels, the SENS:CHAN:SETTLING <*num_samples*>,(@<*ch_list*>) command can be used to add extra settling time for just these problem channels. What SENS:CHAN:SETTLING does is it instructs the VT1422A 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 three 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
Advanced Programming with the VT1529B

About This Chapter

This chapter describes the advanced capability that is possible with the VT1529B Remote Strain Conditioning Unit (RSCU). This includes being able to scan thermocouples and high-level dc voltages in addition to strain measurements. Also, the strain bridge excitation voltage can be included as part of the scan list and have the VXI*plug&play* driver use the updated excitation voltage for a more accurate engineering units conversion. Using the advanced features changes some of the VT1422A behavior and this chapter describes these changes.

Additional Capabilities of the VT1529B

With the VT1529B/39A, only strain measurements can be made. For other measurements, such as 1-16 V dc or thermocouple temperature measurements, other SCPs (for example, the VT1511A Direct Input SCP or the VT1509A Fixed Filter and Amplifier SCP) must be used. The disadvantage of this solution is that a direct connection must be made between the VT1422A and the sensor being measured and this can mean long lengths of expensive shielded, twisted pair cables for each sensor, when it would probably be prefered to make these measurements using the VT1529A, with its 32:1 multiplexing and single cable back to the VT1422A.

Additionally, all VT1529A strain excitation voltages are measured ***once***, when the INITiate:IMMediate command is executed. If the excitation supply varies over time, it is typically preferred that the excitation voltage be measured periodically and the new values be used in the strain computation.

The VT1529B provides the flexibility to make all of the measurements normally needed for structural test applications without the need for other SCPs: strain, temperature and dc voltage (from string pots or dc-dc LVDT). The VT1529B also allows channels to be dedicated to the strain excitation voltage and includes these channels in the scan list so that the excitation voltage is measured during every scan. The updated reference measurements can be used in the strain engineering units conversions, yielding more accurate data.

With the VT1529B option 002 VT1586A-to-VT1529B cable, the VT1586A Rack-Mount Terminal Panel can be connected to the VT1529B yielding an accurate and dependable isothermal reference needed for thermocouples.

Finally, the VT1529B is 100% backwards compatible with the VT1529A so programs written for the VT1529A will work unchanged with the VT1529B. This chapter ***only*** describes the new capability of the VT1529B.

Note The new capability is available ***only*** through the VXI*plug&play* driver and ***only*** with direct VXI interfaces (VXI embedded PC, MXI-2, or Agilent/HP E8491B IEEE-1394 PC Link to VXI). The new capability is ***not*** available using GPIB and the Agilent/HP E1406A VXI Command Module.

Changes to the Use Model

There are some important changes in the way the VT1422A works with the VT1529B when these new features are used. Note that these changes affect only the new features; if the new features are not used or desired, the VT1422A operates exactly as described in the previous chapters.

The first thing to know is that, as noted above, the VXI*plug&play* driver must be used in order to use the new features and that these features are not available with the Agilent/HP 1406A VXI Command Module. A direct VXI interface must also be used such as a VXI embedded PC, MXI-2, or the Agilent/HP E8491B interface.

Engineering Units Conversion Done in VXI*plug&play* Driver

All of the new functionality is implemented by the VXI*plug&play* driver. The on-board DSP code of the VT1422A has not been changed and this has a significant impact on the use model. In particular, on channels programmed using the new features, engineering units conversions are performed by the driver, not by the on-board DSP. This means that all FIFO and CVT data for these channels will be *voltages*, not engineering units.

Additionally, if algorithms have been downloaded into the VT1422A, the channel values provided to the algorithms will be *voltages*, not engineering units. If the algorithms require engineering units, either the engineering units conversions in the algorithm code must be performed or the algorithm must be modified to accept voltages.

Because the EU conversions for temperature and strain are performed in the host computer, not by the on-board DSP, a fast computer may be needed in order to keep up with multiple VT1529B's scanning at full speed. These conversions use the full strain and ITS-90 thermocouple conversion equations, which offers more accuracy than the on-board DSP approximations, but at the cost of being slower to compute. A PC with a Pentium® III 800 MHz processor or faster is recommended.

Note Channels programmed using the old commands are unaffected by these changes. The on-board DSP will perform the engineering units conversion. Only channels programmed using the new commands are affected.

Note Channels programmed with the old and new commands on a VT1529B can be intermixed.

Must Count writefifo Calls in Algorithms

Because the VXI*plug&play* driver does the engineering units conversion for all channels programmed with the new commands, the driver needs to understand the order of data in the FIFO, matching the FIFO data with the appropriate channel's EU. This means that the driver must be made aware of any extra data written to the FIFO by any algorithms that has been downloaded. All *writefifo()* calls need to be counted in an algorithm and this count should be provided to the driver so that proper synchronization between the FIFO data and the scan list is maintained. This count is the third parameter to the *hpe1422_readFifoPost_Q()* driver call.

New SCPI Commands

In the VXI*plug&play* driver, there are some new calls. There are also new SCPI commands that support the new functionality of the VT1529B. Some of these commands are completely new while other commands are similar to the existing SCPI commands except that they have the :POST keyword and/or an extra parameter added to them. The new SCPI commands are used with the "pass-through" functions in the VXI*plug&play* driver. For complete information on the new SCPI commands, see the Command Reference Section (Chapter 7).

Note See the VT1422A VXI*plug&play* driver's help file for details on the "pass-through" functions.

For example, the existing SCPI commands to measure thermocouples with a VT1508A SCP and the VT1422A are:

SENS:REF THER,5000,(@108)	*Sense the reference temperature on channel 108 using a 5k thermistor*
SENS:FUNC:TEMP TC,J,(@109:111)	*Indicates channels 109 through 111 are type J thermocouples*
SENS:REF:CHAN (@108),(@109:111)	*Specifies channel 108 as the isothermal reference for channels 109 through 111*
ROUT:SEQ:DEF (@108:111)	*Sets scan list to include the reference and the thermocouples*
INIT	*Start scan*

To measure thermocouples using the VT1529B and VT1586A using the new commands:

SENS:REF:POST THER,5000,(@10000),(@10001)	*Specifies that the reference thermistor voltage is on channel 10001 and its excitation voltage is on channel 10000*
SENS:FUNC:TEMP:POST TC,J,(@10003:10005)	*Indicates channels 10003 through 10005 are type J thermocouples*
SENS:REF:CHAN:POST (@10001),(@10003:10005)	*Specifies channel 10001 as the isothermal reference thermistor for channels 10003 through 10005*
ROUT:SEQ:DEF (@10003:10005,10001,10000)	*Sets scan list to include the thermocouples, the reference thermistor and excitation voltage*
INIT	*Start scan*

Similarly, the VXI*plug&play* call to read the FIFO data changes from:

hpe1422_readFifo_Q(vi, 1000)	*Read 1000 readings*

to

hpe1422_readFifoPost_Q(vi, 2, 0, 65024, f64_array)	*Read 2 scans of data, post-convert to EU, no extra data in FIFO from algorithms*

Strain Measurements

With the VT1529A, the excitation voltages are measured once prior to the start of scanning (INITiate:IMMediate command). The measured value is then sent to the DSP for use in the on-the-fly EU conversions. Therefore, for long tests, for the best possible accuracy, the excitation supply must be both long-term and short-term stable.

With the VT1529B, the excitation voltage can be measured during each scan and the VXI*plug&play* driver can use updated measurements for the conversions. This provides better accuracy and uses a less expensive excitation supply.

Field Wiring for Excitation Measurements

To take advantage of the on-the-fly excitation measurement feature means that at least one channel of each VT1529B will need to dedicated to the excitation voltage measurement. Any of the 32 channels can be used for this purpose. If the application requires more frequent measurements than once per scan, multiple channels can be dedicated and a separate excitation measurement channel for each strain measurement channel can even be used. The excitation measurement channel must be specified for each strain measurement channel using SENS:FUNC:STR:<*bridge_type*>:POST.

For the dedicated excitation measurement channel, simply leave the RJ-45 ports for the channel unconnected. (Putting a short between the strain gage Sense + and Sense - inputs, however, may be desired. See "Settling Time Considerations" on page 178 for a discussion of why this may be desired.) This works because there is an internal 10 kΩ resistor between the Bridge Excitation input on the front of the VT1529B and the Excitation Sense + and - inputs for each channel in the 8-channel bank that each Bridge Excitation input provides power to. Thus, making an excitation voltage measurement on an unconnected channel input will measure the voltage supplied to the Bridge Excitation input.

Note For systems with multiple VT1529Bs, the excitation measurements must be made on the same VT1529B as the strain gage channels that use that excitation voltage. This means that for a system with two VT1529Bs, the excitation voltage cannot be measured on only one VT1529B with that measurement being applied to the strain gage measurements on both VT1529Bs.

Note The wiring for the strain gage channels is unchanged.

Strain Measurement Command Sequence

The steps for making strain measurements with the new VT1529B functionality may already be familiar. The unstrained voltage and gage factor for the strain channels need to be provided. The excitation channel for each of the strain channels must also specified as well as the channels to be scanned. One difference is that the excitation channel(s) must be included in the scan list. Figure 5-1 shows the sequence of VXI*plug&play* driver calls to configure and scan strain channels using the new functionality.

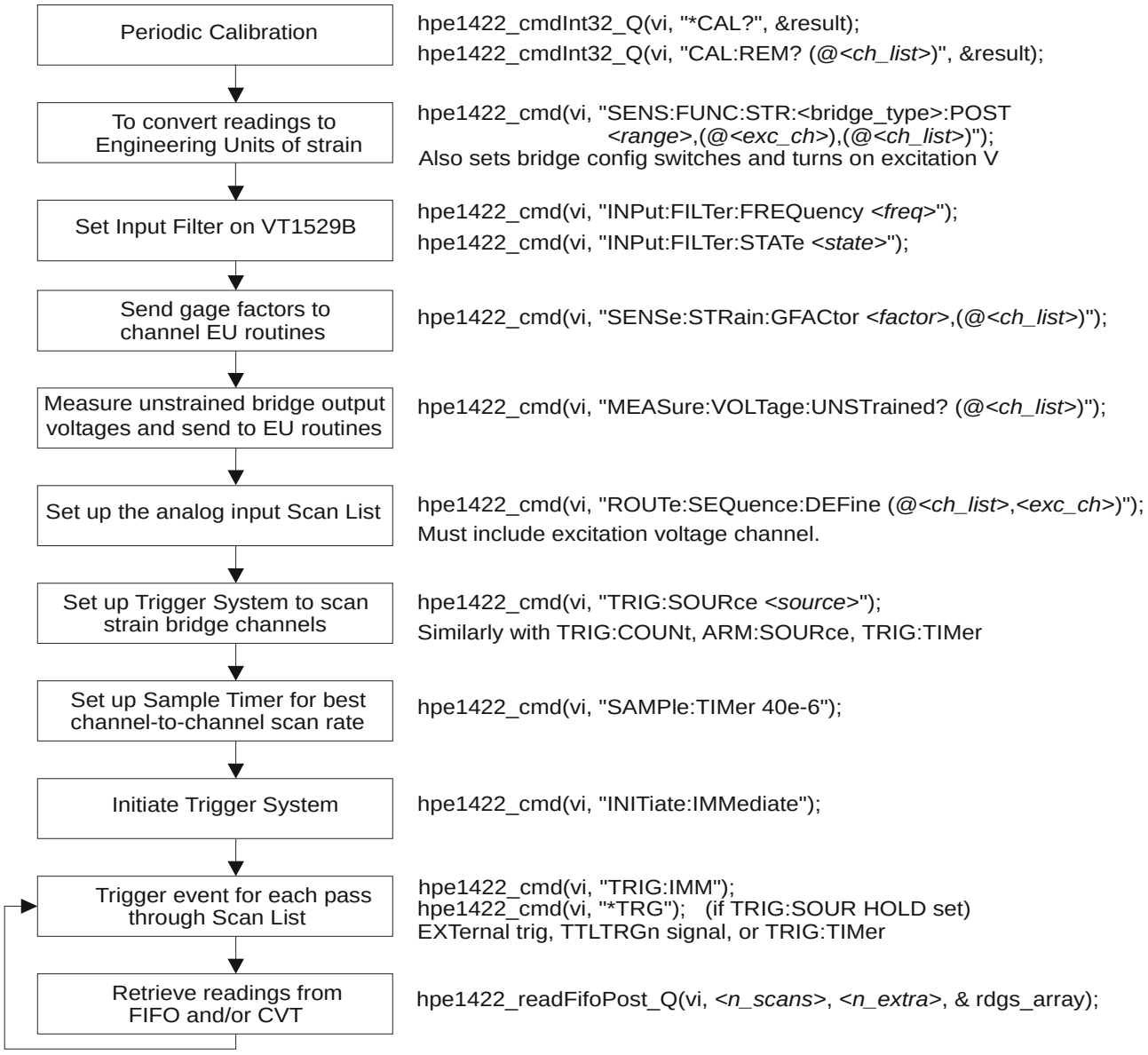


Figure 5-1. Sequence for VT1529B Strain Conversion

Strain Conversion Sequence

Below is an example VXI*plug&play* command sequence. Note that this is not executable; it has been simplified for easier reading. The C++ example source file (*eupost.cpp*) is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

```
/* set function to strain; note ch 10006 is excitation voltage */
errStatus = hpe1422_cmd(vi,
                    "sens:func:str:hben:post auto,(@10006),(@10000:10005)");

/* optionally set VT1529B input filters (2, 10 or 100 Hz) */
errStatus = hpe1422_cmd(vi, "inp:filt:freq 10,(@10000:10005)");
errStatus = hpe1422_cmd(vi, "inp:filt:stat on,(@10000:10005)");

/* set gage factors for EU conversion routines */
errStatus = hpe1422_cmd(vi,"sens:str:gfac 2,(@10000:10005)");

/* measure the unstrained bridge voltage at each bridge. The values go to the
   channel EU conversion as well as the FIFO. */
errStatus = hpe1422_cmdInt16_Q(vi,"meas:volt:unst? (@10000:10005)",&result16);

/* set up the scan list to include the excitation and strain channels */
errStatus = hpe1422_cmd(vi,"rout:seq:def (@10000:10006)");

/* 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(vi,"trig:coun 1");     /* *RST default */
errStatus = hpe1422_cmd(vi,"trig:sour tim");   /* *RST default */
errStatus = hpe1422_cmd(vi,"arm:sour imm");    /* *RST default */

/* set up the sample timer. This controls the channel-to-channel scan
   rate and can be important when channels need more that the default
   40 us sample time. */
errStatus = hpe1422_cmd(vi,"samp:tim 40e-6");  /* *RST default */

/* INITiate the trigger system to execute a measurement scan */
errStatus = hpe1422_cmd(vi,"init:imm");

/* retrieve one scan's data from the FIFO */
errStatus = hpe1422_readFifoPost_Q(vi, 1, 0, 65024, f64_array);
```

Alternate Method of Computing Strain

As stated above, to use the additional features of the VT1529B, the EU conversions are normally done in the VXI*plug&play* driver, not by the on-board DSP. For channels programmed using the new capability, all measurement data in the FIFO, CVT, and algorithms is voltage, not strain. For full- and half-bridge strain channels, there is an alternate method of doing the EU conversions if having the strain engineering units in the algorithms is needed while still using the most recent value of the excitation voltage for more accurate data. This method involves making strain measurements using the DSP while also scanning the excitation voltage. Because the full- and half-bridge strain conversions are linear with respect to the excitation voltage, an algorithm can be created that uses the ratio of the initial and current excitation voltages to correct the DSP-computed value to get a more accurate value.

Note This method cannot be used for quarter-bridge strain channels because the EU conversion for quarter bridge is non-linear with respect to the excitation voltage. If a quarter-bridge strain measurements is required in an algorithm, the full EU conversion must be done in the algorithm.

Here are the steps to use this method:

1. Configure the channels as one would for the VT1529A. The following simplified command sequence assumes that the strain channels are 10000 through 10002 and that the common excitation voltage for these channels is monitored on channel 10006. It also assumes that the excitation voltage supply is short-term stable, so that the supply variation is negligible between the time the first and last channels are measured.

MEAS:VOLT:EXC? (@10000:10002)	*Measure excitation on strain channels - the FIFO data needs to read and saved for later*
MEAS:VOLT:UNST? (@10000:10002)	*Measure unstrained voltages*
SENS:STR:GFAC 2.2,(@10000:10002)	*Specify gage factor*
SENS:FUNC:STR:FBEN (@10000:10002)	*Setup for full-bridge strain on channels (DSP will do EU)*

2. To have only the corrected value appear in the FIFO and CVT, disable sending the uncorrected values to the FIFO and CVT using the ROUT:DEF command:

ROUT:DEF (@(0(10000:10002),3(10006)	*Disable sending of uncorrected strain data to FIFO and CVT; enable sending to both for excitation data*

Alternatively, simply omit the channels for the uncorrected values from the scan list – the VT1422A driver will automatically add them to the scan list because they appear in the algorithm and won't send them to the FIFO or CVT.

3. Next, create a custom linear EU conversion for the excitation voltage channel. This EU will create the ratio of the newly measured value divided by the original value. The first channel's pre-INITiate excitation value measured by the MEAS:VOLT:EXC? query above will be used to create this EU. For this example, assume that this initial value was 5.60 V.

DIAG:CUST:MXB 1.0/5.60,0.0,(@10006)	*Create custom linear EU to compute ratio of current excitation voltage to initial value*
SENS:FUNC:CUST:HVOL (@10006)	*Enables custom EU on excitation channel*

4. Use the ratio inside the algorithm to adjust the DSP-computed value. After computing the corrected reading, the *writefifo*(), *writecvt()* or *writeboth()* algorithm calls can be used to send the corrected data to the FIFO and/or CVT. This is a code segment from the algorithm:

```
{
  float ratio;
  float chan0, chan1, chan2;

  /* get the ratio of new to initial excitation voltages */
  ratio = I10006;

  /* read and correct channel data */
  chan0 = I10000 * ratio;
  chan1 = I10001 * ratio;
  chan2 = I10002 * ratio;

  /* write corrected values to FIFO and the normal
     position in CVT for this channel */
  writeboth(chan0, 10);
  writeboth(chan1, 11);
  writeboth(chan2, 12);
  writeboth(chan3, 13);
} /* end of code segment */
```

5. Download the algorithm and start scanning.

6. Read the data with SENS:DATA:FIFO? This data will be corrected strain readings. Note that the data order in the FIFO will be channel 10006, 10000, 10001, 10002.

Note Doing these corrections in an algorithm will increase the duration of the Execute Algorithm phase of the VT1422A operational cycle (see Figure 6-2 on page 195) and can slow down the frequency at which the VT1422A can scan channels. For this reason, this method should be limited to only those channels needed for the algorithms.

Temperature Measurements

The VT1529B adds the capability to measure E, J, and T type thermocouples in addition to strain (other thermocouple types are not supported with the new functionality). Either the VT1586A Rack-Mount Terminal Panel or a user provided isothermal panel can be used. For this discussion, assume that the VT1586A and the VT1529B option 002 VT1586A-to-VT1529B cable are being used.

Note If thermocouple types other than E, J, or T need to be used, they must be configured for dc voltage measurements and their compensation routines must be provided.

Connecting the VT1586A to the VT1529B

To use the VT1586A with the VT1529B, the VT1529B option 002 VT1586A-to-VT1529B cable should be used. This cable provides connections for up to sixteen channels of temperature measurements (therefore, two cables are required for the 32 channels on a VT1586A).

Note For each VT1586A used with the VT1529B, two channels for the reference thermistor measurement must be dedicated. This leaves thirty channels for thermocouple measurements, unless the VT1586A is in a temperature-controlled environment and SENS:REF:TEMP:POST is used to specify the reference temperature for the EU conversion routines.

On one end of the cable, there is a SCSI connector that plugs into one of the two SCSI jacks on the back of the VT1586A. On the other end, there are sixteen RJ-45 jacks that plug into the front of VT1529B. The RJ-45 jacks are numbered to make the interconnection easy. Figure 5-2 and Figure 5-3 show how to make the connections.

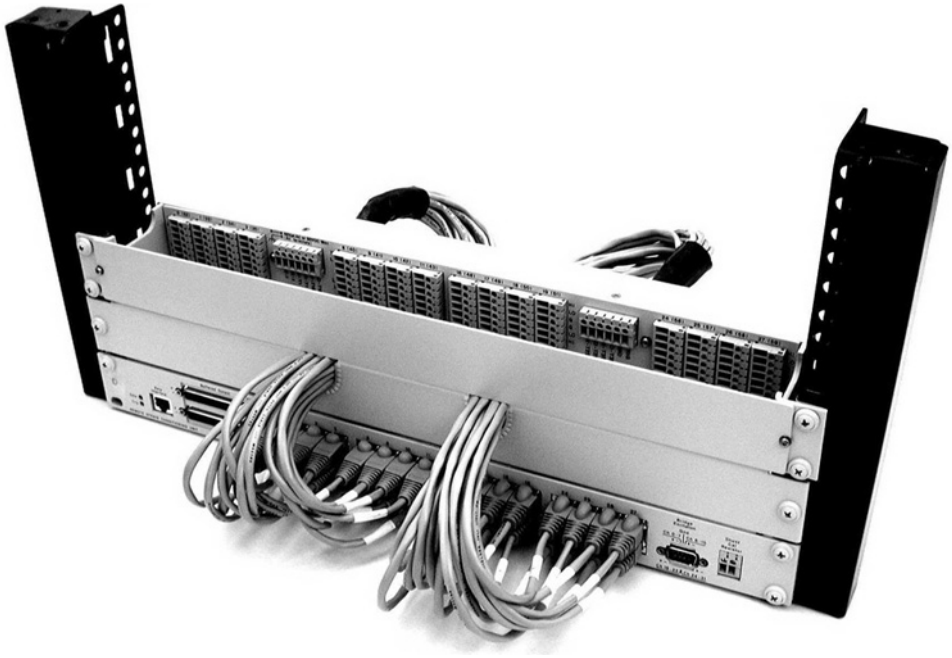


Figure 5-2. Connecting the VT1586A to the VT1529B (front view)

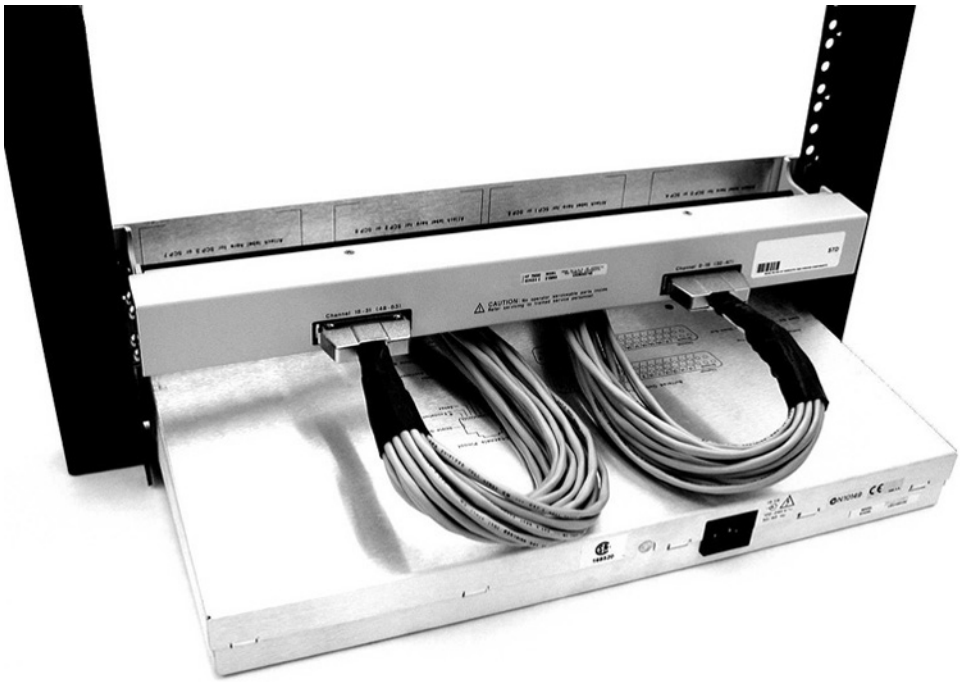


Figure 5-3. Connecting the VT1586A to the VT1529B (rear view)

Field Wiring of the VT1586A

The field wiring for the VT1586A, when used with the VT1529B, is different from the wiring of the VT1586A when used stand-alone. In particular, with the VT1529B option 002 VT1529B-to-VT1586A cable, the G (Guard) terminals of the VT1586A are redefined to be the excitation source and excitation sense terminals for the reference thermistor(s). Figure 5-4 shows the VT1586A front panel connections. Note that the Sense + and - terminals for the channels are the same using the VT1586A.

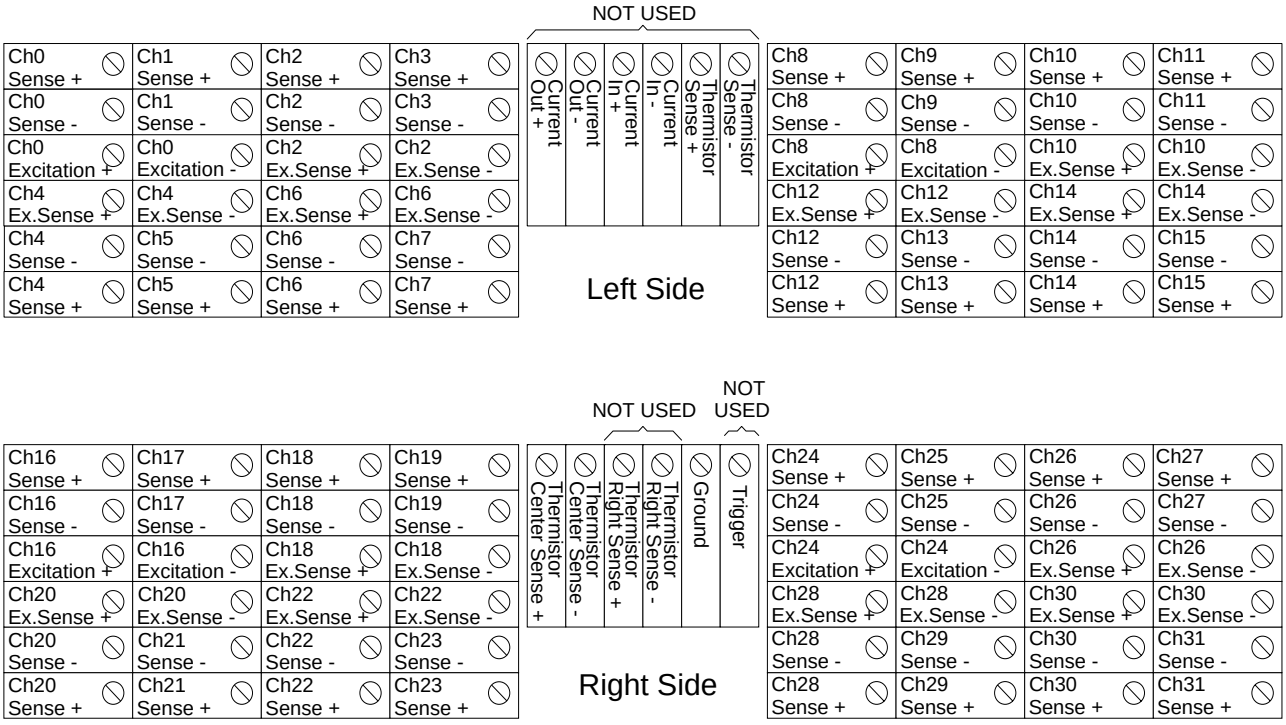
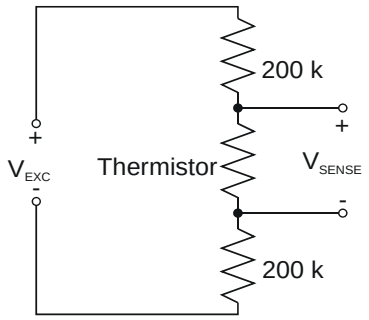


Figure 5-4. VT1586A Terminal Panel Connections

When using the VT1586A with the VT1529B to measure temperature, instead of using a current source to excite the isothermal reference thermistor, use a voltage source and a resistor divider network. The excitation voltage supply that the VT1529B is connected to is routed through the VT1529A-to-VT1586A cable and is made available on the four Excitation + and - terminals on the VT1586A for use with the divider. The recommended thermistor divider circuit is shown in Figure 5-5 and the recommended field wiring for the divider in shown in Figure 5-6. Note that the excitation voltage is not being connected to a measurement channel. Instead, the same dedicated excitation measurement channel on the VT1529B can be used that was use for strain.

Note The Excitation Sense + and - channels shown in Figure 5-4 are normally not used. They can be used to "wrap around" the excitation voltage outputs on the VT1586A to the corresponding measurement channel on the VT1529B, if necessary.

Suggested Wiring for Thermistor Measurement



Note: V_{SENSE} should be balanced with respect to ground

Figure 5-5. Recommended Reference Thermistor Divider Circuit

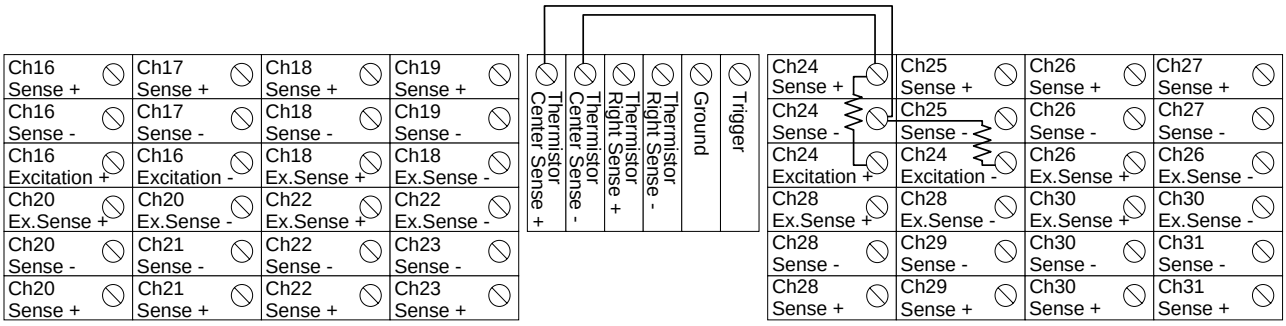


Figure 5-6. Recommended Reference Thermistor Field Wiring

Notes

The 200 kΩ resistors in the recommended divider circuit were chosen so that the V_{SENSE} output voltage from the 5 kΩ reference thermistor will be less than 0.5 V over the temperatures above -15 °C with an excitation voltage of 5 V or less. (0.5 V is the maximum input voltage that is allowed between the Sense + and - terminals of the VT1529B.) The thermistor compensation routines in the VXI*plug&play* driver assume that the sum of the two resistors is exactly 400 kΩ. The SENS:REF:THERM:RES:POST command can be used to specify the actual total resistance value in the divider circuit.

If the combination of excitation voltage and isothermal temperature range produces divider output voltages greater than 0.5 V, change the resistor values shown to keep the output voltage below 0.5 V. Also, use the SENS:REF:THERM:RES:POST command to specify the sum of the resistor values so that the compensation routines will work correctly.

Temperature Measurement Command Sequence

The steps for measuring temperature with the VT1529B and VT1586A should, again, be familiar. If a controlled temperature isothermal block is used, the command sequence is identical, except for the :POST keyword being added to some commands. Otherwise, the isothermal block temperature needs to measured, the major difference being that it is necessary to measure the excitation voltage, thermistor output voltage, and thermocouple voltage instead of just the thermistor and thermocouple voltage. This is because the VT1529B with VT1586A excites the thermistor with the excitation supply voltage that is provided instead of a calibrated current.

The thermistor measurement channel must be associated with its excitation voltage measurement channel as well as associating each thermocouple to the thermistor. The excitation voltage and thermistor measurement channels must be part of the scan list. Figure 5-7 shows the sequence of VXI*plug&play* driver calls to configure and scan thermocouple channels using the new functionality.

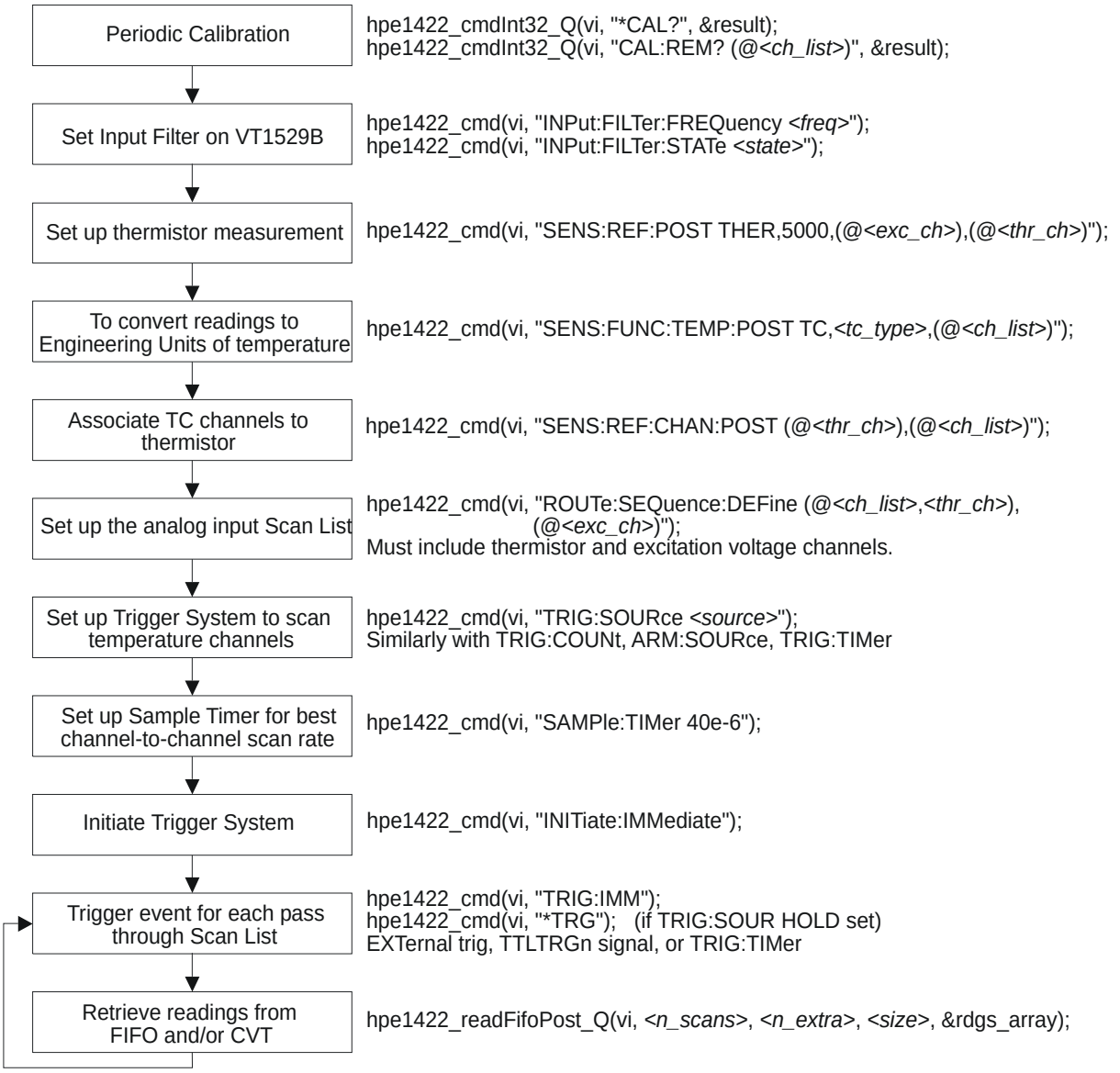


Figure 5-7. Sequence for VT1529B Temperature Conversion

Temperature Conversion Sequence

Below is an example VXI*plug&play* command sequence. Note that this is not executable; it has been simplified for easier reading. The C++ example source file (*tcpost.cpp*) is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

```
/* set up reference thermistor measurement
   note that ch 10006 is excitation voltage, 10007 is thermistor voltage */
errStatus = hpe1422_cmd(vi, "sens:ref:post ther,5000,(@10006),(@10007)");

/* set function to temperature on channels 10000-10005 */
errStatus = hpe1422_cmd(vi, "sens:func:temp:post TC,J,(@10000:10005)");

/* associate TC channels with reference thermistor channel */
errStatus = hpe1422_cmd(vi, "sens:ref:chan:post (@10007),(@10000:10005)");

/* optionally set VT1529B input filters (2, 10 or 100 Hz) */
errStatus = hpe1422_cmd(vi, "inp:filt:freq 10,(@10000:10005)");
errStatus = hpe1422_cmd(vi, "inp:filt:stat on,(@10000:10005)");

/* set up the scan list to include the TC, thermistor and excitation channels */
errStatus = hpe1422_cmd(vi,"rout:seq:def (@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(vi,"trig:coun 1");     /* *RST default */
errStatus = hpe1422_cmd(vi,"trig:sour tim");   /* *RST default */
errStatus = hpe1422_cmd(vi,"arm:sour imm");    /* *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 us sample time. */
errStatus = hpe1422_cmd(vi,"samp:tim 40e-6");  /* *RST default */

/* INITiate the trigger system to execute a measurement scan */
errStatus = hpe1422_cmd(vi,"init:imm");

/* retrieve one scan's data from the FIFO */
errStatus = hpe1422_readFifoPost_Q(vi, 1, 0, 65024, f64_array);
```

Voltage Measurements

With the VT1529B, low- or high-level dc voltage measurements can also be made for sensors such as LVDT and string pots. While the VT1422A does not provide built-in engineering units conversions for these sensor types, custom EU conversions can be provided by the user.

Note If a custom EU conversion is provided on a high-level dc voltage path, use the SENSe:FUNCtion:CUSTom:HVOLtage command. Also see the DIAGnostic:CUSTom:MXB command for specifying the custom conversion parameters.

Field Wiring for dc Voltage Measurements

There are separate input pins on the RJ-45 connector for the high-level and low-level DCV signals. See Figure 5-8 for the connector pin-out.

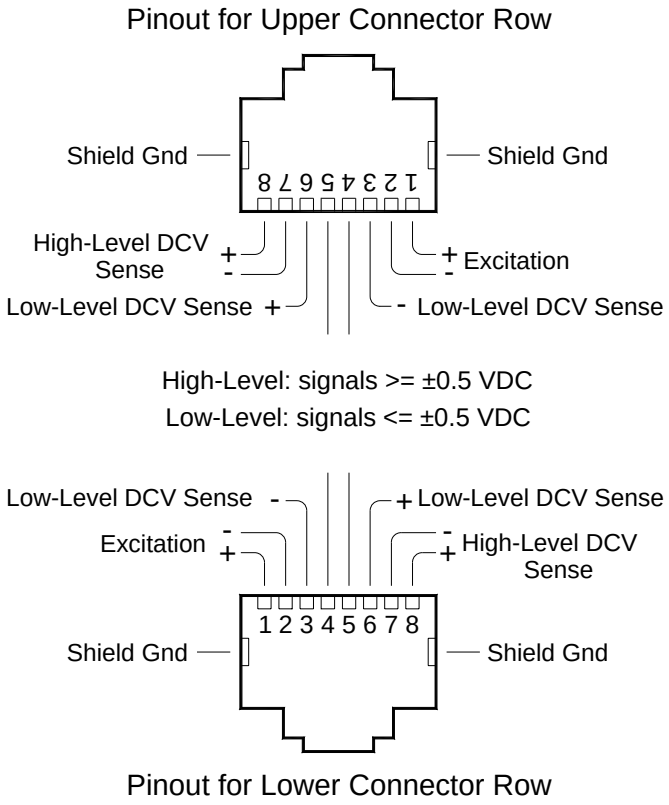


Figure 5-8. Pin-out for DCV Measurements

Low-Level Inputs

The low-level input pins are limited to signals less than ±0.5 V dc. These inputs can be balanced with respect to chassis ground or can be single-ended. There are three optional low-pass filters that can be used on the low-level inputs, with cut-off frequencies of 2, 10, and 100 Hz. These filters are selected by the INPut:FILTer[:LPASS]:FREQuency command. After the filter, the low-level inputs go through an amplifier to give higher noise immunity before being sent to the VT1422A. When the VT1422A measures the amplified low-level signal, it internally compensates for the amplification so that the value sent to algorithms, the CVT and the FIFO is the actual voltage at the low-level input pins.

High-Level Inputs

The high-level input pins can accept signals up to 10 V dc. These inputs should be balanced with respect to chassis. An unbalanced signal will not settle quickly if a long cable is placed between the VT1422A and the VT1529A/B, which can cause measurement errors both on the unbalanced input ***as well as on the next few channels*** in the scan list.

There is no filtering or amplification in the high-level input path.

Note The high-level inputs actually use the same excitation voltage measurement path that is used for strain measurements. This means that there is an internal 10 kΩ resistor from each terminal to a common point for each 8-channel block as shown in Figure 5-9. During excitation voltage measurement of any channel in any bank on a VT1529B, the switches shown in the Figure 5-9 are closed on all banks, connecting the Bridge Excitation inputs on the front panel to the input resistors, which will cause a small current to be sourced to the other channel inputs. During other measurements, the switches are opened and the resistors become a connection between all bank + inputs and all - inputs. This connection can affect the input signal levels unless the inputs are driven from a low impedance source.

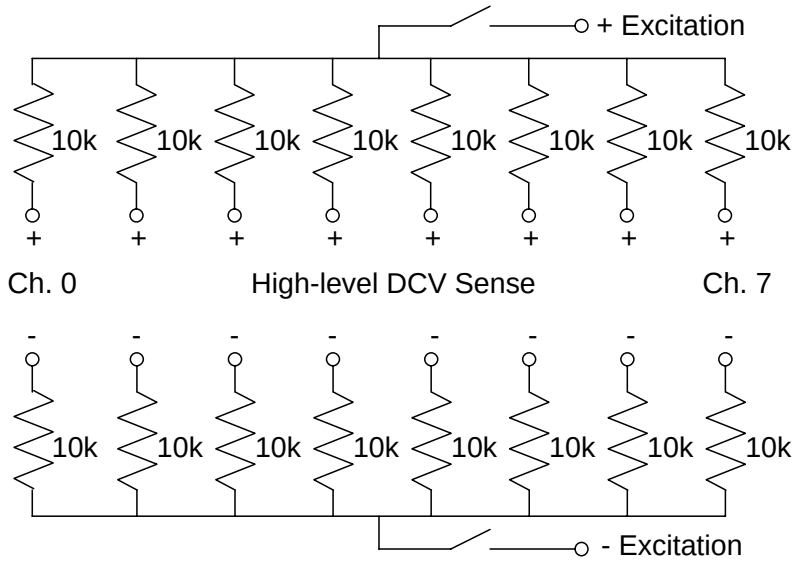


Figure 5-9. High-Level Input Circuit

DCV Measurement Command Sequence

For the low-level inputs, the input filter needs to be set up and the channels for voltage measurements need to be linked. The VXI*plug&play* driver calls for this are:

```
hpe1422_cmd(vi, "INP:FILT:FREQ 10");                    Set up 10 Hz input filter
hpe1422_cmd(vi, "INP:FILT:STAT ON");                    Enable filter
hpe1422_cmd(vi, "SENS:FUNC:VOLT AUTO,(@10000:10005)");  Measure DCV on
                                                        low-level inputs on channels
                                                        10000-10005
```

For the high-level inputs, simply link the channels for voltage measurements. Use the “SENS:FUNC:HVOL” command:

```
hpe1422_cmd(vi, "SENS:FUNC:HVOL AUTO,(@10008:10015)");  Measure DCV on
                                                        high-level inputs on channels
                                                        10008-10015
```

DCV Measurement Sequence

Below is an example VXI*plug&play* command sequence. Note that this is not executable; it has been simplified for easier reading. The C++ example source file (*dcv.cpp*) is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

```
/* This shows measuring low-level inputs on channels 10000-10005 */
/* and high-level inputs on channels 10008-10015 */

/* optionally set VT1529B input filters (2, 10 or 100 Hz) */
errStatus = hpe1422_cmd(vi, "inp:filt:freq 10,(@10000:10005)");
errStatus = hpe1422_cmd(vi, "inp:filt:stat on,(@10000:10005)");

/* set up channels 10000-10005 for DCV, low level inputs */
errStatus = hpe1422_cmd(vi, "sens:func:volt auto,(@10000:10005)");

/* set up channels 10008-10015 for DCV, high level inputs */
errStatus = hpe1422_cmd(vi, "sens:func:hvol auto,(@10008:10015)");

/* set up the scan list to include the DCV channels */
errStatus = hpe1422_cmd(vi,"rout:seq:def (@10000:10005,10008:10015)");

/* 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(vi,"trig:coun 1");     /* *RST default */
errStatus = hpe1422_cmd(vi,"trig:sour tim");   /* *RST default */
errStatus = hpe1422_cmd(vi,"arm:sour imm");    /* *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 us sample time. */
errStatus = hpe1422_cmd(vi,"samp:tim 40e-6");  /* *RST default */

/* INITiate the trigger system to execute a measurement scan */
errStatus = hpe1422_cmd(vi,"init:imm");

/* retrieve one scan's data from the FIFO */
/* Note: if you're not doing any measurements that require post-processed
   EU conversions, you can use hpe1422_readFifo_Q instead */
errStatus = hpe1422_readFifoPost_Q(vi, 1, 0, 65024, f64_array);
```

Settling Time Considerations

Channel settling interactions are present in every scanning system. A significant engineering effort has been applied to minimize these interactions, however, if the correct situation is present, errors caused by settling can be detected. This section presents a brief tutorial on settling errors and explains how to control the source of these errors.

Settling errors occur when the voltage value of one channel seems to leak into successive channels being scanned. This can be called *switching crosstalk*. Traditional crosstalk occurs when one input signal interferes with another input and is related to the frequency of the interfering signal. However, switching crosstalk is not related to the signal frequency, but rather is a function of the following factors which can be controlled:

- The dc voltage of the interfering channel relative to the channel being measured.
- The amount of time the system remains on the interfering channel before measurement of the desired channel ("soak" time).
- The time elapsed between switching from the interfering channel to the actual measurement on the desired channel ("discharge" time).

Some of the mechanisms by which previous channels interfere are micro-thermals in semiconductors, dielectric absorption in materials and simple capacitance. The cumulative effects of all of the mechanisms from previous channels result in errors on the current channel. A simple model of the error is not possible. It should also be noted that not all of the effects have the same polarity or time constant.

The following section describes several things that can be done to keep switching crosstalk to a minimum.

1. Limit the voltage of the channel(s) before a low-voltage channel is measured. The VT1529A/B can measure 0.5 V and 1 μV on successive channels. However, if this is done at the fastest scanning rate, the VT1529A/B must try to slew 114 dB in less than 40 μs and switching crosstalk errors will occur. Instead, order the channels to minimize the channel-to-channel input voltage change. For the VT1529A/B, the following channel sequences are recommended:

- If the scan list on a VT1529A/B includes an excitation voltage channel, the order should be as follows:
  - -- Channels with small voltages (μV or mV level) on the low-level inputs: strain gages, thermocouples, etc.
  - -- Channels with large voltages (greater than 0.1 V) on the low-level inputs: reference thermistor voltages.
  - -- Channels using the high-level inputs: excitation voltages, high-level dc voltages. Short the low-level inputs on these channels to chassis ground.
- If the scan list on a VT1529A/B does *not* include an excitation voltage channel, the order should be as follows:
  - -- Channels with large voltages (greater than 0.1 V) on the low-level inputs: reference thermistor voltages.
  - -- Channels with small voltages (μV or mV level) on the low-level inputs: strain gages, thermocouples, etc.

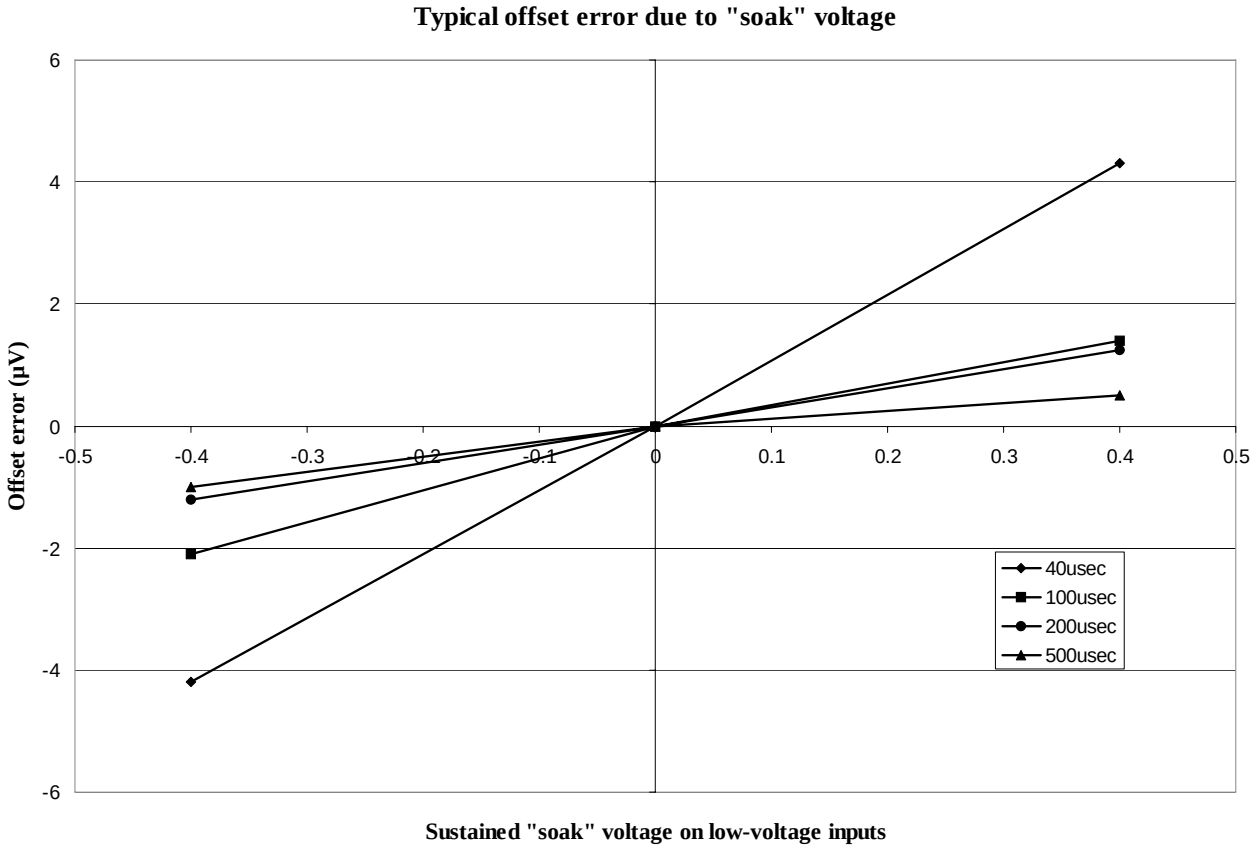
If an unconnected channel is included in a scan list, short the inputs to chassis ground. An unconnected input will drift to the full-scale input and can cause switching crosstalk errors. Also, shorting the low-level inputs of channels used for high-level measurements to chassis ground is recommended – the VT1529A/B always switches *both* the high-level and low-level inputs and then selects between the two inputs. Shorting the low-level inputs on these channels will prevent drifting of the amplifier for the low-level inputs during high-level measurements.

2. Minimize the time the system remains on a large signal (> 0.1 V) on the low-level inputs. Leaving a channel on a large voltage will "soak" the system, but quickly scanning through a high voltage will cause minimal error. The longer the system soaks, the longer it takes to discharge. For example, at the fastest scanning speed of the VT1529A/B (40 μs if only one channel in the middle of a scan is 80% full scale (0.4 V)), then the effect on the following channel at near 0 V is an offset of about 2.4 μV. But, if many channels are scanned at 0.4 V followed by a channel near 0 V, then the effect is about 4 μV. The key point to remember is to avoid soaking the system unnecessarily.

When a VT1529A/B scans through a list of channels, it stays on the last channel in the scan list for that VT1529A/B until the unit is scanned again. To minimize the “soaking” effect, have a low voltage or a short on the low-level inputs on this last channel.

3. Reduce “switching crosstalk” by increasing time between channels. Increasing the time between channel measurements will give the VT1529A/B more time to settle. In the worst case, when all but one channel soaks the system with 80% of full-scale input voltage (0.4 V) and the system remains on this soaking voltage between scans, the VT1529A/B can settle to within about 4 μV in 40 μs. Note that this is within the offset spec of 2 μV within 100 μs. For best accuracy, scan at the slowest rate that is acceptable for that particular application.

The graph below shows the worst-case scenario for the VT1529A/B. As stated earlier, if only one channel is 80% of full scale, the switching crosstalk is barely measurable. Most system configurations do not approach the worst-case scenario.



Chapter 6
Creating and Running Algorithms

Learning Hint This chapter builds upon the "VT1422A Programming Model" information presented in Chapter 4. Reading Chapter 4 is recommended before moving on to this chapter.

About This Chapter

This chapter describes how to write algorithms that apply the VT1422A's measurement, calculation, and control resources. It describes these resources and how to access them with the VT1422A's Algorithm Language. This manual assumes that the user has programming experience already, ideally, in the 'C' language since the VT1422A'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 VT1422A'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'. An algorithm cannot call another user-written function subprogram.

It is important to note that while the VT1422A'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 F page 487 shows how to convert user defined functions into piece-wise linear interpolated tables which can be downloaded into the VT1422A. The VT1422A can extract function values from these tables in approximately 18 μs 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 the user already programs in some language. If the experience is in the 'C' programming language, this chapter is probably all that will be needed to create an algorithm. If unfamiliar with the 'C' programming language, studying the "Program Structure and Syntax" section before beginning to write algorithms is recommended.

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 193.

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 seen 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 20 mA 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 parallel to provide 20 mA */
/* 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 VT1422A provides for algorithms. Here, the relationship of algorithms to the **main()** function that calls it is described.

The Main Function

All 'C' language programs consist of one or more functions. A 'C' program must have a function called **main()**. In the VT1422A, the **main()** function is usually generated automatically by the driver when the INIT command is executed. The **main()** function executes each time the module is triggered and controls execution of algorithm functions. See Figure 6-1 for a partial listing of **main()**.

How User Algorithms Fit In

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

Since the function-calling interface to an algorithms is fixed in the **main()** function, the "header" of an algorithm function is also pre-defined. This means that, unlike standard 'C' language programming, an algorithm program (a function) need not (must not) include the function declaration header, opening brace "{" and closing brace "}". Only the "body" of the function is supplied; the VT1422A's driver supplies the rest.

Think of the program space in the VT1422A in the form of a source file with any global variables first, then the **main()** function followed by as many algorithms as defined. Of course, what is really contained in the VT1422A's algorithm memory are executable codes that have been translated from the downloaded source code. While not an exact representation of an algorithm execution environment, Figure 6-1 shows the relationship between a normal 'C' program and two VT1422A algorithms.

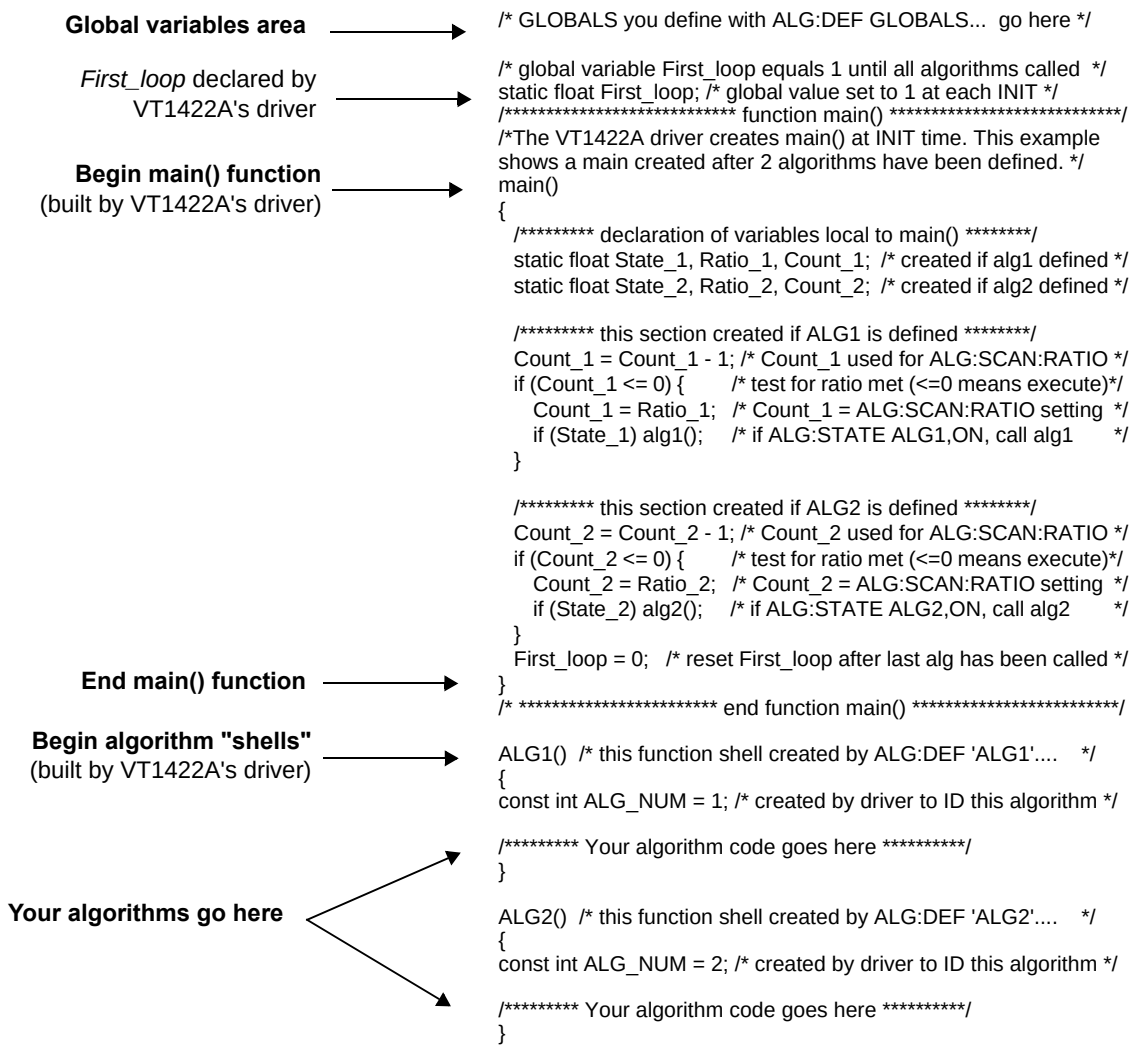


Figure 6-1. Source Listing of Function main( )

Accessing the VT1422A's Resources

This section describes how algorithms access hardware and software resources provided by the VT1422A. The resources are listed below:

- I/O channels.
- Remote Scan Status variables.
- Global variables defined before algorithms are defined.
- The constant ALG_NUM which the VT1422A makes available to the algorithms. 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 the 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 (*RSCU*s), there are two forms of input channel syntax. The general on-board input channel identifier syntax is "I1*cc*" where *cc* is a channel number from 00 (channel 0) through 63 (channel 63). The Remote input channel syntax is "I1*ccrr*" 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 231 for more information.

For output channels, the syntax is "O1*cc*," where *cc* is a channel number from 00 (channel 0) through 63 (channel 63). Like all VT1422A 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 VT1533A), the channel's identifiers (O1cc and I1cc) can take on the values 0 through 255. To access individual bit values, ".B*n*" may be appended 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 VT1534A), its channel identifiers can only take on the values 0 and 1. Examples:

```
O100 = 1;                    assign value to output chan 0 on
                             VT1534A.
Inp_val = I108;              from 8-bit channel on VT1533A
                             Inp_val will be 0 to 255.
                             np_val will be 0 to 255.
Bit_4 = I109.B4;             assign VT1533A 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;         complement 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 Output Channel Buffer element 56

```
writefifo(I124);
```
send value of input channel 24 to FIFO buffer

Defined Input and Output Channels

Algorithms "reference" channels. They can reference input or output channels. But, in order for these channels to be available to an algorithm, they must be "defined." What this means is that an SCP or RSCU must be installed and an appropriate SOURce or SENSe:FUNCtion must explicitly (or implicitly, in the case of VT1531A and VT1532A SCPs) be tied to the channels. If an 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 the algorithm is defined with ALG:DEF.

Accessing Remote Scan Status Variables

Two remote scan status variables exist for each VT1539A SCP. The variable syntax is "S1<*xx*>" where <*xx*> can be one of 00, 01, 08, 09, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, and 57. These values are sixteen possible on-board channels for VT1539A SCPs. These variables are treated as input channels and can only be read from, not written to. The returned value represents the operational status of the RSCU connected to the SCP channel.

The possible values are: 0 = normal operation, 1 = RSCU cable disconnected after INIT and 2 = RSCU scan list is out of synchronization (scan trigger problem during RSCU's scan).

Runtime Remote Scan Verification

For most data acquisition and control applications, it is necessary to know that the data acquired via an RSCU is not corrupted at runtime. Though the VT1422A/RSCU system is designed to be very robust and will normally acquire reliable data, there is a possibility that remote data can be corrupted in the following ways:

- The remote box (RSCU) may lose power after a scan is initialized.
- The cable to the remote box may become disconnected or damaged during the scanning process.
- The multiplexer in the remote box may lose synchronization with the host VT1422A due to false or missing triggers.

For all of the above conditions, the VT1422A/RSCU system should report the existence of corrupted data immediately to the host system.

Runtime Scan States

The Remote Signal Conditioning Units (RSCU) runtime scan can be described as being in one of the following three states: normal, disconnected and out of synch. In the disconnected state, the VT1422A fails to communicate with an RSCU for various reasons (power down, no connection, a malfunction in the cable, etc.). In the out of synch state, the RSCU channel switching multiplexer could be out of synch with the VT1422A due to an extra or missing trigger.

Algorithm Language Support

Since each RSCU is connected to one of the main channels in the host VT1422A, the operating status of each RSCU during the scanning can be represented by the status of the corresponding main channel. A block of memory has been reserved to hold the status for each main channel that is connected to an RSCU. Referencing the Scan Status variable in an algorithm will return the operating status of the specified RSCU.

The language support for status reporting is designed as follows.

Scan status variables have the syntax *S1xx*. *S* stands for Status. *1xx* is the corresponding main channel that is connected to an RSCU. Though the possible range for main channels on a VT1422A is 00 to 63, not all of the main channels are valid for an RSCU connection. Therefore, the current valid range for *xx* is the set of discrete numbers (00, 01, 08, 09, 16, 17, 24, 25, 32, 33, 40, 41, 48, 49, 56, 57). For example, in order to read the operating status of an RSCU connected to channel 24 of a VT1422A, S124 would be referenced in the algorithm. S1xx is treated as an input channel and can only be read from, not written to.

Currently, S1xx can be in one of the three states which are represented by 0, 1, and 2.

- S1xx is 0, when RSCU is in the normal state.
- S1xx is 1, when RSCU is in the disconnected state.
- S1xx is 2, when RSCU is in the out of synch state.

Operating Model

Scan Status Variables are updated just after the all channels are scanned in the Input Phase (refer to “The Operating Sequence”). The value of the scan status variables is available to algorithms when they execute.

Example Scan Verification Algorithms

Following are some simple examples demonstrating the use of scan verification in an algorithm.

By using S1xx in an algorithm, various results for control or data acquisition applications can be achieved. Assuming that one is interested in monitoring the scan state of an RSCU at VT1422A channel 25, the following can be done:

1. In the VT1422A algorithm:

```
if( S125 != 0) {
/* User application specific algorithm code for shutting down critical hardware or
reporting. Such as:
*/
O156 = 5.0 ;                    /* send 5.0 volts to output chan, then open relay */
      writecvt(S125, cvt_loc);  /* for reporting */
      interrupt();              /* force a VXI bus interrupt. */
}
```

2. In the application:

S1xx states can also be achieved by using various *Plug&Play* functions or SCPI commands, such as:

```
alg:scal? 'algn','S125'
```

Timing Impact

One of the tradeoffs for runtime scan verification is an impact on the overall scan rate. The total time spent in scan verification depends on the number of unique S1xx locations that are referenced in the algorithms. The total time in scan verification can be determined by the following formula:

Total Time in scan verification:

= 0 when no S1xx locations are referenced.

= 230 μs + 40 μs*(number of unique S1xx identifiers referenced in algorithms).

max time = 870 μs, if all sixteen S1xx locations are referenced.

Thus, the flexibility to make tradeoffs between maximum scan rate and scan status checking is achieved.

The impact of using scan verification on the overall scan rate depends on both the number of channels in the analog input scan list and the number of channels in status list. In a system which uses the maximum number of 512 analog input channels and the maximum sixteen status channels, the total timing overhead will be 870/(512*40) = 4.24%.

Also, if the status information is used to make decisions within an algorithm, additional time will also be required. The timing impact in an algorithm will depend on its complexity.

Defining and Accessing Global Variables

Global variables are those declared outside of both the **main()** function and any algorithms (see Figure 6-1). A global variable can be read or changed by any algorithm. To declare global variables, 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];'
```

Global variables are accessed within an 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 an algorithm are local to that algorithm. If a variable with the same identifier as an existing global variable is locally declared, the algorithm will access the local variable only.

Determining First Execution (First_loop)

The VT1422A 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*, an 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 VT1422A operations.

1. When algorithms are defined 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 an 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 an algorithm executes. By placing an assignment statement within an algorithm. This will be executed each time the algorithm is executed. Example statement.

   ```
   my_variable = 22.95;/* reset variable to 22.95 every pass        */
   ```

3. When an algorithm first executes after an INIT command. By using the global variable *First_loop*, an 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 an algorithm to the application program (running in the 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 7-4 on page 324). To send a value to a CVT element, 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 an 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 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 323. for controlling CVT entries from the analog scan list.

Reading CVT elements

An 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 an 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 the actual algorithm execution rate can be determined (see "Programming the Trigger Timer" on page 131), 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 134.

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 382 and "*SRE" on page 404). 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 An Algorithm's Identity (ALG_NUM)

When an algorithm is defined with the ALG:DEF 'ALGn',... command, the VT1422A's driver makes available to the algorithm the constant *ALG_NUM*. *ALG_NUM* has the value *n* from "ALG*n*." For instance, an algorithm is defined with <*alg_name*> equal to "ALG3", then *ALG_NUM* within that algorithm would have the value 3.

What can be done with this value? A short example of the code that uses *ALG_NUM* is provided to illustrate:

```
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 identical code to be written 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 an algorithm. Essentially, what is provided with the VT1422A 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 m*x*+b over the interval (*x* is the value provided 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 a function equation, see "Generating User Defined Functions" in Appendix F page 487. For details on the ALG:FUNC:DEF command, see page 249 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 VT1422A. The function can be called 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 VXI*plug&play* program that shows the use of a user defined function is supplied on the driver disc ("*tri_sine.cpp*"). The program is on the CD supplied with the instrument. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

Operating Sequence

This section explains another important factor in an algorithm's execution environment. Figure 6-2 shows the same overall sequence of operations seen in Chapter 3, but also includes a block diagram to show which parts of the VT1422A 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 seen it here, however, it can be easily avoided. The following algorithm statements will help explain:

```
O124.B0 = 1;/* digital output bit on VT1533A in SCP position 3 */
O124.B0 = 0;
```

Traditionally, the first of these two statements is expected 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 advantageously. 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, the order in which the algorithms will be executed must known. When algorithms are defined, they are given one of 32 pre-defined algorithm names. These range from 'ALG1' to ALG32.' The algorithms will execute in order of their name. For instance, 'ALG5' is defined, then 'ALG2,' then 'ALG8,' and finally 'ALG1'; when they are run, 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 201.

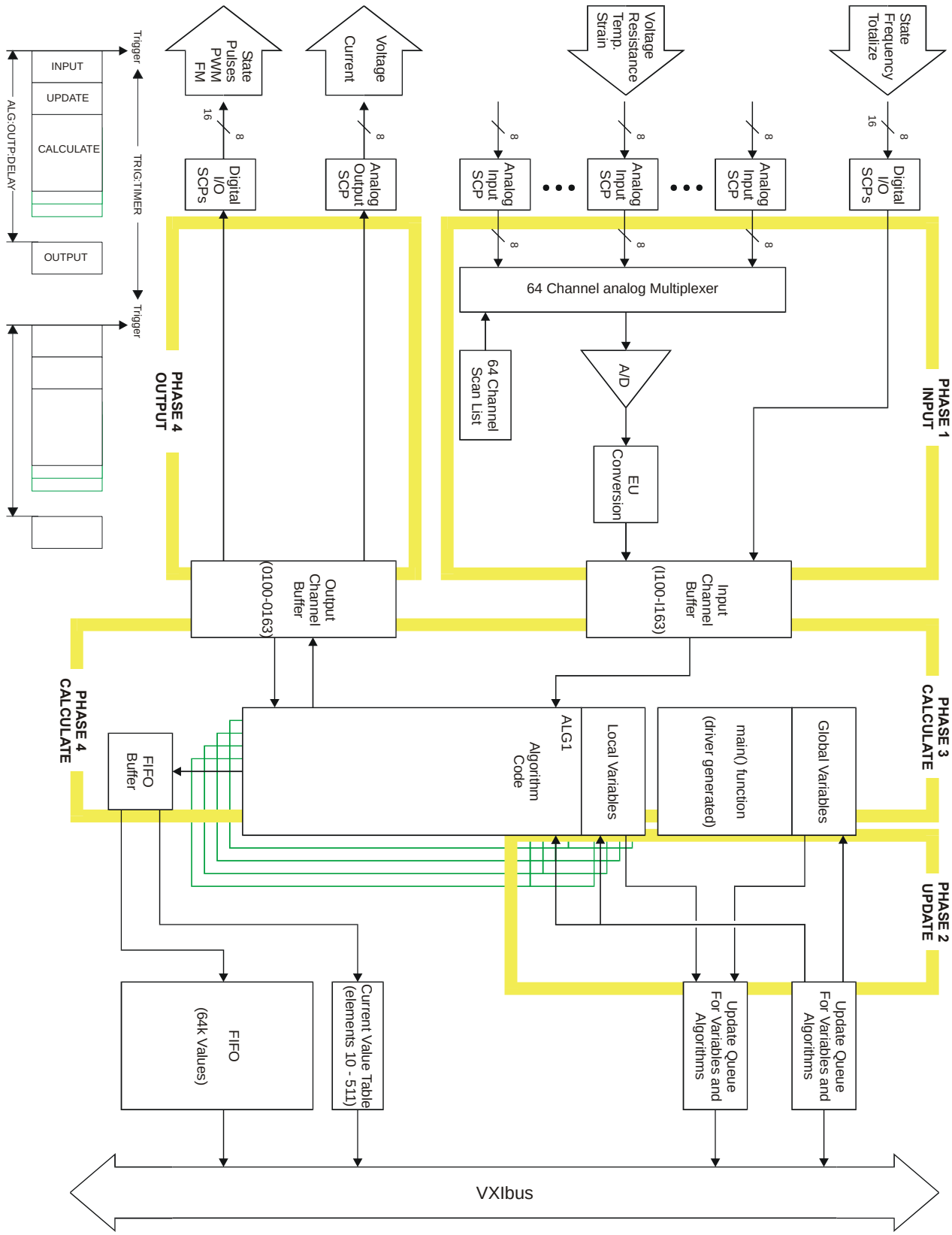


Figure 6-2. Algorithm Operating Sequence Diagram

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 VXI*plug&play* Users

While the following discussion of algorithm definition is useful for *plug&play* users with regards to the coding of an algorithm or global variable definition, generating an algorithm code and actually downloading it to the VT1422A is much easier because of the *plug&play* hps1422.exe Soft Front Panel program and hpe1422_downloadAlg(...) *plug&play* driver function.

Using the SFP "Algorithm Panel," an algorithm can be created and tested and then stored to a file. The hpe1422_downloadAlg(...) *plug&play* driver function was created specifically to download algorithms from files into the VT1422A as part of the application program.

ALG:DEFINE in the Programming Sequence

*RST erases all previously defined algorithms. All algorithms must be erased before beginning to re-define them (except in the special case described in "Changing a Running Algorithm" on page 197 later in this section).

ALG:DEFINE's Three Data Formats

The ALG:DEFINE '<*alg_name*>','<*source_code*>' command sends an algorithm's source code to the VT1422A'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, it is 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 receives an End Identifier with the last data byte. Use this form only when it is certain that the 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. The null byte must be appended 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 VT1422A 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( VT1422A,  "ALG:DEF  'ALG8',#0%s%c\n", source_buffer, null );
```

See the section "Running the Algorithm" later in this chapter for more information on loading algorithms from files.

Changing a Running Algorithm

The VT1422A has a feature that allows for a given algorithm to be swapped with another even while it is executing. This is useful if, for instance, it is necessary to alter the function of an algorithm that is currently controlling a process and it would be undesirable for that process to be uncontrolled. In this case, when the original algorithm is defined, 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*>, the amount of algorithm memory is specified to allocate for algorithm <*alg_name*>. Make sure to allocate enough space for the largest algorithm expected to be defined 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. The 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?

The example algorithm definition above will be used for this discussion. When a value for <*swap_size*> is specified at algorithm definition, the VT1422A 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," arbitrarily called space A and space B. The algorithm is loaded into ALG3's space A at first definition. Later, while algorithms are running, "replace" ALG3 by again executing:

```
ALG:DEF ALG3,#42435<2435char_alg_source>
```

Notice that <*swap_size*> is not (must not be) included this time.
The ALG:DEF command works like an Update Request. The VT1422A 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 VT1422A simply switches to executing ALG3's new algorithm from space B at the next Update Phase (see Figure 6-2). If yet another ALG3 were sent, it would be loaded and executed from ALG3's space A.

Determining an Algorithm's Size

In order to define an algorithm for swapping, it is necessary to know how much algorithm memory to allocate for it or any of its replacements. It can be queried from the VT1422A using the following sequence:

1. Define the algorithm without swapping enabled. This will cause the VT1422A 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. The minimum amount of memory required for the algorithm is now known.

3. Repeat 1 and 2 for each of the algorithms that are to be swapped with the original. From this, the minimum amount of memory required for the largest is known.

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 that is to be run now. Remember, not to send the <*swap_size*> parameter with these. If an "Algorithm too big" error is not received, 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 an algorithm is re-defined (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 INIT is sent. 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 G page 491 for program listings. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

A Very Simple First Algorithm

This section shows 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 VT1422A, downloads (defines) the algorithm, starts, and then communicates with the running algorithm.

Writing the Algorithm

The most convenient method of creating an algorithm is to use the hps1422.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 "*file_alg.cpp*" is compiled, type `file_alg mxplusb.c` to run the example and load the algorithm. View the readme.txt file provided with the VXI*plug&play* driver for example program file location.

Modifying an Example PID Algorithm

While the example PID algorithms supplied as source files with the VT1422A can provide excellent general closed loop process control, there will be times when the process has specialized requirements that are not addressed by the as-written form of these PID algorithms. In this section, coping and modifying an example PID algorithm is described.

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 desired.

The value written 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 VT1422A'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, the operation of an example PIDA will be analyzed with a digital output as its control output.

How the Example PIDA Operates

A PID algorithm is to control a bath temperature at 140°F. 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 to go up rather than down. This process is now out of control!

Modifying the Example PIDA

This behavior is easy to fix. Simply modify the example PIDA algorithm source code (supplied with the VT1422A 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 a 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 desired (e.g., I100).

4. As in step 3, change all occurrences of "outchan" to the digital output channel/bit identifier desired (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 VT1422A is that this communication can take place entirely within the algorithms' environment. The 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 6-3 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;
```

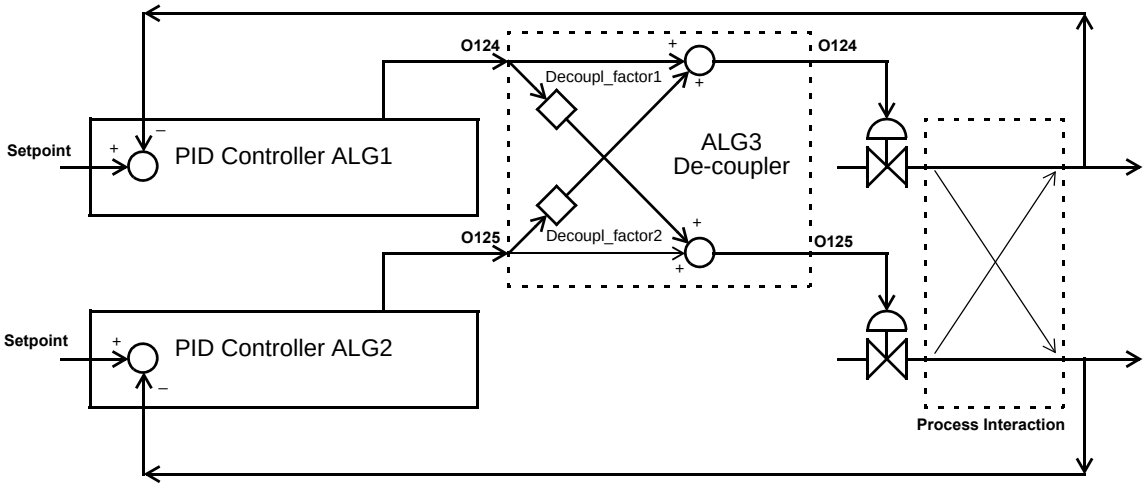


Figure 6-3. Algorithm Communication with Channels

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 6-4 shows the process diagram.

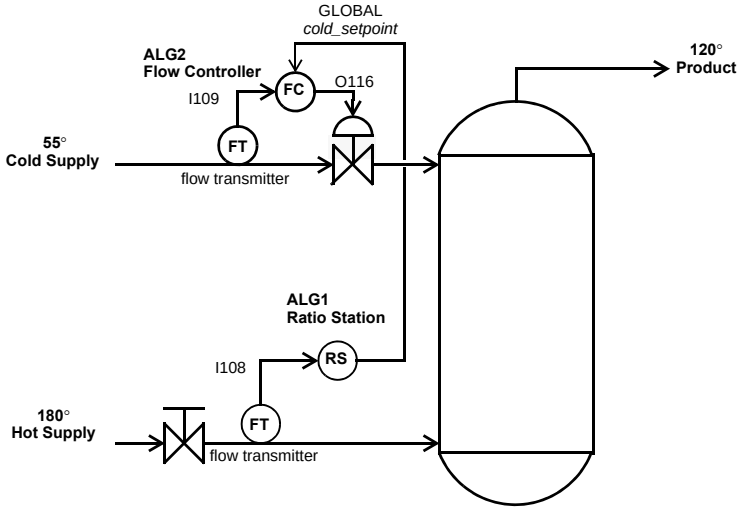


Figure 6-4. Inter-algorithm Communication with Globals

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 VT1422A 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 VT1533A 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 that are desired as inputs for 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 a device under test is desired to be cycled 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 1,024 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, successive 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 can be used.

- 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 1,023, the switch to the new array elements will take place. If the array is downloaded AFTER the index reaches 1,023, the same setpoint profile will be executed until index reaches 1,023 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 10 ms
ALG:ARRAY "alg1","setpoints",<block_data>     set first profile
ALG:UPD                                       force change
TRIG:TIMER .001                               trigger event at 1 ms
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 20 ms
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 VT1422A's may be used in a system. In this case, the VT1422A toggling the digital bit would be the master for the other VT1422A'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'. User variables may not be created using these names. Note that the keywords that are shown underlined and bold are the only ANSI 'C' keywords that are implemented in the VT1422A.

auto	double	int	struc
break	**else**	long	switch
case	enum	register	typeof
char	extern	**return**	union
const	**float**	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	**if**	**static**	while

NOTE While all of the ANSI 'C' keywords are reserved, only those keywords that are shown in bold are actually implemented in the VT1422A.

Special VT1422A Reserved Keywords

The VT1422A implements some additional reserved keywords. User variables may not be created 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.

Special Identifiers for Remote Scan Status

Remote Scan Status identifiers appear as variable identifiers within the algorithm and have a fixed, reserved syntax. The identifiers S100, S101, S108, S109, S116, S117, S124, S125, S132, S133, S140, S141, S148, S149, S156 and S157 specify scan status variables that are linked to the on-board channels of VT1539A SCPs. These VT1539A SCP channels are in turn connected to Remote Signal Conditioning Units (RSCUs). The identifiers are treated like input channel specifiers and may only appear on the right side of an assignment operator. The "S" must be upper case.

When accessed, these identifiers return one of three values: 0=normal RSCU operation, 1=RSCU cable disconnected after INIT and 2=RSCU scan is out of synchronization (RSCU scan trigger problem during scan).

NOTE Trying to declare a variable with a channel or status identifier will generate an error.

Operators

The VT1422A'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. Any variable may be tested 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 VT1422A'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 VT1422A. 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 VT1422A 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 an algorithm. It is not required to include a **return** statement to end an algorithm. The translator treats the end of an 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, it is necessary to understand that not all arithmetic statements in an algorithm are actually performed within the VT1422A's DSP chip at algorithm run-time. Where expressions can be simplified, the VT1422A'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 VT1422A. For example, look at the statement:

```
a = 5 + 8;
```

When the VT1422A'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." 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 was wanted, but is exactly what the algorithm instructed.

These subtle problems can be avoided by specifically including a decimal point in decimal constants where an integer operation is not desired. For example, if either of the constants in the division above is made into 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 VT1422A 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 an algorithm. Treating all variables this way allows an algorithm to "remember" its previous state. The static variable is local in scope, but otherwise behaves as a global variable. Also, note that declaring variables within a compound statement is not permitted.

Data Structures

The VT1422A 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 an array is declared, 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 an 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 VT1422A implements bitfield syntax that allows individual bit values to be manipulated 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. It is not necessary 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 for 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

Only simple variables can be initialized (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 190.

Global Variables

To declare global variables, 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 VT1422A'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 210.

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 1,023

Init-declarator:

declarator
declarator **=** *constant-expression*
NOTES: 1. May not initialize array declarator.
2. Arrays limited to single dimension of 1,024 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, the portion of the 'C' programming language that is directly applicable to the VT1422A' Algorithm Language will be discussed. To do this, the 'C' Algorithm Language elements will be compared with equivalent BASIC language elements.

Declaring Variables

In BASIC, the DIM statement is usually used to name variables and allocate space in memory for them. In the Algorithm Language, the variable type and a list of variables is specified:

BASIC	'C'
DIM a, var, array(3)	static float a, var, array[3];

Here, three variables are declared. 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, they must be explicitly declared as such.
- All variables in an algorithm are **static**. This means that each time the 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, the symbol "=" is used 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 ... . It will be seen later in this chapter that '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 the differences become familiar.

The Arithmetic Operators

The arithmetic operators available to the VT1422A 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 VT1422A 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.

BASIC	Examples	'C'	Examples
AND	IF A=B AND B=C	&&	if((a == b)&&(b == c))
OR	IF A=B OR A=C	||	if((a == b) || (a == c))
NOT	IF NOT B	!	if (! b)

Conditional Execution

The VT1422A 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 6-5).

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 6-5. 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 6-6 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
IF A<=0 THEN C=ABS(A)		if(a <= 0) c=abs(a);
IF A<>0 THEN C=B/A END IF		if(a != 0) c = b / a;

IF A<>B AND A<>C THEN
  A=A*B
  B=B+1
  C=0
END IF
		if((a != b) && (a != c))
		{
		  a = a * b;
		  b = b + 1;
		  c = 0;
		}

IF A=5 OR B=-5 THEN
  C=ABS(C)
  C= 2/C
ELSE
  C= A*B
END IF
		if((a == 5) || (b == -5))
		{
		  c = abs(c);
		  c = 2 / c;
		}
		else
		{
		  c = a * b;
		}

Figure 6-6. 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 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 it will be described how the VT1422A'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 7
VT1422A 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 VT1422A.

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 VT1422A 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. Upper or lower case letters can be used. 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 a second level command is sent but is not sent preceding implied command. In this case, the instrument assumes that the implied command is intend to be used and it responds as if it was sent. 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, 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 that will be seen 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, ms=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 VT1422A 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 VT1422A uses the VT1539A SCP to support RSCUs like the VT1529A/B. Through these SCPs, the VT1422A can address up to 512 remote channels. The VT1539A SCP has two 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 two 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 VT1422A. 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 VT1529A/B Remote Strain Bridge Conditioning Unit.

Some example remote channel specifications:
10000 = RSC channel 00 connected to VT1422A channel 00
10100 = RSC channel 00 connected to VT1422A channel 01
10122 = RSC channel 22 connected to VT1422A channel 01
12522 = RSC channel 22 connected to VT1422A 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 VT1422A will correctly address all channels in the range according to the type of SCP installed at the channel position (remote for VT1539A 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 VT1422A's **ROUTe:SEQuence:DEFine (@<ch_list>)** command. For the VT1422A, 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 VT1422A 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 together 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 GPIB 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 four 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 a value is not specified for an optional parameter, the instrument chooses a default value. For example, consider the FORMAT:DATA <*type*>[,<*length*>] command. If the command is sent without specifying <*length*>, a default value for <*length*> will be selected depending on the <*type*> of format specified. 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 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 sending a complete SCPI command to set each function, send:

```
FUNC:RES 10000,(@100:107);TEMP RTD, 92,(@108:115);VOLT (@116,123)
```

This sets the first eight channels to measure resistance, the next eight channels to measure temperature and the next eight 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 programming commands for the VT1422A. 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 VT1422A'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 be dangerous. Make certain that the process is put into a safe state before execution of a controlling algorithm is halted.**

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 VXI*plug&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. Do 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 VT1422A allocates twice the memory required to store the array. When the ALG:ARRAY command is sent, the new values for the array are loaded into the second space for this array. When the ALG:UPDATE or ALG:UPDATE:CHANNEL commands are sent, the VT1422A 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 Agilent/HP E1406A command module, the Definite Length Arbitrary Block <*array_block*> parameter must always use "Big Endian" (Motorola) byte ordering for the packed 64-bit float values.
- The <*alg_name*> paragraph 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 VXI*plug&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 Agilent/HP E1406A command module, the Definite Length Arbitrary 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 VXI*plug&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 VXI*plug&play* function:** hpe1422_downloadAlg(...). This function loads an algorithm from a file. The VXI*plug&play* Soft Front Panel program allows algorithms to be created and tested on-line, then to be stored as 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 it is desired to change <*alg_name*> later while it is running. The value can range up to about 23k words (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 VT1422A 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 a replacement algorithm is sent later (it must be sent without the <*swap_size*> parameter), it will be placed in "space2." An ALG:UPDATE command must be sent for execution to switch from the original to the replacement algorithm. If the algorithm for <*alg_name*> is again changed, 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 VT1422A 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 VT1422A'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, it is recommended 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)>

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;Ø (where "Ø" is a null byte)

Note For Block Program Data, the Algorithm Parser requires that the <*source_code*> data end with a null (0) byte. The null byte must be appended 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 receives an End Identifier with the last data byte. Use this form only when it is certain that the 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;Ø (where "Ø" is a null byte)

Note For Block Program Data, the Algorithm Parser requires that the <*source_code*> data end with a null (0) byte. The null byte must be appended 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

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
```

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, not '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 an algorithm is re-defined (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 redefine all algorithms that will replace <*alg_name*> by swapping them before INIT is sent. This insures that all channels referenced in these algorithms will be available after INIT.
2. If an algorithm is redefined (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. Do 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 VXI*plug&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 VXI*plug&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 the specified algorithm to be executed 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 multiple ALG:SCAN:RATIO commands to be sent with their effect synchronized with an ALG:UPDATE command.
2. ALG:SCAN:RATio places a variable update request in the Update Queue. More update requests cannot be placed 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 VXI*plug&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 VXI*plug&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 VXI*plug&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 multiple ALG:STATE commands to be sent with a synchronized effect.
2. ALG:STATE places a variable update request in the Update Queue. Do 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 be dangerous. Make certain that the process is in a safe state before halting the execution of a controlling algorithm.**

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 VXI*plug&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 VXI*plug&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 VT1422A 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 VXI*plug&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 487 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 VT1422A'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 17 μs. 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.
- Values are generated for <*range*>, <*offset*>, and <*func_data*> with a program supplied with the VT1422A. It is provided in C-SCPI and BASIC forms. For full information, see Appendix F "Generating User Defined Functions" on page 487.

- The <*range*> and <*offset*> parameters 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 VXI*plug&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 (2.5 µs 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 7-1). Also note that not all outputs will occur at the same time but will take approximately 10 µs 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 <*delay*> is set to less time than is required for the Input + Update + Calculate, ALG:OUTP:DELAY? will report the set time, 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.

- To set the time from trigger to the beginning of OUTPUT, use the following procedure. After defining all other 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, reissue the ALG:OUTP:DEL AUTO and ALG:OUTP:DEL? queries to determine the new delay that includes the added algorithm.

- **When Accepted: Before INIT only.**

- ***RST Condition:** ALG:OUTP:DELAY AUTO

- **Send with VXI*plug&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 VXI*plug&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 7-1 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.

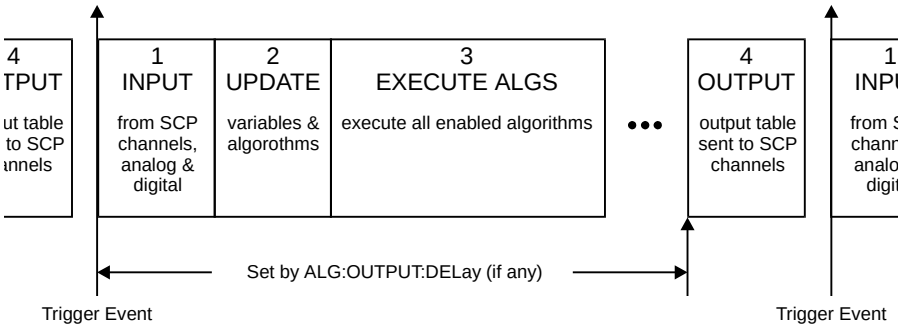


Figure 7-1. Updating Variables and Algorithms

- **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 VXI*plug&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 7-1) following the channel's change of digital state. This command is useful to synchronize multiple VT1422As when all variable updates are 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 VT1533A: Iccc.Bb for VT1534A: 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 7-1) 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 VT1422A begins to closely monitor the state of the update channel and cannot 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 VXI*plug&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 may need to be performed 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 it is known that fewer updates may be needed, 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

A maximum of eight variables will need to be updated 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 will be changed in one update after the algorithms are running. It may be necessary to 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 UPDATE window. The type is **int16**
- **Send with VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

ARM

The ARM subsystem is only useful when the TRIGer:SOURce is set to TIMer. With the VT1422A, 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 7-2 shows the overall logical model of the Trigger System.

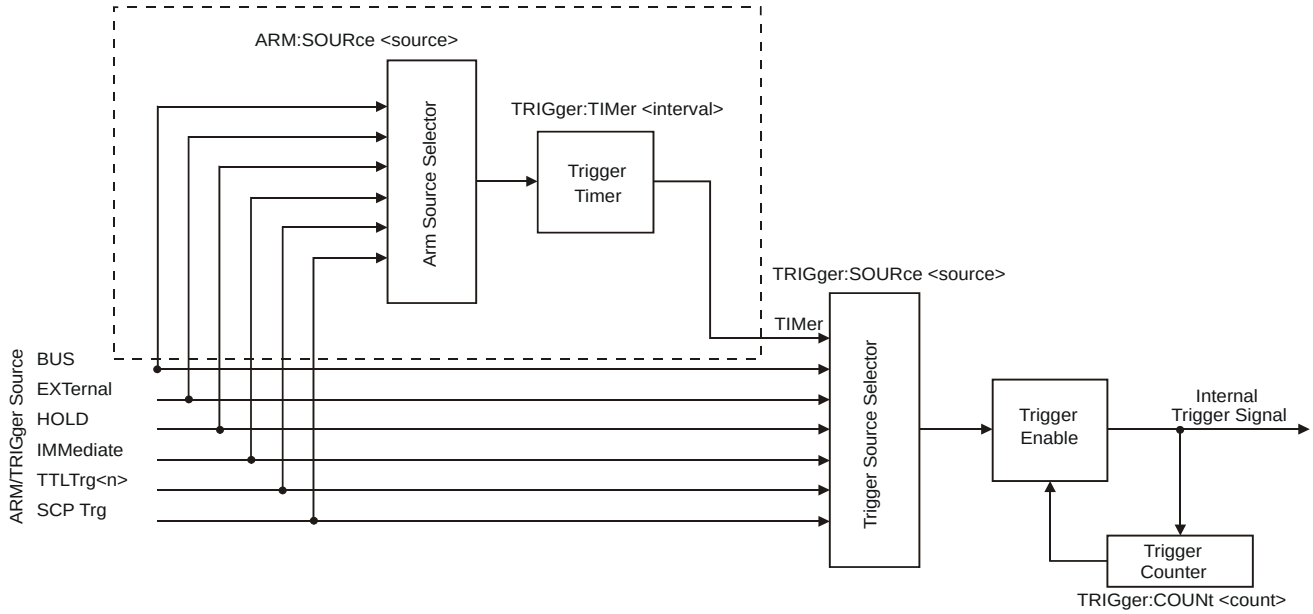


Figure 7-2. Logical Trigger Model

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 VXI*plug&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 SCP Breadboard)
TTLTrg<n>	The VXIbus TTLTRG lines (n=0 through 7)

- See note about ARM subsystem on page 256.
- 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, simply 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 VXI*plug&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 VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

Usage

```
ARM:SOUR?                         An enter statement return arm
                                  source configuration
```

CALCulate

The Calculate subsystem allows post-processed temperature compensation to be done for voltage readings of the on-board reference thermistor of the VT1586A and for E, J, and T thermocouple types. These commands will normally not be used, instead, the conversions will be done by the DSP on the VT1422A or by the VXI*plug&play* driver (for VT1529B).

These commands may be chosen if using the VT1586A with the VT1529B and do not have a PC fast enough to keep up with the real-time conversions done in the VXI*plug&play* driver (*hpe1422_readFifoPost_Q()* call). In this case, program the VT1422A to do voltage measurements, not temperature measurements, on the reference thermistor, the thermistor's excitation voltage and the thermocouple channels, then use the CALCulate subsystem commands to perform the conversions off-line.

Subsystem Syntax

CALCulate
:TEMPerature
:THERmistor? <*thr_volts*>,<*exc_volts*>[,*resistance*]
:TCouple? <*type*>,<*thr_temp*>,<*volt_array*>

CALCulate:TEMPerature:THERmistor?

CALCulate:TEMPerature:THERmistor? <*thr_volts*>,<*exc_volts*>[,*resistance*] calculates the temperature of the VT1586A reference thermistor from the voltage measured across it, the excitation voltage applied and the sum of the resistor values in the divider circuit (see Figure 5-5 on page 173 for the recommended circuit for measuring the VT1586A reference thermistor using the VT1529B). The temperature is returned in ºC.

This command is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
thr_volts	numeric (float32)	-16.0 to 16.0	volts
exc_volt	numeric (float32)	-15.99 to +15.99	volts
resistance	numeric (float32)	1800 to 520,000 default value = 400,000	ohms

Comments

- **Returned Value:** thermistor temperature in ºC. The data type is **float32**.
- **Related Commands:** CALC:TEMP:TC?
- **Send with VXI*plug&play* Function:** hpe1422_cmdReal64_Q(...)

Usage CALC:TEMP:THER? 0.0617,5.0

Compute thermistor temperature with thermistor voltage = 0.0617 V, excitation voltage = 5.0 V and resistance = 400 kΩ

CALCulate:TEMPerature:TCouple?

CALCulate:TEMPerature:TCouple? <*type*>,<*thr_temp*>,<*volt_array*> converts an array of thermocouple output voltages into temperatures. The thermocouple type is specified in the <*type*> parameter and the isothermal reference temperature is specified in the <*thr_temp*> parameter. The temperature values are returned in ºC.

This command is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	discrete (string)	E | J | T	none
thr_temp	numeric (float32)	-50 to +100	ºC
volt_array	see description	depends on thermocouple type	volts

Comments

- The <*volt_array*> is an IEEE-488.2 definite length block of **double** (float64) values, which is the array of thermocouple output voltages.
- **Returned Value:** array of temperatures in ºC. The data type is double (float64).
- **Related Commands:** CALC:TEMP:THER?
- **Send with VXI*plug&play* Function:** hpe1422_cmdTempTC_Q(...)

Usage

The following code segment shows the VXI*plug&play* command sequence:

```
ViSession vi;
ViStatus errStatus;
ViReal64 degC;   /* thermistor temperature */
ViReal64 voltArray[64];
ViReal64 tempArray[64];
ViInt32 howManyRead;
ViString command;

/* set up thermocouple measurement. In command below, channel 10001 is the
   thermistor channel and channel 10000 is the excitation voltage used for the
   thermistor measurement.
 */
errStatus = hpe1422_cmd(vi, "SENS:REF THER,5000,(@10000),(@10001)");

/* set up thermocouple channels */
errStatus = hpe1422_cmd(vi, "SENS:FUNC:TEMP:POST TC,J,(@10002:10031)");

/* specify thermistor as reference for thermocouples */
```

```
errStatus = hpe1422_cmd(vi, "SENS:REF:CHAN (@10001),(@10002:10031)");

/* Specify those channels to be scanned */
errStatus = hpe1422_cmd(vi, "ROUT:SEQ:DEF (@10001:10031,10000)");

errStatus = hpe1422_cmd(vi, "INIT");

/* read voltages back into array */
errStatus = hpe1422_cmd(vi, "FORM PACK,64");

/* Get unconverted data from the FIFO */
errStatus = hpe1422_readFifo_Q(vi, 32, 64, voltArray, &howManyRead);

/* build command string for thermistor temp calculation,
 voltArray[0] is thermistor voltage and voltArray[31] is
 excitation voltage. Bridge resistance is 397,000 ohms.
 */
sprintf(command, "CALC:TEMP:THER? %lf,%lf,%lf\n", voltArray[0],voltArray[31],
           397000);

/* get the thermistor temperature */
errStatus = hpe1422_cmdReal64_Q(vi, command, &degC);

/* Get unconverted thermocouple data from the FIFO */
errStatus = hpe1422_calcTempTC_Q(vi, "J", degC, 30, &voltArray[1], tempArray,
                                 &howManyRead);
```

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 five of its ranges. The multimeter also determines the value of the VT1422A'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"—Which has three levels (see Figure 7-3):

   -- "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 VT1422A 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 the 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 VT1539A 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.

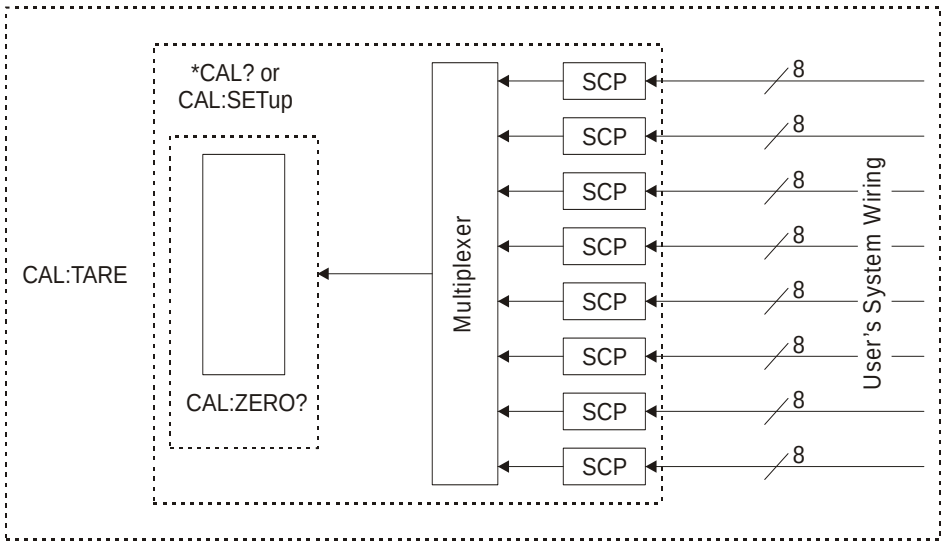


Figure 7-3. Levels of Working Calibration

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 VXI*plug&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 five following values: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, 16 V dc. This parameter 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 16 V range where the output is 10 V.
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:VAL:VOLT, STOR ADC
- **Send with VXI*plug&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 (0.25, 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 VT1529A/B. 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 VT1422A and the SCPs it contains. They do not calibrate Remote Signal Conditioning Units like the VT1529A/B. CAL:REMote must be used 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 a VT1539A 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 failure codes below.
-1	Couldn't start remote calibration	Query the Error Queue (SYST:ERR?) See error messages starting on page 453.

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 VXI*plug&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 VT1422A 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 VT1422A. Values for channel positions that are not installed are "place holders."

- ***RST Condition:** Stored calibration constants are unchanged

- **Send with VXI*plug&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 1,024 float32 values that represent a gain and offset pair for each of 512 possible remote channels. The block is always 1,024, float64 values (8192 bytes) regardless how many RSCUs are actually connected to the VT1422A. Values for channel positions where RSCUs are not installed are set to 0.000.
- **Returned Value:** the 1,024 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 VXI*plug&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 a VT1539A SCP, a 3007 "Invalid signal conditioning plug-on" error is generated.

Note

An RSCU's Flash Memory has a finite lifetime of approximately 10,000 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 the RSCUs are to be calibrated often, especially after a line power failure, they need not be stored in Flash memory.
- **Send with VXI*plug&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 399.

Note that the scope of the *CAL? and CAL:SETup commands is limited to the VT1422A and the SCPs it contains. They do not calibrate Remote Signal Conditioning Units like the VT1529A/B. CAL:REMote? must be used 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 there are multiple VT1422As in a system, a CAL:SET operation can be started on each and then a CAL:SET? command can be executed to complete the operation on each instrument.
- **Related Commands:** CAL:SETup?, *CAL?
- **When Accepted:** Not while INITiated
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
CAL:SET          start SCP Calibration on 1st VT1422A
:                start SCP Calibration on more VT1422As
CAL:SET          start SCP Calibration on last VT1422A
CAL:SET?         query for results from 1st VT1422A
:                query for results from more VT1422As
CAL:SET?         query for results from last VT1422A
```

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 454 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 VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

Usage See CAL:SETup.

CALibration:STORe

CALibration:STORe *<type>* stores the VT1422A'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 VT1422A's Flash Memory has a finite lifetime of approximately 10,000 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 29).
- 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 VXI*plug&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 153.

Note for RSC Units

CAL:TARE does not remove offsets in an RSCs field wiring. For the VT1529A/B, the SENS:STR:UNSTrained value is the equivalent of the tare offset. Any offset in the analog signal line between the VT1529A/B SCP and an RSC is accounted for during the CAL:REMote? operation.

Notes For Thermocouples

1. CAL:TARE must not be used on field wiring that is made up of thermocouple wire. The voltage a thermocouple wire pair generates cannot be removed by introducing a short anywhere between its junction and its connection to an isothermal panel (either the VT1422A'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 a thermocouple wire pair is inadvertently CAL:TARE’d, use CAL:TARE:RESET to reset all tare constants to zero.
2. CAL:TARE should be used to compensate wiring offsets (copper wire, not thermocouple wire) between the VT1422A 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 +0.1 volt offset with 0 volts (a short) applied at the Unit Under Test (UUT). Before CAL:TARE the module would return a reading of 0.1 volts 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.

- The CAL:TARE command may be issued to several VT1422As 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 the range or gain setup is later changed, 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 OTD is turned off by the program before executing CAL:TARE, the application should also wait one 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	0.40104	0.20009	0.04970
4	0.82101	0.10101	0.05007	0.01220
1	0.23061	0.02721	0.01317	0.00297
0.25	0.07581	0.00786	0.00349	0.00055
0.0625	0.03792	0.00312	0.00112	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 VT1422A's Flash Memory has a finite lifetime of approximately 10,000 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 takes the VT1422A 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 seventeen 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 VXI*plug&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 VXI*plug&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 453. 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 VXI*plug&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 7500 Ω nominal reference resistor value or a -222 'Data out of range' error will be generated. If this error occurs, verify the external measurement equipment and run *TST? on the VT1422A.
- <*ref_ohms*> may be specified in kOhm (kohm).
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:CONF:RES, CAL:STORE ADC
- **Send with VXI*plug&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 0.0625 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 the external reference voltmeter is in excess of 4% error from nominal voltage, verify the voltmeter and execute *TST? on the VT1422A.

- <*ref_volts*> may be specified in millivolts (mV).
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:CONF:VOLT, CAL:STORE ADC
- **Send with VXI*plug&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 the 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 453

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 VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

Usage

```
CAL:ZERO?
enter statement here                    returns 0 or -1
```

DIAGnostic

The DIAGnostic subsystem allows special operations to be performed 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 VT1531A and VT1532A 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 the 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 VXI*plug&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 the 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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VT1529A/B to verify its measurement paths by measuring either the internal calibration source or an internal short for all 32 channels. A matching scan list must be defined, the instrument triggered, and the results read 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

- The <*source*> parameter specifies the source to measure. NORMal configures all inputs to measure user inputs. SHORT specifies the internal calibration short. SOURce specifies the internal 100 mV calibration source.
- The <*mode*> parameter: 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 0 V, odd channels are 0.1 V.
- The <*ch_list*> parameter specifies which VT1529A/B to configure. Specifying any channel on the a VT1529A/B configures all channels on the unit.
- DIAG:CONN NORM,ALL,(@<*ch_list*>) must be executed to reset units for normal measurements.
- **Related Commands:** [SENSe:]DATA:FIFO?
- ***RST Condition:** DIAG:CONN ALL, NORM for all VT1529A/B channels
- **Send with VXI*plug&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 VT1422A. Use the "[SENSe:]FUNCtion:CUSTom" on page 340 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": $(f_{outp1} - f_{outp0})/(V_{in1} - V_{in0})$
- The <*offset*> parameter 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 VXI*plug&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 VT1422A. Use the "[SENSe:]FUNCtion:CUSTom" on page 340 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": $(f_{outp1} - f_{outp0})/(V_{in1} - V_{in0})$

- The <*offset*> parameter 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 VXI*plug&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 VT1422A. Contact a VXI Technology System Engineer for more information on Custom Engineering Unit Conversion for specific applications.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
table_range	numeric (float32)	0.015625 | 0.03125 | 0.0625 | 0.125 | 0.25 | 0.5 | 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

- The <*table_block*> parameter 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 VXI*plug&play* function *hpe1415_sendBlockUInt16(ViSession vi, ViString cmd_str, ViInt32 table[ ], ViInt32 size)* adds the header automatically.
- The <*table_range*> parameter specifies the range of voltage that the table covers (from -<*table_range*> to +<*table_range*>).
- The <*ch_list*> parameter specifies which channels may use this custom EU table.
- **Related Commands:** [SENSe:]FUNCtion:CUSTom
- ***RST Condition:** All custom EU tables erased.
- **Use VXI*plug&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 VT1422A
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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

the 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 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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VT1422A'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 one minute to allow channels to settle, perform the calibration, and then re-enable OTD.

- **Send with VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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 VXI*plug&play* Function:** hpe1422_cmdInt32_Q(...)

Usage

```
DIAG:QUERY:SCPREAD? 258          read SCP #4 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. A custom format can be designed for information storage. For example, the 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 10,000 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

- The <*channel*> parameter 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 VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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 VT1539A 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

- The <*test_num*> parameter specifies the test to perform. For explanations of the test numbers, See “DIAGnostic:TEST:REMote:SELFtest?” on page 288.
- The <*iterations*> parameter specifies the number of times to perform a test.
- The <*channel*> parameter 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 453.

The data type for this returned value is **int16**.

- **Send with VXI*plug&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 VT1539A SCP. An example of an RSCU is the VT1529A/B Remote Strain Bridge Conditioning unit.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- The <*channel*> parameter 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 453.

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 ±45 mV 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 ±45 mV 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 eight 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 the VT1529A/B were tested at channel 10000 and the seventh trigger point had bad data, the failure would be logged as 10039).

Test 5: This tests the filter settings on each bank (of eight 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 VT1529A/B 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 VT1529A/B (is cable connected?)
  b. Invalid channel number or multiple channels specified
  c. Not enough memory to allocate internal arrays to hold data
  d. VT1422A is currently performing a calibration operation.
  e. VT1422A is currently performing a measurement operation.

- **Related Commands:** *TST?, *CAL?, CAL:REMote?, SYST:ERR?

- **Send with VXI*plug&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,E1422A,US34000478,A.04.00,Thu Aug 5 9:38:07 MDT 1994
- The data type is **string**.
- **Use VXI*plug&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 Agilent/HP E1405B/06A 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 233. 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 Workstation BASIC and 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          1 megabyte (MB) or 262,144 readings
MEM:VME:STAT ON
  x
  °   (set up VT1422A for scanning)
  x
TRIG:SOUR IMM                  let unit trigger on INIT
INIT                           program execution remains here until
                               VME memory is full or the VT1422A 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 an external trigger (EXT trigger) is used. 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 Workstation BASIC and 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 233. 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 something else had been set before executing *TST?. If the FIFO needs to read 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 7-1. Data Formats

- **Send with VXI*plug&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 VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

Usage

FORMAT?	*Returns REAL, +32 | REAL, +64 | PACK, +64 | ASC, +7*

INITiate

The INITiate command subsystem moves the VT1422A 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 VXI*plug&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

- The <*cutoff_freq*> parameter may be specified in kilohertz (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*>.
- For the VT1529A/B, this command affects an 8-channel bank, not a single channel.
- 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 VT1529A/B is set to 10 Hz.
- **Send with VXI*plug&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 15622
```

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

- The <*channel*> parameter 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 VT1529A/B is set to 10 Hz.
- **Send with VXI*plug&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 only a 20 kHz, single-pole low pass filter. 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.
- For the VT1529A/B, this command affects an 8-channel bank, not a single channel.
- **When Accepted:** Not while INITiated
- ***RST Condition:** ON
- **Send with VXI*plug&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**.
- The <*channel*> parameter must specify a single channel only.
- **Send with VXI*plug&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 SCP or 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 the SCP manual for specific information on the SCP being 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 VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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 VT1529A/B's Wagner Voltage connection is only controlled by the command "[SENSe:]STRain:BRIDge[:TYPE]" on page 362.

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 VXI*plug&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 VT1529A/B.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 163	none

Comments

- The <*channel*> parameter must specify a single channel only.
- **Returned Value:** Returns FLO or WV. The data type is **string**.
- **Related Commands:** INP:LOW
- **Send with VXI*plug&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 the 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 VXI*plug&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

- The <*channel*> parameter 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 the SCP's User's Manual to determine its capabilities.
- **Returned Value:** returns "NORM" or "INV." The type is **string**.
- **Send with VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

MEASure

The MEASure subsystem 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 VT1422A to measure the bridge excitation voltage at each channel in <*ch_list*> and starts a measurement scan. It averages 32 measurements 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 for use in user defined conversion processes. The command returns a single value which is the number of readings sent to the FIFO.

Note The maximum excitation voltage the VT1422A can sense through the VT1529A/B's excitation sense path is 16 volts (±8 V dc centered about the Gnd terminal). If a higher excitation voltage is supplied through the VT1529A/B, 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 VT1422A 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, it will be necessary 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 VT1529A/B. If executed on channels connected to other strain SCPs, a 3007 "Invalid signal conditioning plug-on" error message will be generated
- This command 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 392 μs
- 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 a strain channel is changed 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 VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

Usage

```
MEAS:VOLT:EXC?  (@10000:10931)
```
determine excitation voltage on 128 strain channels through VT1539As in SCP positions 0 and 1

```
SENS:DATA:FIFO:RESET
```
As in most cases, since the values have been sent to the strain EU conversion, the individual excitation values do not need to be seen.

MEASure:VOLTage:UNSTrained?

MEASure:VOLTage:UNSTrained? (@<*ch_list*>) This command automatically configures the VT1422A to measure the bridge output voltage at each channel in <*ch_list*> and initiates a measurement scan. It averages 32 measurements 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 to be viewed later if desired. 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 VT1422A 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, it will be necessary 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 VT1529A/B. If executed on channels connected to other strain SCPs, a 3007 "Invalid signal conditioning plug-on" error message will be generated.

- This command 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 392 μs
- 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 a strain channel is changed back to the strain function with a SENS:FUNC:STRAIN... command, the unstrained values for these channels 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 VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

Usage

```
MEAS:VOLT:UNST? (@10000:10931)      determine unstrained voltage on 128
                                    strain channels through VT1539As in
                                    SCP positions 0 and 1
SENS:DATA:FIFO:RESET                As in most cases, since the values have
                                    been sent to the strain EU conversion,
                                    the individual unstrained values do not
                                    need to be seen.
```

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 Agilent/HP E1405B/06A command module.

Use Sequence

```
*RST
MEM:VME:ADDR #H300000
MEM:VME:SIZE #H100000          1 MB or 262,144 readings
MEM:VME:STAT ON
  *
  * (set up VT1422A for scanning)
  *
TRIG:SOUR IMM                  let unit trigger on INIT
INIT
*OPC?                          program execution remains here until
                               VME memory is full or the VT1422A 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 an external trigger (EXT trigger) is used. 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 Agilent/HP E1405B/06A command module.
- The default (if MEM:VME:ADDR not executed) is 240000_{16}.
- The *<A24_address>* parameter 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 Agilent/HP E1405B/06A command module, the first VT1422A 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 Agilent/HP E1405B/06A 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 Agilent/HP E1405B/6A command module.
- <*mem_size*> may be specified in decimal, hex (#H), octal (#Q), or binary (#B).
- <*mem_size*> should be a multiple of four 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 kilobytes (kB) 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 Agilent/HP E1405B/06A 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 Agilent/HP E1405B/06A 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 Agilent/HP E1405B/06A 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 VT1505A 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 VT1532A Current Output SCP.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
amplitude	numeric (float32)	MIN | 30E-6 | MAX | 488E-6	A dc
ch_list	channel list (string)	100 - 163	none

Comments

- Select 488E-6 (or MAX) for measuring resistances of less than 8000 Ω. Select 30E-6 (or MIN) for resistances of 8000 Ω 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 (488 μA)	for RTD,85 | 92 and THER,2250
MIN (30 μA)	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 VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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 VXI*plug&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 a VT1505A 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

- The <*channel*> parameter 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 VXI*plug&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 VXI*plug&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

- The <*channel*> parameter must specify a single channel.
- **Returned Value:** returns one of NORM or INV. The type is **string**.
- **Send with VXI*plug&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 VT1529A/B 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 VT1529A/B can be specified in <*ch_list*>. This is because a single resistor is used to shunt each of a VT1529A/Bs 32 channels. The <*ch_list*> may specify one channel on each of several VT1529A/Bs.
- **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 VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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 VT1529A/B channel 0124*

OUTPut:SHUNt:SOURce

OUTPut:SHUNt:SOURce <*select*>,(@<*ch_list*>) selects the source of the bridge shunt resistance for a VT1529A/B Remote Strain Bridge Conditioning unit. The VT1529A/B 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 VT1529A/B strain SCP, a 3007 "Invalid signal conditioning plug-on" error is generated.
- Only one channel on each VT1529A/B needs to be specified since a single resistor is used for all channels in the module.
- **When Accepted:** Not while INITiated

- **Related Commands:** OUTPut:SHUNt…, SENSe:FUNCtion:STRain…, [SENSe:]STRain…
- ***RST Condition:** OUTP:SHUNT:SOURCE INT on all VT1529A/B channels
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
OUTP:SHUNT:SOUR EXT,(@10000,10800)
```

select user supplied shunt resistor on VT1529As 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 VT1529A/B Strain channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- The <*channel*> parameter must specify a single channel only and since there is a single shunt resistor for all channels on a VT1529A/B, it can be any channel on the VT1529A/B.
- If <*channel*> specifies a non VT1529A/B 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 VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

Usage

```
OUTPUT:SHUNT:SOURCE? (@11600)
```

Check source of shunt resistance on VT1529A/B 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 VT1422A 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
OUTP:TTLT:SOUR TRIG
```

toggle TTLTRG line every time module is triggered (use to trigger other VT1422As)

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 VXI*plug&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 VXI*plug&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 VXI*plug&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 the SCP's User's Manual to determine its capabilities.
- PASSive configures the digital channel/bit to be passive (resistor) pull-up to allow more than one output to wire-OR'd 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 VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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 VT1511A for example).

Note This command is not used to set output voltage on SCPs like the VT1531A 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 VT1511A
- **Send with VXI*plug&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 a VT1511A SCP, the query will return zero.

Comments

- The <*channel*> parameter must specify a single channel only.
- **Returned Value:** Numeric, one of 0, 1, 2, 5, or 10 for VT1511A SCP, 3.9 for non- programmable VT1506A/07A SCPs. Data type is **float32**.
- **Related Commands:** OUTP:VOLT:AMPL
- **Send with VXI*plug&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 VT1422A'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. (All digital and analog output channels must be specified in the algorithm.) Channels specified with ROUT:SEQ:DEF will be scanned each time the VT1422A 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 defined 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

- The <*ch_list*> parameter must contain analog input channels only.
- Multiple occurrences of the same channel number in a ROUT:SEQ:DEF command generate multiple occurrences of that channel in the analog input scan list. This is useful when a particular channel needs additional settling time. However, a scan list can contain only 32 total channels per VT1529A/B. This means that for each VT1529A/B duplicate channel reference in a scan list, some other channel on that same VT1529A/B must be left out of the scan.
- When the same channel is referenced both by an algorithm and the ROUT:SEQ:DEF command, only a single reference to the channel is added to the analog input scan list. For each scan operation, this channel's measured value is made available to both the algorithm and to the FIFO/CVT.
- 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 interaction, a scan list should be ordered so that 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. See “Channel List (Relative Form)” on page 232 for a discussion of the syntax. 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 to FIFO 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.

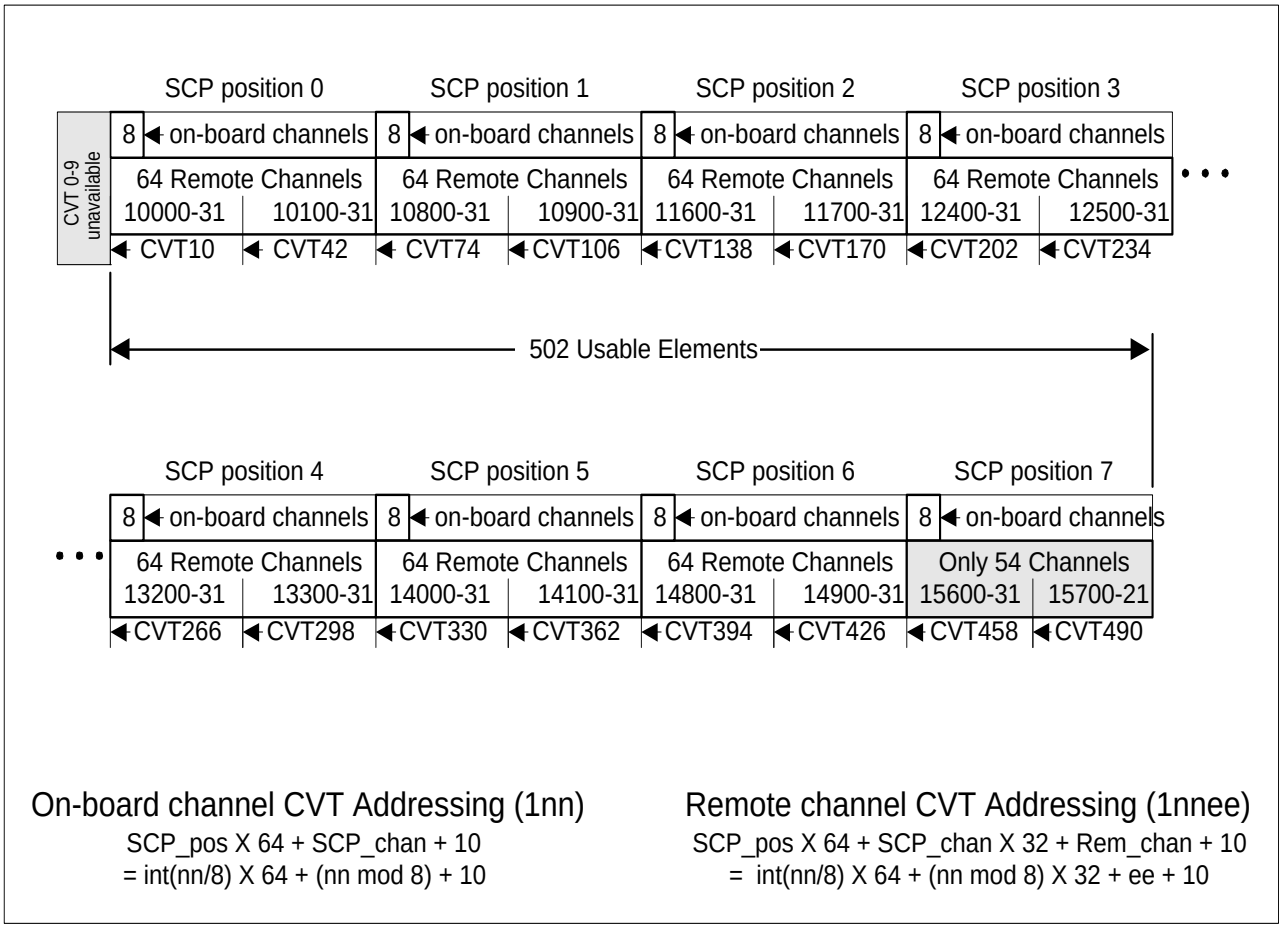


Figure 7-4. Channel Number vs. CVT Element

- 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 VXI*plug&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 eight 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 VXI*plug&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 VXI*plug&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 VXI*plug&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 157

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 µs. The resolution for <*interval*> is 2.5 µs.
- 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

SAMPLE:TIMER 150E-6 — *Pace measurements at 150 µs 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 VXI*plug&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, and temperature. Sense commands allow one to configure and extract data from the A/D-EU conversion portion of the instrument (see “INPut” subsystem on page 297 for input signal conditioning.

Subsystem Syntax

```
[SENSe:]
    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>)
            :HVOLtage [<range>,](@<ch_list>)
            :REFerence [<range>,](@<ch_list>)
            :TC <type>,[<range>,](@<ch_list>)
        :FREQuency (@<ch_list>)
        :HVOLtage (@<ch_list>)
        :RESistance <excite_current>,[<range>,](@<ch_list>)
        :STRain
            :FBENding [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :FBPoisson [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :FPOisson [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :HBENding [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :HPOisson [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            [:QUARter] [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :Q120 [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :Q350 [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
            :USER [<range>,](@<ch_list>)
                :POST [<range>,](@<exc_ch>),(@<ch_list>)
```

```
    :TEMPerature <sensor_type>,<sub_type>,[<range>,](@<ch_list>)
        :POST TC,<sub_type>,[<range>,](@<ch_list>)
    :TOTalize  (@<ch_list>)
    :VOLTage[:DC]  [<range>,](@<ch_list>)
REFerence  <sensor_type>, [<sub_type>,](@<ch_list>)
    :POST THERmistor,<res>,[<range>],(@<exc_ch>),(@<thr_ch>)
    :CHANnels (@<ref_channel>),(@<ch_list>)
        :POST (@<ref_channel>),(@<ch_list>)
    :TEMPerature  <degrees_celsius>
        :POST  <degrees_celsius>,(@<ch_list>)
    :THERmistor
        :RESistance
            :POST? <resistance>,(@<thr_list>)
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:]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 the latest values from algorithms and/or analog scans to be "viewed."

- 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 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 323 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 233.
  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 VXI*plug&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 VXI*plug&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 233. 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 7-1 on page 294 for actual values for each data format).

- **Related Commands:** SENSE:DATA:FIFO:HALF?, ROUT:SEQ:DEFine
- ***RST Condition:** FIFO is empty
- **Use VXI*plug&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 VXI*plug&play* function:** hpe1422_sensDataFifoCoun_Q(...)
- **Send with VXI*plug&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 VXI*plug&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, an 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 233. 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 7-1 on page 294 for actual values for each data format).

- **Related Commands:** DATA:FIFO:COUNT:HALF?
- ***RST Condition:** FIFO buffer is empty
- **Send with VXI*plug&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 indicate 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 VXI*plug&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 VXI*plug&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 233. 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 7-1 on page 294 for actual values for each data format).

- **Related Commands:** DATA:FIFO:COUNT?
- ***RST Condition:** FIFO buffer empty
- **Use VXI*plug&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 VXI*plug&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)	0.001 to 1 (0.001 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 the SCP's User's Manual for its capabilities.
- **Related Commands**: SENSe:FUNCtion:FREQuency
- ***RST Condition:** 0.001 s
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENS:FREQ:APER .01,(@144)                set channel 44 aperture to 10 ms
```

[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 the 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 VXI*plug&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 VT1533A SCP senses 8 digital bits on each channel specified by this command. The VT1534A 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

To set second 8-bits of VT1533A at SCP position 4 and upper 4-bits of VT1534A 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 a VXI Technology System Engineer for more information on Custom Piecewise Engineering Unit Conversion for specific applications.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see first comment	V dc
ch_list	channel list (string)	100 - 15731	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 150.
- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error -222 "Data out of range." Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If using amplifier SCPs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
DIAG:CUST:MXB slope,offset,(@116:123)     send M*X+B to VT1422A for chs 16-23
SENS:FUNC:CUST 1,(@116:123)               link custom EU with chs 16-23
INITiate then TRIGger module
```

[SENSe:]FUNCtion:CUSTom:HVOLtage

[SENSe:]FUNCtion:CUSTom:HVOLtage [<*range*>,](@<*ch_list*>) links VT1529B high-level input (DCV measurement) channels with the custom Engineering Unit Conversion table loaded with the DIAG:CUST:MXB or DIAG:CUST:PIECE commands.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see first comment	V dc
ch_list	channel list (string)	10000 - 15731	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 150.
- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error -222 "Data out of range." Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- **When Accepted:** Not while INITiated
- **Related Commands:** DIAG:CUST:…
- ***RST Condition:** all custom EU tables erased
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
DIAG:CUST:MXB slope,offset,(@10000)      send M*X+B to VT1422A for ch 10000
SENS:FUNC:CUST:HVOL 1,(@10000)           link custom EU with ch 10000
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	V dc
ch_list	channel list (string)	100 - 163	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 150.
- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If using amplifier SCPs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc 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 VXI*plug&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 an EU table to be used that is custom matched to thermocouple wire that has been characterized. Contact a VXI Technology System Engineer for more information on Custom Piecewise Engineering Unit Conversion for specific applications.

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	V dc
ch_list	channel list (string)	100 - 163	none

Comments

- See “Creating and Loading Custom EU Conversion Tables” on page 150.
- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If using amplifier SCPs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc or an input out-of-range condition will exist.
- The <*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 VXI*plug&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 the 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENS:FUNC:FREQ  (@144)                       set channel 44's to frequency
```

[SENSe:]FUNCtion:HVOLtage

[SENSe:]FUNCtion:HVOLtage [<*range*>,](@<*ch_list*>) links the specified VT1529B channels to return dc voltage on the high-level input (Excitation Sense + and -) pins on the RJ-45 connector for the channel.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see comments	V dc
ch_list	channel list (string)	10000 - 15731	none

Comments

- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- The VT1529A/B has a fixed gain of 32. Keep this in mind when <*range*> is set.
- See “DCV Measurement Command Sequence” on page 177.
- **When Accepted:** Not while INITiated
- **Related Commands:** CAL:REMote?
- ***RST Condition:** SENSE:FUNC:VOLT AUTO on all channels and bridges set to FBEN on all strain channels.
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
FUNC:HVOL (@10000:10001)
```

Channels 0 - 1 measure high-level input voltage in auto-range (defaulted)

[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	V dc
ch_list	channel list (string)	100 - 163	none

Comments

- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If using amplifier SCPs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc 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 not 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 and microamps.
- 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 109.
- **When Accepted:** Not while INITiated
- **Related Commands:** OUTP:CURR, *CAL?
- ***RST Condition:** SENSE:FUNC:VOLT AUTO on all channels and bridges set to FBEN on all strain channels
- **Send with VXI*plug&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 109.

Note When the SENS:FUNC:STR:<*bridge_type*> command is used with VT1529A/B 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 VT1529A/B 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 VT1529A/B while some are exclusive to one or the other.

The following table relates the command syntax to bridge type. See the Strain SCP user's manual for bridge schematics and field wiring information.

Command	Bridge Type	Strain SCP and VT1529A/B Usage
:FBENding	Full Bending Bridge	Both VT1529A/B and SCPs
:FBPoisson	Full Bending Poisson Bridge	SCPs only
:FPOisson	Full Poisson Bridge	SCPs only
:HBENding	Half Bending Bridge	Both VT1529A/B and SCPs
:HPOisson	Half Poisson Bridge	SCPs only
[:QUARter]	Quarter Bridge (default)	Both VT1529A/B and SCPs. For VT1529A/B, selects Q350
:Q120	Quarter Bridge 120 Ω	VT1529A/B only
:Q350	Quarter Bridge 350 Ω	VT1529A/B only
:USER	Quarter Bridge with user installed resistor	VT1529A/B 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 Agilent/HP E1405B/06A. 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	V dc
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 VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If using amplifier SCPs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc or an input out-of-range condition will exist.
- The VT1529A/B has a fixed gain of 32. Keep this in mind when <*range*> is set.
- 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 AUTO on all channels and bridges set to FBEN on all strain channels
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

SENS:FUNC:STRAIN:QUAR 1,(@100,105,107)	*quarter bridge conversion for channels 0, 5, and 7*
FUNC:STRAIN:FBEN 1,(@10800:10931)	*full bridge conversion for VT1529A/B channels 0800 to 0931 (64 channels)*

[SENSe:]FUNCtion:STRain:FBENding:POST
[SENSe:]FUNCtion:STRain:FBPoisson:POST
[SENSe:]FUNCtion:STRain:FPOisson:POST
[SENSe:]FUNCtion:STRain:HBENding:POST
[SENSe:]FUNCtion:STRain:HPOisson:POST
[SENSe:]FUNCtion:STRain[:QUARter]:POST
[SENSe:]FUNCtion:STRain:Q120:POST
[SENSe:]FUNCtion:STRain:Q350:POST
[SENSe:]FUNCtion:STRain:USER:POST

[SENSe:]FUNCtion:STRain:FBENding:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain:FBPoisson:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain:FPOisson:POST [<*rng*>,](@e<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain:HBENding:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain:HPOisson:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain[:QUARter]:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>]
[SENSe:]FUNCtion:STRain:Q120:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain:Q350:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)
[SENSe:]FUNCtion:STRain:USER:POST [<*rng*>,](@<*exc_ch*>),(@<*ch_list*>)

A Note on Syntax: Although the post-processing 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*>:POST [<*range*>,](@<*ch_list*>)
links the post-processing strain EU conversion with the VT1529B channels specified by <*ch_list*> to measure the strain bridge output. See "Strain Measurement Command Sequence" on page 165.

Both the strain channels and excitation channel should appear in the scan list set by ROUT:SEQ:DEF.

Note When the SENS:FUNC:STR:<*bridge_type*>:POST command is used with VT1529A/B 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 VT1529A/B channels that use the SENSe:FUNCtion:STRain:<*bridge_type*> command.

These commands are valid only on the VT1529B and are available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

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 one minute on some platforms. 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	V dc
ch_list	channel list (string)	10000 - 15731	none

Comments

- The VT1529B provides the strain measurement bridges and their excitation voltage sources. <*ch_list*> specifies the voltage sensing channels that are to measure the bridge outputs.
- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- The VT1529B has a fixed gain of 32. Keep this in mind <*range*> is set.
- **When Accepted:** Not while INITiated

- **Related Commands:** CAL:REMote?,[SENSE:]STRAIN…
- ***RST Condition:** SENSE:FUNC:VOLT AUTO on all channels and bridges set to FBEN on all strain channels
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENS:FUNC:STRAIN:QUAR:POST  1,(@10000:10007)   quarter bridge conversion for VT1529B
                                               channels 10000 to 10007
SENSE:FUNC:STRAIN:FBEN:POST  1,(@10800:10931)  full bridge conversion for VT1529B
                                               channels 10800 to 10931
                                               (64 channels)
```

[SENSe:]FUNCtion:TEMPerature

[SENSe:]FUNCtion:TEMPerature <*type*>,<*sub_type*>,[<*range*>,](@<*ch_list*>) links channels to an temperature EU conversion based on the sensor specified in <*type*> and <*sub_type*>. Note: Do not include the thermocouple reference temperature channels in this command (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	V dc
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 (488 μA)	for RTD and THER,2250
MIN (30 μA)	for THER,5000 and THER,10000

- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If using amplifier SCPs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc or an input out-of-range condition will exist.

- The <*sub_type*> parameter: values of 85 and 92 differentiate between 100 Ω (@ 0°C) RTDs with temperature coefficients of 0.00385 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 kΩ (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 109.

- **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 on all channels and bridges set to FBEN on all strain channels

- **Send with VXI*plug&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:TEMPerature:POST

[SENSe:]FUNCtion:TEMPerature:POST TC,<*sub_type*>,[<*range*>,] (@<*ch_list*>) links VT1529B channels to a post-processing temperature EU conversion based on the sensor specified in <*type*> and <*sub_type*>. **Note**: Do not include the thermocouple reference temperature or thermistor excitation channels in this command (for that, use the SENS:REF:POST <*type*>,<*sub_type*>,<*range*>, (@<*exc_ch*>),(@<*thr_ch*>) command).

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
sub_type	discrete (string)	E | J | T	none
range	numeric (float32)	see comments	V dc
ch_list	channel list (string)	10000 - 15731	none

Comments

- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 generates an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- The VT1529B has a fixed gain of 32. Keep this in mind when <*range*> is set.
- See “Temperature Measurement Command Sequence” on page 174.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENS:REF:POST and SENS:REF:TEMP:POST
- ***RST Condition:** SENSE:FUNC:VOLT AUTO on all channels and bridges set to FBEN on all strain channels
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

Link two channels to the J type thermocouple temperature conversion

```
SENS:FUNC:TEMP:POST  TC,J,(@10002,10003)
```

Link channel 1 to measure reference temperature using 5k thermistor on VT1586A; excitation voltage is on channel 0

```
SENS:REF:POST  THER,5000,(@10000),(@10001)
```

[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 VXI*plug&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. On a VT1529B, this specifies the low-level input pins on the RJ-45 connector for the channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
range	numeric (float32)	see comments	V dc
ch_list	channel list (string)	100 - 15731	none

Comments

- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.

- If using amplifier SCPs or RSCs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc 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 109.
- **When Accepted:** Not while INITiated
- **Related Commands:** *CAL?, INPUT:GAIN…
- ***RST Condition:** SENSE:FUNC:VOLT AUTO on all channels and bridges set to FBEN on all strain channels
- **Send with VXI*plug&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	V dc
ch_list	channel list (string)	100 - 15731	none

Comments

- See “Linking Input Channels to EU Conversion” on page 109
- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- If using amplifier SCPs or RSCs, set them first and keep their settings in mind when specifying a range setting. For instance, if the expected signal voltage is to be approximately 0.1 V dc and the amplifier SCP for that channel has a gain of 8, <*range*> must be set no lower than 1 V dc 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 cannot be redefined.
- For <*type*> THERmistor, the <*sub_type*> parameter may be specified in ohms or kilohms.
- 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENSE:REF  RTD,92,(@120)        sense the reference temperature on
                                channel 20 using an RTD
SENSE:REF  THER,5000,(@113)     sense the reference temperature on
                                channel 113 using a 5k thermistor
```

[SENSe:]REFerence:POST

[SENSe:]REFerence:POST THERmistor,<*resistance*>,[<*range*>,] (@<exc_ch>),(@<*thr_ch*>) links the thermistor channel in <*thr_ch*> to the excitation voltage channel in <*exc_ch*> and to the reference junction temperature EU conversion. The excitation channel is automatically set to SENS:FUNC:HVOL and the thermistor channel is set to SENS:FUNC:VOLT. The excitation channel must be on the same VT1529B as the thermistor channel.

The reference junction temperature EU conversion assumes the divider circuit shown in Figure 5-5 on page 173. If a total resistance value other than 400 kΩ is used, the sum of the resistor values can be specified with the SENS:REF:THER:RES command.

Both the thermistor channel and the excitation channel should appear in the scan list set by the ROUT:SEQ:DEF command. If the excitation channel is also used for strain measurements, it only needs to appear once in the scan list.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
resistance	numeric (float32)	5000 (value is ignored)	ohm
range	numeric (float32)	see comments	V dc
exc_ch	channel	10000 - 15731 (one channel only)	
thr_list	channel list (string)	10000 - 15731	none

Comments

- The VT1422A has five ranges: 0.0625 V dc, 0.25 V dc, 1 V dc, 4 V dc, and 16 V dc. To select a range, simply specify the range value (for example, 4 selects the 4 V dc range). If a value is specified that is larger than one of the first four ranges, the VT1422A selects the next higher range (for example, 4.1 selects the 16 V dc range). Specifying a value larger than 16 causes an error. Specifying 0 selects the lowest range (0.0625 V dc). Specifying AUTO selects auto range. The default range (no range parameter specified) is auto range.
- The VT1529B has a fixed gain of 32. Keep this in mind when <*range*> is set.
- See “Temperature Measurement Command Sequence” on page 174.
- **Related Commands:** SENS:FUNC:TEMP:POST, SENS:REF:THER:RES:POST
- ***RST Condition:** No reference assigned
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENS:REF:POST  THER,5000,4,(@10000),(@10001)
```
sense the reference temperature on ch 10001; excitation is on ch 10000; measure on 4 V range

[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 - 163	none
ch_list	channel list (string)	100 - 163	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, 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:** No channels are linked to references.
- **Send with VXI*plug&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:CHANnels:POST

[SENSe:]REFerence:CHANnels:POST (@<*ref_channel*>),(@<*ch_list*>) links the thermocouple channel(s) specified by <*ch_list*> to the temperature reference (thermistor) channel specified by <*ref_channel*>. The reference channel must be on the same VT1529B as the thermocouple channels.

The ROUT:SEQ:DEF command must be used to include the thermocouple channels in the scan list.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
ref_channel	channel list (string)	10000 - 15731	none
ch_list	channel list (string)	10000 - 15731	none

Comments

- Use SENS:FUNC:TEMP:POST to configure channels to measure thermocouples. Then use SENS:REF:POST to configure one or more channels to measure an isothermal reference temperature. Finally, use SENS:REF:CHAN:POST to group the reference channel with its thermocouple measurement channels.
- If thermocouple measurements are made through more than one isothermal reference panel, set up a reference channel for each. Execute the SENS:REF:CHAN:POST command for each reference/measurement channel group.
- **Related commands:** SENS:FUNC:TEMP:POST, SENS:REF:POST
- ***RST Condition:** No channels are linked to references.
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENS:FUNC:TEMP:POST  TC,J,.0625,(@10008:10015)   J type TCs on channels 8 through 15
SENS:REF:POST  THER,5000,4,(@10000),(@10001)     sense the reference temperature on
                                                 ch 10001; excitation on ch 10000;
                                                 measure on 4 V range
SENS:REF:CHAN (@10001),(@10008:10015)            link TC channels to thermistor
ROUT:SEQ:DEF (@10008:10015,10001,10000)          set up scan list
```

[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	°C

Comments

- This command is used to specify to the VT1422A the temperature of a controlled temperature thermocouple reference junction.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENS:FUNC:TEMP TC…
- ***RST Condition:** Reference temperature is 0°C.
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENSE:REF:TEMP  40
```

subsequent thermocouple conversion will assume compensation junction at 40 °C

[SENSe:]REFerence:TEMPerature:POST

[SENSe:]REFerence:TEMPerature:POST <*degrees_c*>,(@<*ch_list*>)
stores a fixed reference junction temperature for use in the post-processing EU conversions for the thermocouple channels in <*ch_list*>. Use when the thermocouple reference junction is kept at a controlled temperature. Note: Setting this value for any channel on a VT1529B also sets the value for all other channels on that VT1529B.

Unlike the SENS:REF:TEMP command for non-VT1529B channels, this command does not override the previous links to thermistor channels, so the two methods can be used on different channels on the same VT1529B.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
degrees_c	numeric (float32)	-126 to +126	°C

Comments

- This command is used to specify to the VT1422A the temperature of a controlled temperature thermocouple reference junction.
- An error will be generated if the channels in <*ch_list*> are not configured as thermocouples (SENS:FUNC:TEMP:POST command)
- **When Accepted:** Not while INITiated
- **Related Commands:** SENS:FUNC:TEMP:POST

- ***RST Condition:** Reference temperature is 0 °C.
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENSE:REF:TEMP:POST  40,(@10008:10015)
```

channels 10008 to 10015 will assume compensation junction at 40 °C

[SENSe:]REFerence:THERmistor:RESistance:POST

[SENSe:]REFerence:THERmistor:RESistance:POST *<resistance>*,(@*<thr_list>*) specifies the sum of the resistor values in the thermistor divider circuit (see Figure 5-5 on page 173) for the thermistor channels in *<thr_list>* when using the VT1529B to make a reference temperature measurement. **Note**: Setting this value for any channel on a VT1529B also sets the value for all other channels on that VT1529B.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
resistance	numeric (float32)	1800 to 520,000	ohms
thr_list	channel list (string)	10000 - 15731	none

Comments

- **When Accepted:** Not while INITiated
- **Related Commands:** SENS:REF:POST
- ***RST Condition:** SENS:REF:THER:RES:POST 400000
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENSE:REF:THER:RES:POST  397000,(@10001)
```

divider circuit for thermistor on channel 10001 has 397 kΩ total resistance

[SENSe:]REFerence:THERmistor:RESistance:POST?

[SENSe:]REFerence:THERmistor:RESistance:POST? (@<*thr_ch*>) returns the programmed sum of the resistor values in the thermistor divider circuit.

This command is valid only on the VT1529B and is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
resistance	numeric (float32)	1800 to 520,000	ohms
thr_ch	channel list (string)	10000 - 15731	none

Comments

- The <*thr_ch*> parameter must be a single channel only.
- **When Accepted:** Not while INITiated
- **Related Commands:** SENS:REF:THER:RES:POST
- ***RST Condition:** SENS:REF:THER:RES:POST 400000
- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENSE:REF:THER:RES:POST  397000,(@10001)
```

divider circuit for thermistor on channel 10001 has 397 kΩ total resistance

[SENSe:]STRain:BRIDge[:TYPE]

[SENSe:]STRain:BRIDge[:TYPE] <select>,(@<*ch_list*>) sets the VT1529A/B'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 description of the effects of <*select*> see "VT1529A/B Bridge Configurations" on page 70.
- **Related Commands:** SENSE:FUNC:…
- ***RST Condition:** SENS:STR:BRIDG[:TYPE] FBEN for all VT1529A/B channels.

- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SENS:STRAIN:BRID  Q120,(@10000:10031)
```

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 VT1529A/B'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

- The <*channel*> parameter 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 VT1529A/B channels
- **Send with VXI*plug&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 VT1529A/B channels specified by <*ch_list*> to sense either the strain bridge output or the bridge excitation supply. Only one channel for each VT1529A/B 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 VT1529A/B channels.

- **Send with VXI*plug&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 VT1529A/B channel specified by <*channel*>.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	10000 - 15731	none

Comments

- The <*channel*> parameter 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 VT1529A/B channels.
- **Send with VXI*plug&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 VT1529A/B, 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 VT1422A can sense through the VT1529A/B's excitation sense path is 16 volts (±8 V dc centered about the Gnd terminal). If a higher excitation voltage is supplied through the VT1529A/B, don't connect the excitation sense terminals.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
excite_v	numeric (float32)	0.01 - 99	volts
ch_list	channel list (string)	100 - 15731	none

Comments

- The <*ch_list*> parameter 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:** 3.9 V
- **Send with VXI*plug&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 VT1529A/B 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

- The <*channel*> parameter 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 VXI*plug&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 a VT1529A/B. Only one channel for each VT1529A/B needs to be specified in <*ch_list*> and all four excitation supply ports on that unit will configure as specified in <*enable*>. The first channel number on each possible VT1529A/B 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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

STRAIN:EXC:STAT ON,(@10800)

turn on all excitation supplies on VT1529A/B connected to on-board channel 08

[SENSe:]STRain:EXCitation:STATe?

[SENSe:]STRain:EXCitation:STATe? (@<*channel*>) returns the state of all four VT1529A/B excitation supply ports referenced in <*channel*>. The first channel number on each possible VT1529A/B 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
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 VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

Usage

STRAIN:EXC:STAT? (@10800)

check state of excitation supply ports on VT1529A/B connected to channel 08

[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

- The <*ch_list*> parameter 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 VXI*plug&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 VT1529A/B 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**.
- The <*channel*> parameter must specify a single channel only.
- **Related Commands:** STRAIN:GFACTOR
- **Send with VXI*plug&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)	0.1 - 0.5	none
ch_list	channel list (string)	100 - 163	none

Comments

- The <*ch_list*> parameter 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 VXI*plug&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**.
- The <*channel*> parameter must specify a single channel only.
- **Related Commands:** FUNC:STRAIN…, STRAIN:POISSON
- **Send with VXI*plug&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 VT1529A/B 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*>.
- The <*ch_list*> parameter 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
- **Power-on Condition:** Unstrained voltage is zero
- ***RST Condition:** unaffected by *RST
- **Send with VXI*plug&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**.
- The <*channel*> parameter must specify a single channel only.

- **Related Commands:** STRAIN:UNST
- **Send with VXI*plug&play* Function:** hpe1422_cmdReal64_Q(...)

Usage

```
STRAIN:UNST? (@107)
enter statement here
```

query unstrained voltage for channel 7 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 VXI*plug&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

- The <*channel*> parameter 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 VXI*plug&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>)
    :VOLTage
        [:AMPLitude]  <-offset_v>,(@<ch_list>)
```

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 VXI*plug&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 2 kHz */
```

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

- The <*channel*> parameter 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 VXI*plug&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 VT1533A SCP sources 8 digital bits on the channel specified by this command. The VT1534A SCP can source 1 digital bit on each of the channels specified by this command.
- **Send with VXI*plug&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 VXI*plug&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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

The frequency control for these channels is provided by the algorithm language function:

```
O143 = 2000 /* set channel 43 to 2 kHz */
```

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 VXI*plug&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

- The <*channel*> parameter must specify a single channel.
- **Returned Value:** returns 1 (on) or 0 (off). The type is **int16**.
- **Send with VXI*plug&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 0.238 µs)	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 ms */
```

- **Send with VXI*plug&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 VXI*plug&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 2 kHz */
```

- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SOUR:PULS:WIDTH  2.50E-3,(@143)
```

set fixed pulse width of 2.5 ms 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

- The <*channel*> parameter 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 VXI*plug&play* Function:** hpe1422_cmdReal64_Q(...)

SOURce:VOLTage[:AMPLitude]

SOURce:VOLTage[:AMPLitude] <-*offset_v*>,(@<*ch_list*>) can be used to reduce bridge offset voltage present at the dynamic strain "Buffered Output" channel connectors.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
-*offset_v*	numeric (float32)	-1.651 to +1.664 | MIN | MAX (resolution 13 mV)	volts
ch_list	string	10000 - 15731	none

Comments

- If the channels specified are not on a VT1529A/B, an error will be generated.
- ***RST Condition:** SOUR:VOLT 0,(@ <*all VT1529A/B channels*>)

- To reduce the offset voltage at each dynamic strain "Buffered Output" channel:

  1. Measure a Buffered Output channel with its bridge unstrained and place the value in a variable arbitrarily called <*offset_v*>.

  2. Send minus <*offset_v*> to that channel with the SOUR:VOLT command. For example: SOUR:VOLT <-*offset_v*>,(@10000)

- **Send with VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
SOUR:VOLT .041,(@10031)
```

correct a -41 mV offset at channel 31 of VT1529A/B 00.

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 the current real-time states of their status signal inputs (signal states are not latched) to be viewed. The Positive and Negative Transition Filter registers allow the polarity of change from the Condition registers that will set Event register bits to be controlled. 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 OR'ed together to form a summary bit in the Status Byte Summary register. Setting a bit to 1 in an Enable register enables the corresponding bit from its Event register.

Note For a complete discussion, See "Using the Status System" on page 142.

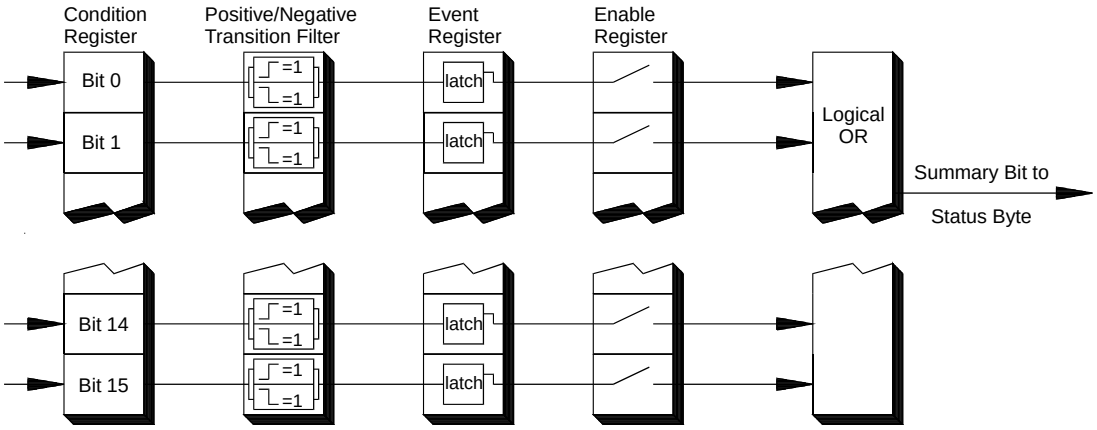


Figure 7-5. General Status Register Organization

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 VT1422A. 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 VT1422A 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; therefore, 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 VXI*plug&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

- The <*enable_mask*> parameter 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 and then is cleared, 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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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; therefore, 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 VXI*plug&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

- The <*enable_mask*> parameter 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 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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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 VXI*plug&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*>)
:CTYPe:REMote? <@<*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

- The <*channel*> parameter must specify a single channel only.
- **Returned Value:** An example of the response string format is:
  HEWLETT-PACKARD,E1422A 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 VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

Usage

```
SYST:CTYPE? (@100)        return SCP type installed at channel 0
```

SYSTem:CTYPe:REMote?

SYSTem:CTYPe:REMote? (@<*channel*>) returns the identification of the Remote Strain Conditioning Unit (VT1529A or VT1529B) installed at the specified channel.

This command is available with VXI*plug&play* driver revision A.01.09 or later (revision A.01.06 or later of hpe1422_32.dll).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	channel list (string)	100 - 15731	none

Comments

- The <*channel*> parameter must specify a single channel only. Either the 3-digit or 5-digit format can be used.
- **Returned Value:** Examples of the response string format are:
  E1539 Strain SCP; Chan 108 = E1529A; Chan 109 = E1529B
  E1539 Strain SCP; Chan 108 = No Device; Chan 109 = E1529B

  The data type is **string**.
- **Send with VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

Usage

```
SYST:CTYPE? (@100)
```
return SCP type installed 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 VXI*plug&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 VXI*plug&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 7-6 shows the overall Trigger System model. The shaded area shows the ARM subsystem portion.

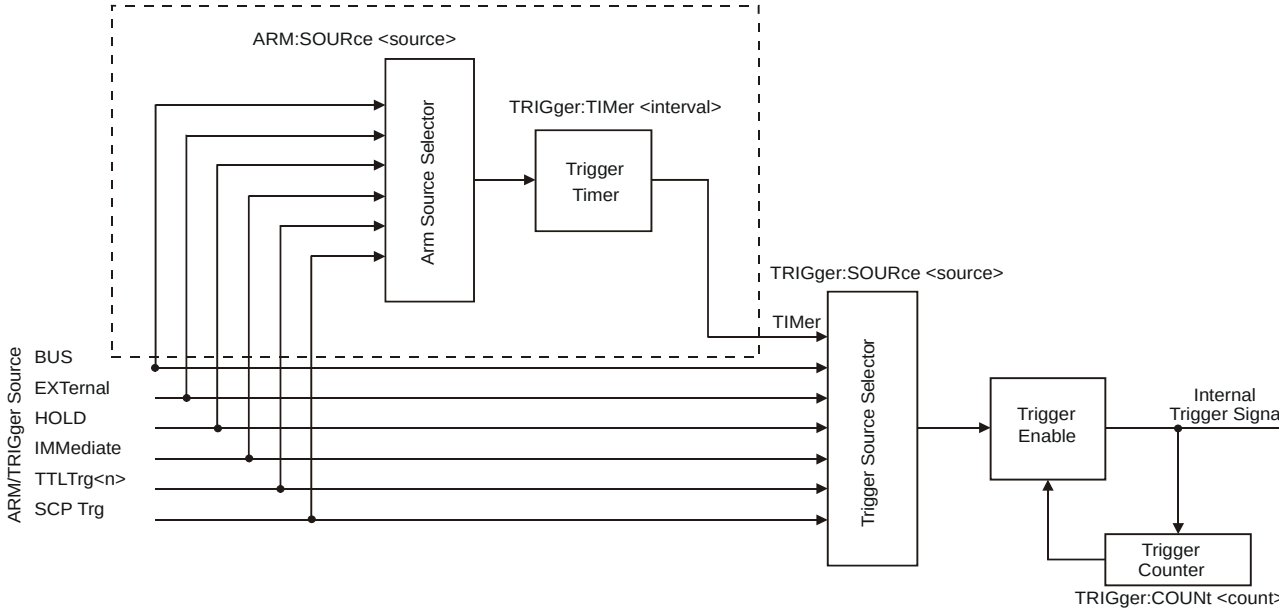


Figure 7-6. Logical Trigger Model

CAUTION **Algorithms execute at most once per trigger event. Should trigger events cease (i.e. external trigger source stops) or become 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 the process has been put into a safe state before halting (stop triggering) execution of a controlling algorithm.**

Event Sequence Figure 7-7 shows how the module responds to various trigger/arm configurations.

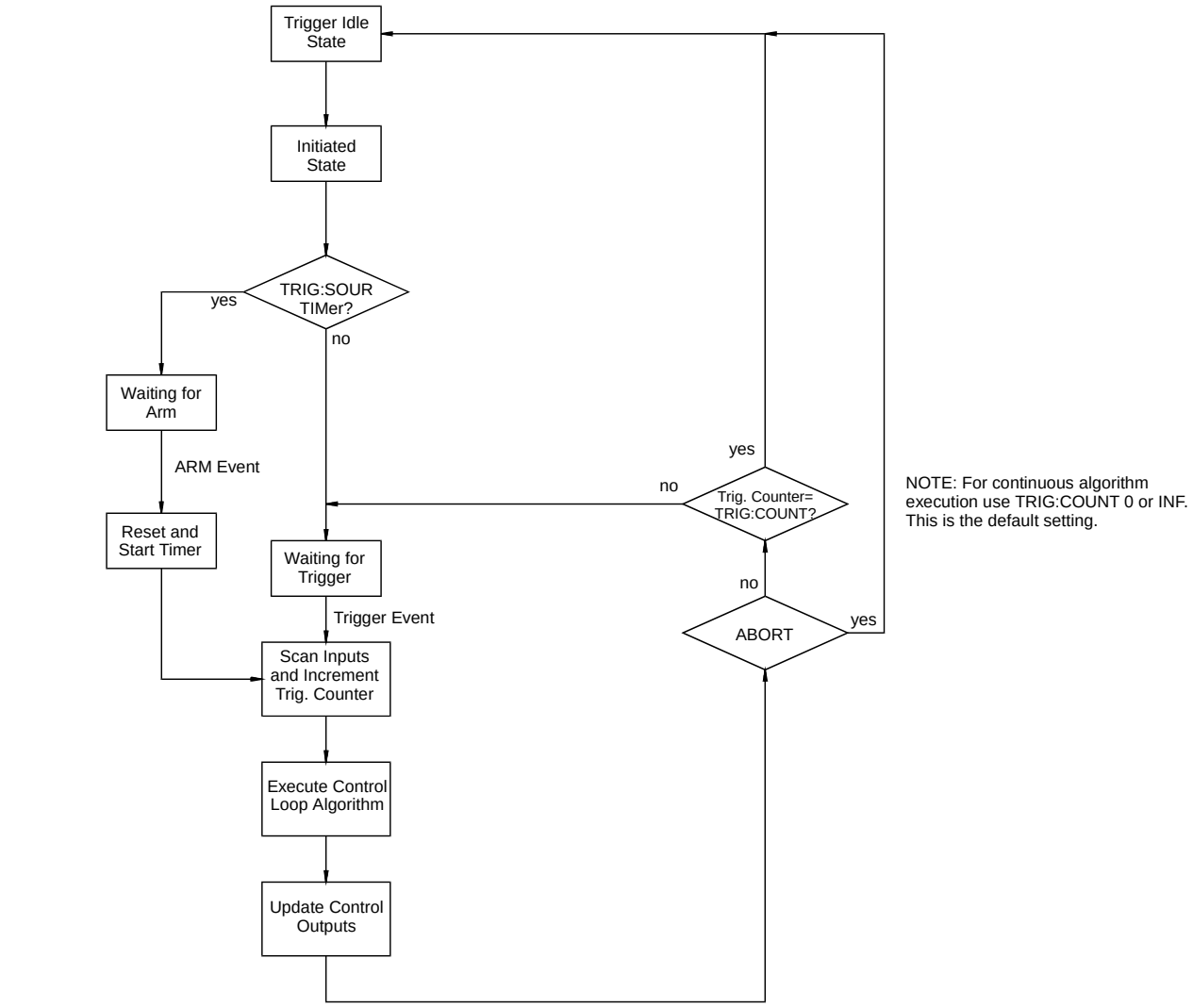


Figure 7-7. Trigger/Scan Sequence Diagram

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 acquisition scan lists easier (only one scan list worth of data in FIFO per trigger). For algorithm operation, it is usually desirable to change the count to INFinite to accept continuous triggers. See Figure 7-7 on page 394.

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 VXI*plug&play* Function:** hpe1422_cmd(...)

Usage

```
TRIG:COUNT 10          Set the module to make ten 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 VXI*plug&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 VXI*plug&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 GPIB)
EXTernal	“TRG” signal on terminal module
HOLD	TRIGger[:IMMediate]
IMMediate	The trigger event is always satisfied.
SCP	SCP Trigger Bus (future 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, simply 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 VXI*plug&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 VXI*plug&play* Function:** hpe1422_cmdString_Q(...)

Usage

TRIG:SOUR?	*ask VT1422A 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; 0.0016, 1.6 ms or 1600 us. The resolution for <*interval*> is 100 μs.
- 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 VXI*plug&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 VXI*plug&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 the VT1422A 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 seventeen 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 VT1422As, use CAL:SETup/CAL:SETup?.

Note that the scope of the *CAL? and CAL:SETup commands is limited to the VT1422A and the SCPs it contains. They do not calibrate Remote Signal Conditioning Units like the VT1529A/B. CAL:REMote? must be used 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 453.

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 VXI*plug&play* Function:** hpe1422_cmdInt16_Q(...)

Note If Open Transducer Detect (OTD) is enabled when *CAL? is executed, the module will disable OTD, wait one minute to allow channels to settle, perform the calibration and then re-enable OTD. If OTD is turned off by the program before executing *CAL?, it should also wait one 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, and 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 VXI*plug&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 the *EMC 0 command.

<*cmd_data*> is sent as *arbitrary block program data* (see "Arbitrary Block Program and Response Data" on page 233).

*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. An event is enabled 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 VXI*plug&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 233.

*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 the 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), synchronization between the instrument and an external computer or between multiple instruments can be ensured.

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 VXI*plug&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 the computer to read this response before continuing program execution, synchronization can be ensured 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 continuously, 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 VT1422A 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 VT1422A 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 VT1531A and VT1532A Analog Output SCP channels to zero

- **When Accepted:** Not while INITiated

- **Use VXI*plug&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 1 kΩ to +3 V, a TTL one) and analog control outputs change to zero (current or voltage). Keep these changes in mind when applying the VT1422A to the system or engineering a system for operation with the VT1422A. Also, note that each analog output channels disconnects for 5 - 6 milliseconds to discharge to zero at each *RST.**

- 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 one to identify which of these events will assert an GPIB 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 GPIB SRQ line true). An event is enabled 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" 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 VXI*plug&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 VXI*plug&play* function:** hpe1422_readStatusByte_Q(...)

Send with VXI*plug&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 VXI*plug&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 five 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 VXI*plug&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 = 0.0625, 201 = 0.25V, 202 = 1 V, 203 = 4 V, and 204 = 16 V 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 the one causing the problem is found.

  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. To read the FIFO for *TST? diagnostic information in a format other than 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 403 for a list of the module's default states.
- *TST? performs the following tests on the VT1422A 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
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 -15 V 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 SCP offset
304: Current Output SCP gain shift
305: Current Output SCP offset
306: Current Output SCP linearity
307: Current Output SCP linearity
308: Current Output SCP turn over
313: Voltage Output SCP offset
315: Voltage Output SCP offset
316: Voltage Output SCP linearity
317: Voltage Output SCP linearity
318: Voltage Output SCP turn over
331: Digital I/O SCP internal digital interface
332: Digital I/O SCP user input
333: Digital I/O SCP user input
334: Digital I/O SCP user output
335: Digital I/O SCP user output
336: Digital I/O SCP output current
337: Digital I/O SCP output current
341: Freq/PWM/FM SCP internal data0 register
342: Freq/PWM/FM SCP internal data1 register
343: Freq/PWM/FM SCP internal parameter register
344: Freq/PWM/FM SCP on-board processor self-test
345: Freq/PWM/FM SCP on-board processor self-test
346: Freq/PWM/FM SCP user inputs
347: Freq/PWM/FM SCP user outputs
348: Freq/PWM/FM SCP outputs ACTive/PASSive
349: Freq/PWM/FM SCP output interrupts
350: Watchdog SCP enable/disable timer
351: Watchdog SCP relay drive and coil closed
352: Watchdog SCP relay drive and coil open
353: Watchdog SCP I/O Disconnect line
354: Watchdog SCP I/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 VXI*plug&play* Function: hpe1422_cmd(...)

Command Quick Reference

The following tables summarize SCPI and IEEE-488.2 Common (*) commands for the VT1422A 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
CALCulate	
:TEMPerature:THERmistor? <*thr_volts*>,<*exc_volts*>[,*resistance*]	Calculates temperature (in °C) of VT1586A reference thermistor.
:TEMPerature:TCouple? <*type*>,<*thr_volts*>,<*volt_array*>	Converts array of thermocouple output voltages into temperatures (in °C)
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

SCPI Command Quick Reference	
Command	**Description**
CALibration (continued)	
:REMote?	Calibrates Remote Signal Conditioning Units
:DATA	Sends RSC cal constants from CAL:REM:DATA? back to VT1422A
:DATA?	Queries VT1422A 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)
: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 VT1529A/B 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,E1422A,US34000478,A.04.00, Wed Jul 08 11:06:22 MDT 1994
FETCh?	Return readings stored in VME Memory (format set by FORM cmd)

SCPI Command Quick Reference

Command	Description
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 formatted for 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
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 setup 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 VT1529A/B'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.

SCPI Command Quick Reference	
Command	**Description**
OUTPut (continued)	
: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
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
:HVOLtage [<*range*>,](@<*ch_list*>)	Links VT1529B high-level input (DCV measurement) channels to custom EU conversion table loaded by DIAG:CUST:MXB or DIAG:CUST:PIEC
: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
:HVOLtage (@<*ch_list*>)	Links the specified VT1529B channels to return DCV on high-input pins on RJ-45 connector for channel

SCPI Command Quick Reference

Command	Description
SENse:FUNCtion (continued)	
: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
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:FBPoisson [*<range>*,](@*<ch_list>*)	Full Bending Poisson
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:FPOisson [*<range>*,](@*<ch_list>*)	Full POisson
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:HBENding [*<range>*,](@*<ch_list>*)	Half BENding
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:HPOisson [*<range>*,](@*<ch_list>*)	Half Poisson
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
[:QUARter] [*<range>*,](@*<ch_list>*)	QUARter
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:Q120 [*<range>*,](@*<ch_list>*)	Quarter bridge with the 120 Ω resistor selected (VT1529A/B only)
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:Q350 [*<range>*,](@*<ch_list>*)	Quarter bridge with the 350 Ω resistor selected (VT1529A/B only)
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
:USER [*<range>*,](@*<ch_list>*)	Quarter bridge with the user supplied resistor selected (VT1529A/B only)
:POST [*<range>*,](@*<exc_ch>*),(@*<ch_list>*)	Links post-processing strain EU conversion with specified VT1529B channels
RTD, 85 | 92 TCouple, CUST | E | EEXT | J | K | N | S | T THERmistor, 2250 | 5000 | 10000	RTDs thermocouples thermistors
:TEMPerature *<sensor_type>*,*<sub_type>*, [*<range>*,](@*<ch_list>*)	Configure channels for temperature measurement types above: excitation current comes from Current Output SCP.
:POST TC, *<sub_type>*,[*<range>*,](@*<ch_list>*)	Links VT1529B channels to post-processing temperature EU conversion
:TOTalize (@*<ch_list>*)	Configure channels to count digital state transitions
:VOLTage[:DC] [*<range>*,](@*<ch_list>*)	Configure channels for dc voltage measurement
RTD, 85 | 92 THERmistor,5000	RTDs thermistors
REFerence *<sensor_type>*,*<sub_type>*,[*<range>*,](@*<ch_list>*)	Configure channel for reference temperature measurements above:
:POST THERmistor, *<resistance>*,[*<range>*], (@*<exc_ch>*),(@*<thr_ch>*)	Links thermistor channel to excitation voltage channel and to reference junction temperature EU conversion
:CHANnels (@*<ref_channel>*),(@*<ch_list>*)	Groups reference temperature channel with TC measurement channels
:POST (@*ref_channel*>), (@*<ch_list>*)	Links thermocouple channels to temperature reference (thermistor) channel
:TEMPerature *<degrees_c>*	Specifies the temperature of a controlled temperature reference junction
:POST *<degrees_c>*, (@*<ch_list>*)	Stores fixed reference junction temperature for post-processing EU conversions for specified thermocouple channels
:THERmistor	
:RESistance:POST *<resistance>*, (@*<thr_list>*)	Specifies sum of resistor values in thermistor divider circuit for specified thermistor channels when using VT1529B for temperature measurements
:RESistance:POST? (@*<thr_ch>*)	Returns programmed sum of resistor values in thermistor divider circuit

SCPI Command Quick Reference	
Command	**Description**
:STRain	
:BRIDge[:TYPE] <*select*>,(@<*ch_list*>)	Set bridge configuration switches on VT1529A/B
:BRIDge[:TYPE]? (@<*channel*>)	Returns the current bridge configuration setting
:CONNect BRIDge | EXCitation,(@<*ch_list*>)	Set switches to sense bridge output or excitation voltage on VT1529A/B
: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 VT1529A/B 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
:PULSe	
: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
:VOLTage	
[:AMPLitude] <-*offset_v*>,(@<*ch_list*>)	Used to correct for bridge offset at dynamic strain connector (Buffered Output)
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
:NTRansition <*transition_mask*>	Sets mask bits to enable pos. Condition Reg. transitions to Event reg
:NTRansition?	Returns positive transition mask value
:PTRansition <*transition_mask*>	Sets mask bits to enable neg. Condition Reg. transitions to Event reg
:PTRansition?	Returns negative transition mask value
:PRESet	Presets both the Operation and Questionable Enable registers to 0

SCPI Command Quick Reference	
Command	**Description**
STATus (continued)	
: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. Condition Reg. transitions to Event reg
:NTRansition?	Returns positive transition mask value
:PTRansition *<transition_mask>*	Sets mask bits to enable neg. Condition 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,E1422A,<*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	Triggers module when TRIG:SOUR is HOLD.
	*WAI	Wait to Complete	

Notes:

Notes:

Appendix A
Specifications

VT1422A Specifications

Power Requirements (with no SCPs installed)

	+5 V		+12 V		-12 V		+24 V		-24 V		-5.2 V	
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	Δ Pressure (mm H_2O)	Air Flow (liters/s)
14	0.08	0.08

Power Available for SCPs
(See VXI Catalog or SCP manuals for SCP current)

1.0 A ±24 V, 3.5 A 5 V

VT1539A Power Requirements
(in Amps)

5 V: 0.112A typ, 0.168 A max

Measurement Ranges

dc volts	(VT1501A or VT1502A) ±62.5 mV to ±16 V 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	(VT1505A with VT1501A) 512 Ω to 131 kΩ FS
Strain	25,000 $\mu\in$ 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	100 ppm (0.01%) 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 100 ns. Since each trigger starts a complete scan of two 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 VT1513A Divide by 16 Attenuator SCP: Operating: <±60 V dc, <±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 VT1513A Divide by 16 Attenuator SCP: Operating: <±60 V dc, <±42 V_{PEAK}
Common Mode Rejection	0 to 60 Hz -105 dB
Input Impedance	greater than 90 MΩ differential (1 MΩ with VT1513A 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	0.82101	0.23061	0.07581	0.03792

The following specifications reflect the performance of the VT1422A with the VT1501A Direct Input Signal Conditioning Plug-on. The performance of the VT1422A 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 < 3.8 V, add ±0.02% to linearity specifications.

for readings ≥ 3.8 V, add ±0.05% to linearity specifications.

A/D range ± V F. Scale	Linearity % of Reading	Offset Error	Noise 3 sigma	Noise* 3 sigma

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 < 3.8 V, add ±0.02% to linearity specifications.

for readings ≥ 3.8 V, add ±0.05% to linearity specifications.

0.0625	0.01%	5.3 μV	18 μV	8 μV
0.25	0.01%	10.3 μV	45 μV	24 μV
1	0.01%	31 μV	110 μV	90 μV
4	0.01%	122 μV	450 μV	366 μV
16	0.01%	488 μV	1.8 mV	1.5 mV

Temperature Coefficient: Gain - 10 ppm/°C. Offset - (0 - 40 °C) 0.14 μV/°C, (40 - 55 °C) 0.8 μV + 0.38 μV/°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 transducer accuracy to determine total measurement error.

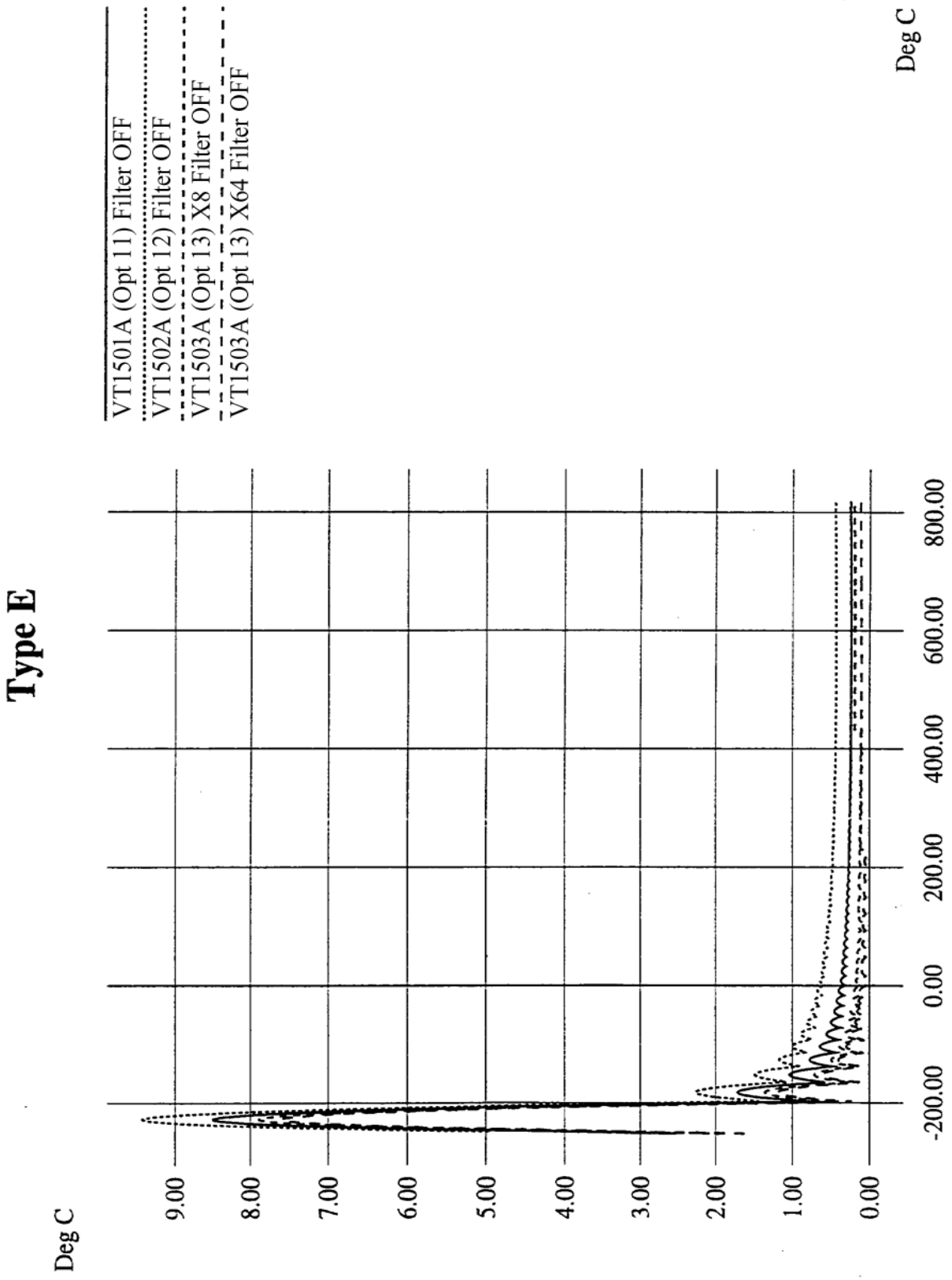
The thermocouple graphs on the following pages include only the errors due to measuring the voltage output of the thermocouple, as well as the algorithm errors due to converting the thermocouple voltage to temperature. To this error must be added the error due to measuring the reference junction temperature 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 VXI Technology termination blocks are used as the isothermal reference, see the notes below.

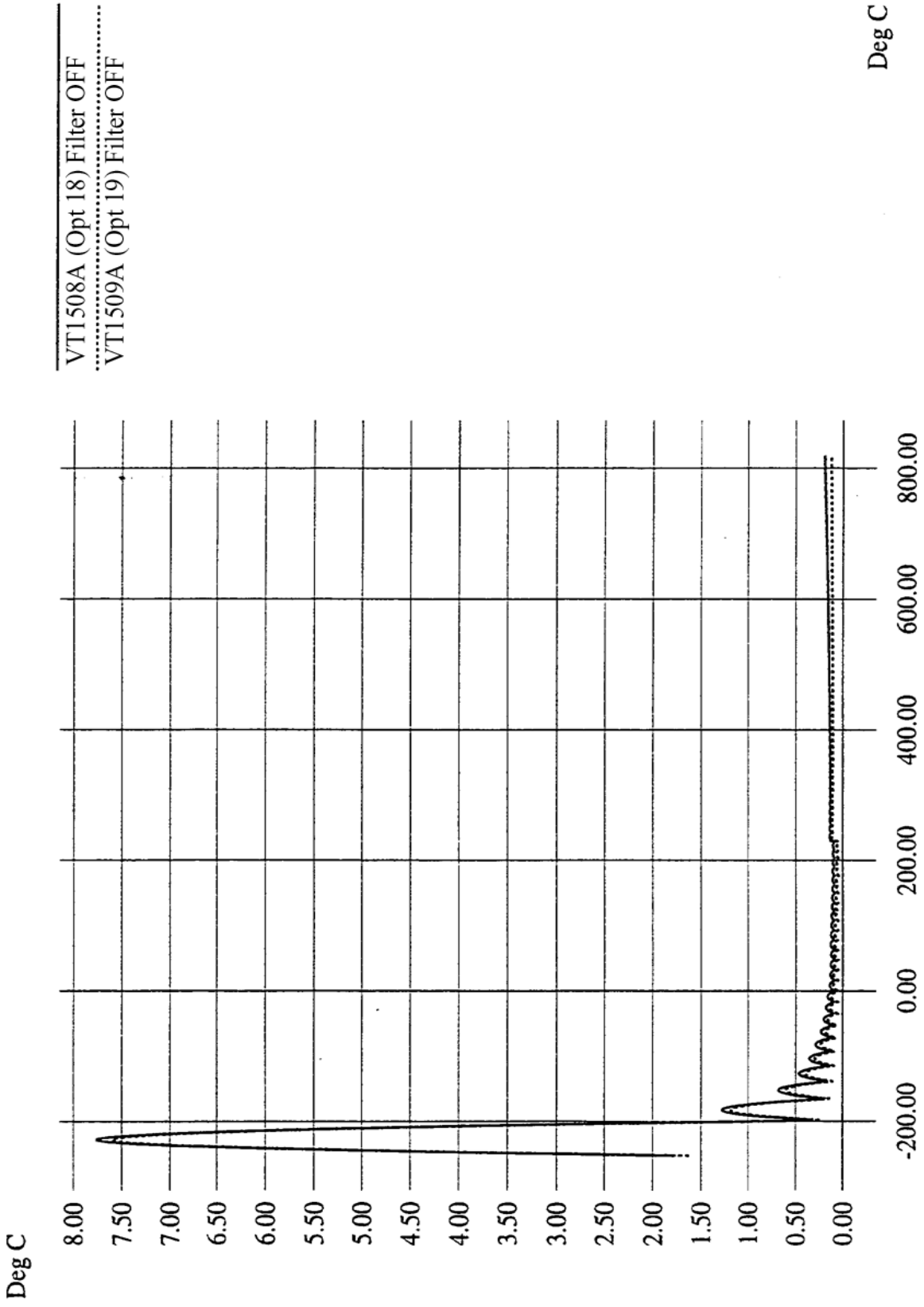
NOTES

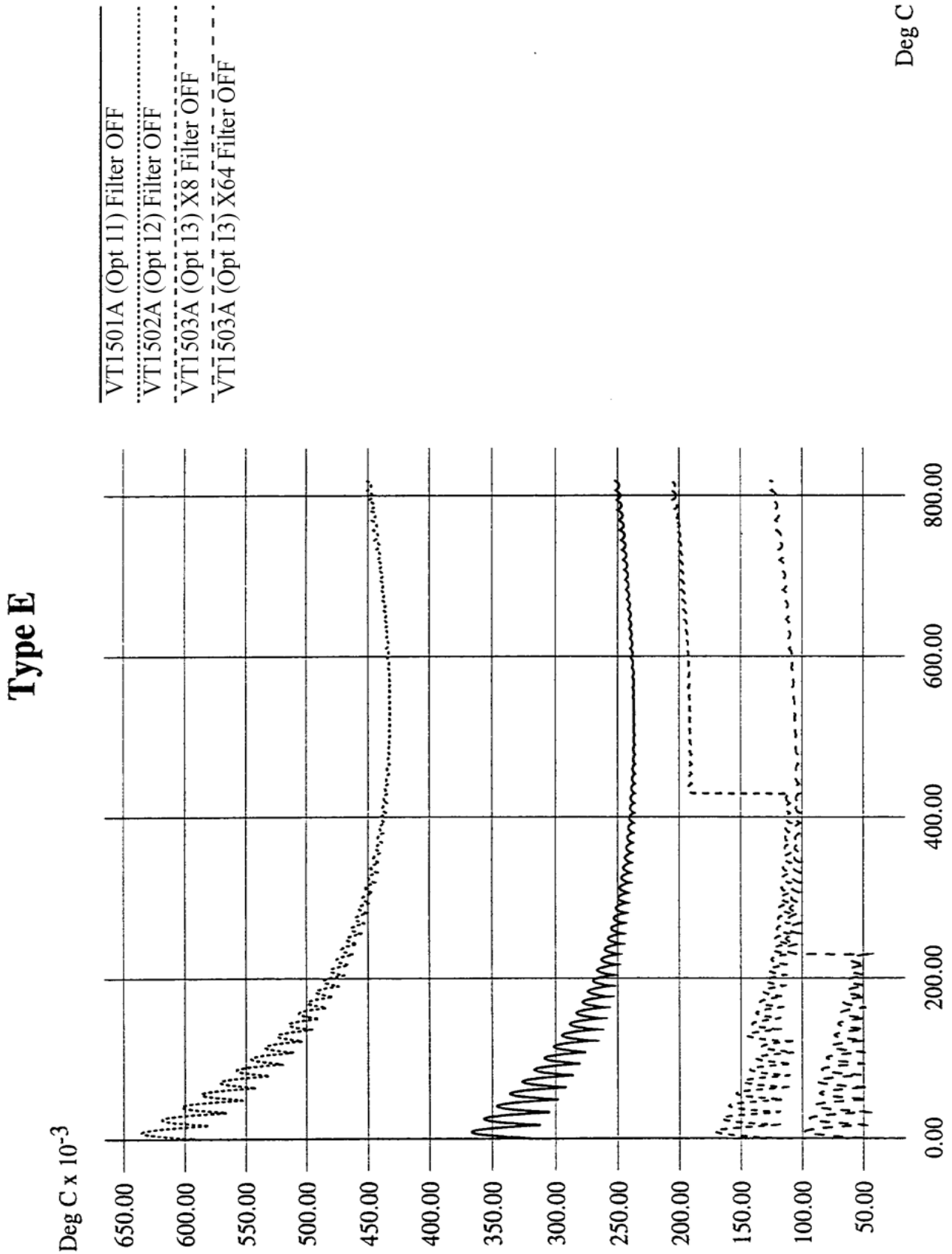
1) When using the Terminal Module as the isothermal reference, add ±0.6 °C to the thermocouple accuracy specs to account for temperature gradients across the Terminal Module. The ambient temperature of the air surrounding the Terminal Module must be within ±2 °C of the temperature of the inlet cooling air to the VXI mainframe.

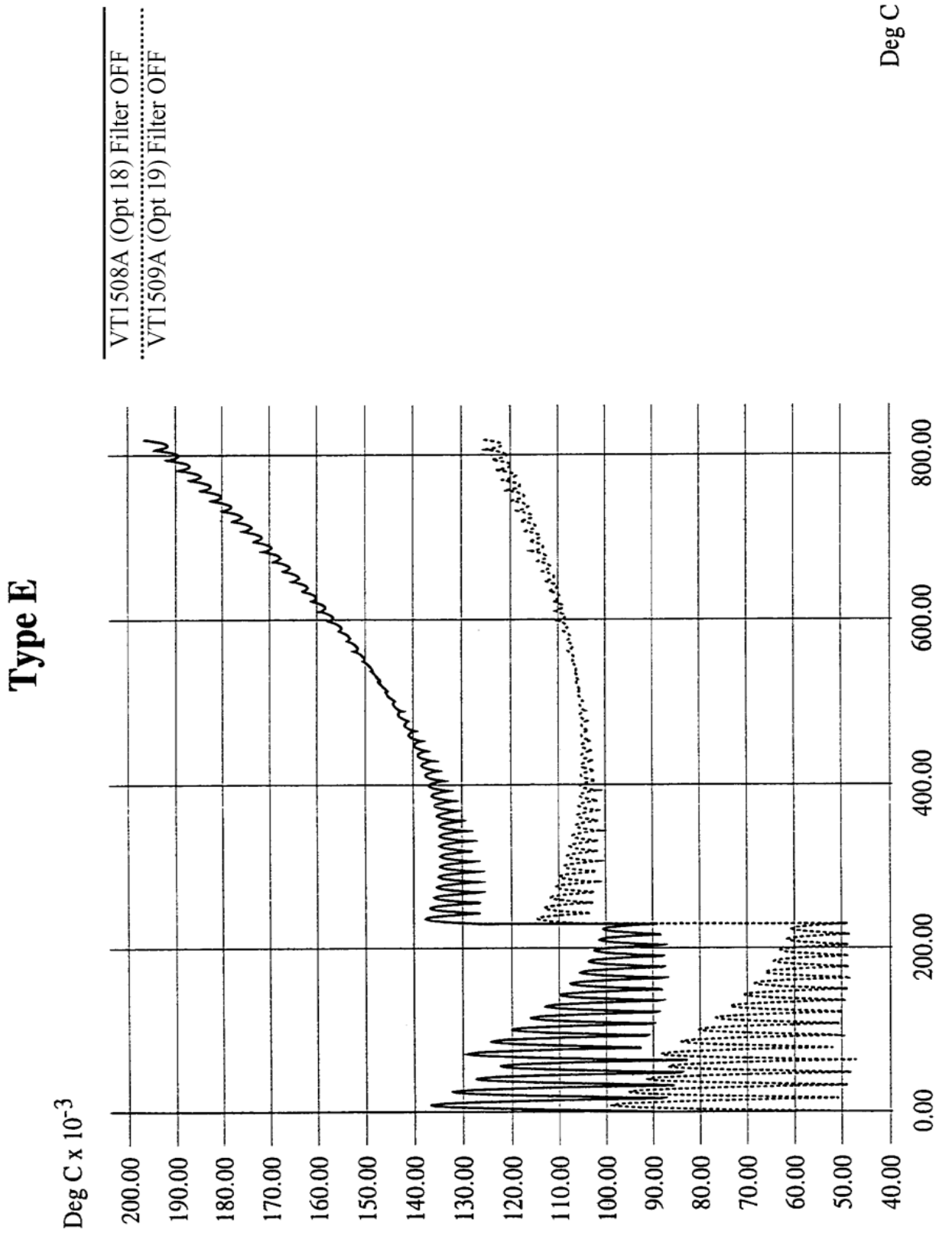
2) When using the VT1586A Rack-Mount Terminal Panel as the isothermal reference, add ±0.2 °C to the thermocouple accuracy specs to account for temperature gradients across the VT1586A. The VT1586A should be mounted in the bottom part of the rack, below and away from other heat sources for best performance.

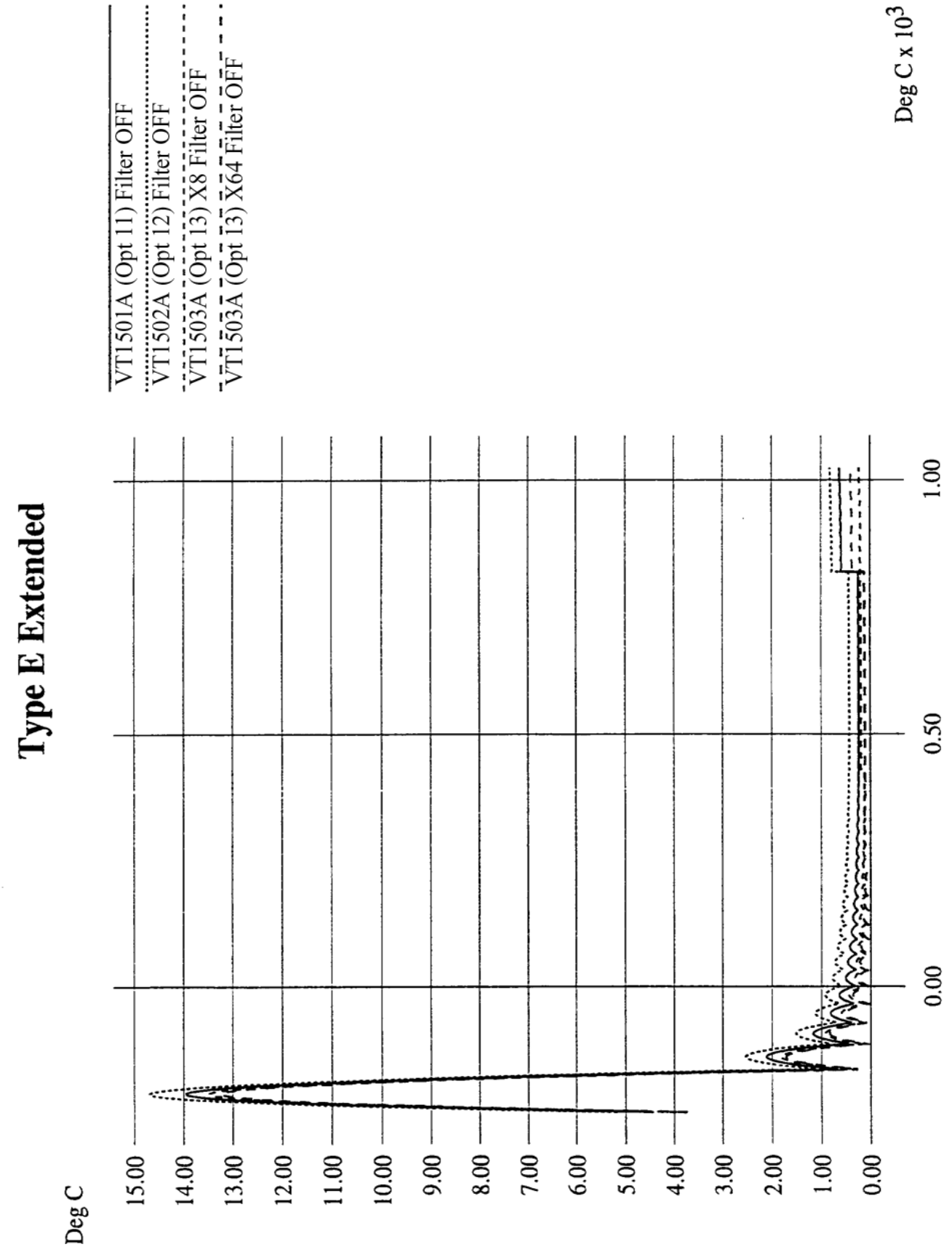
The temperature specification graphs are found on the following pages:

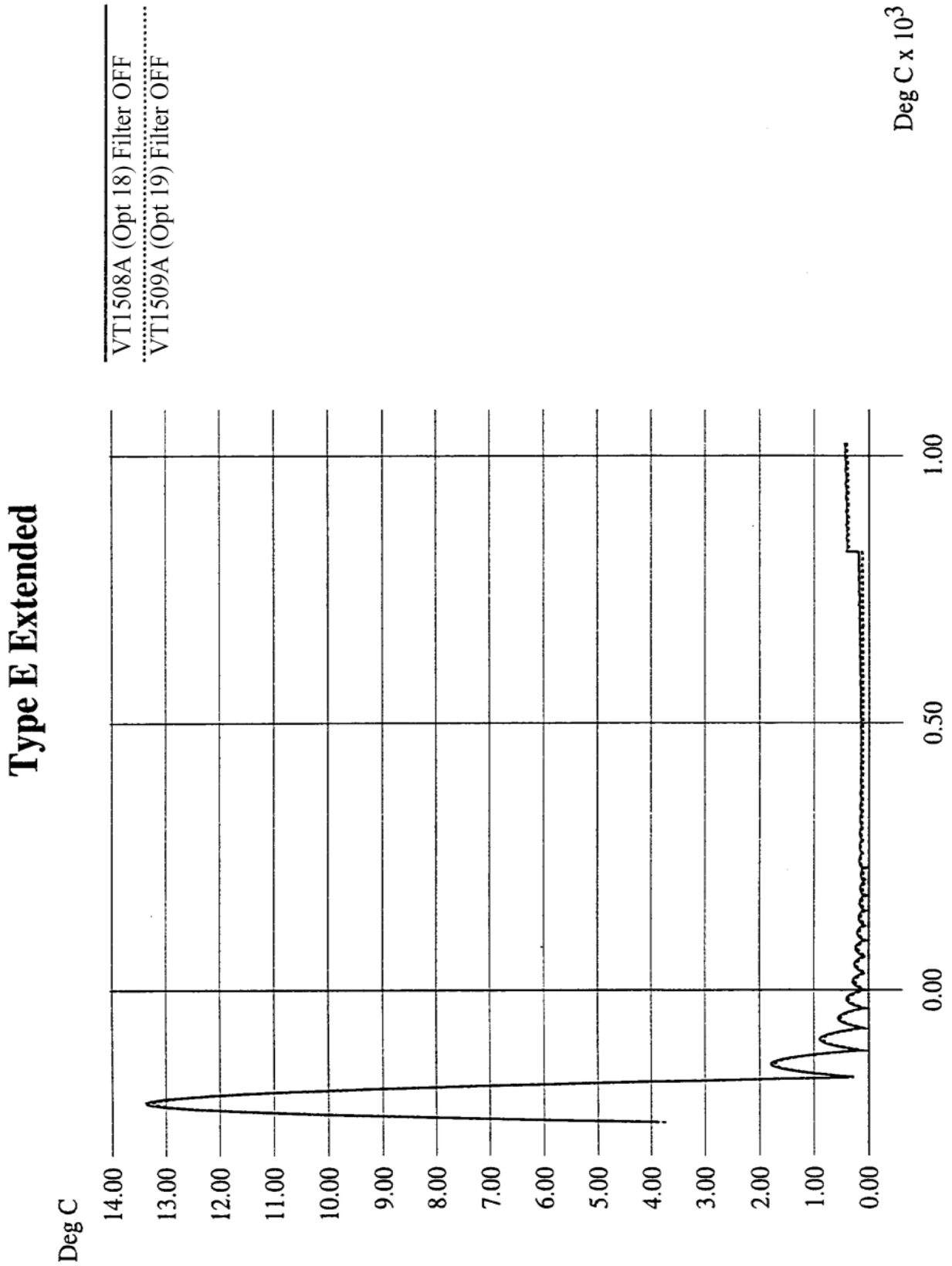


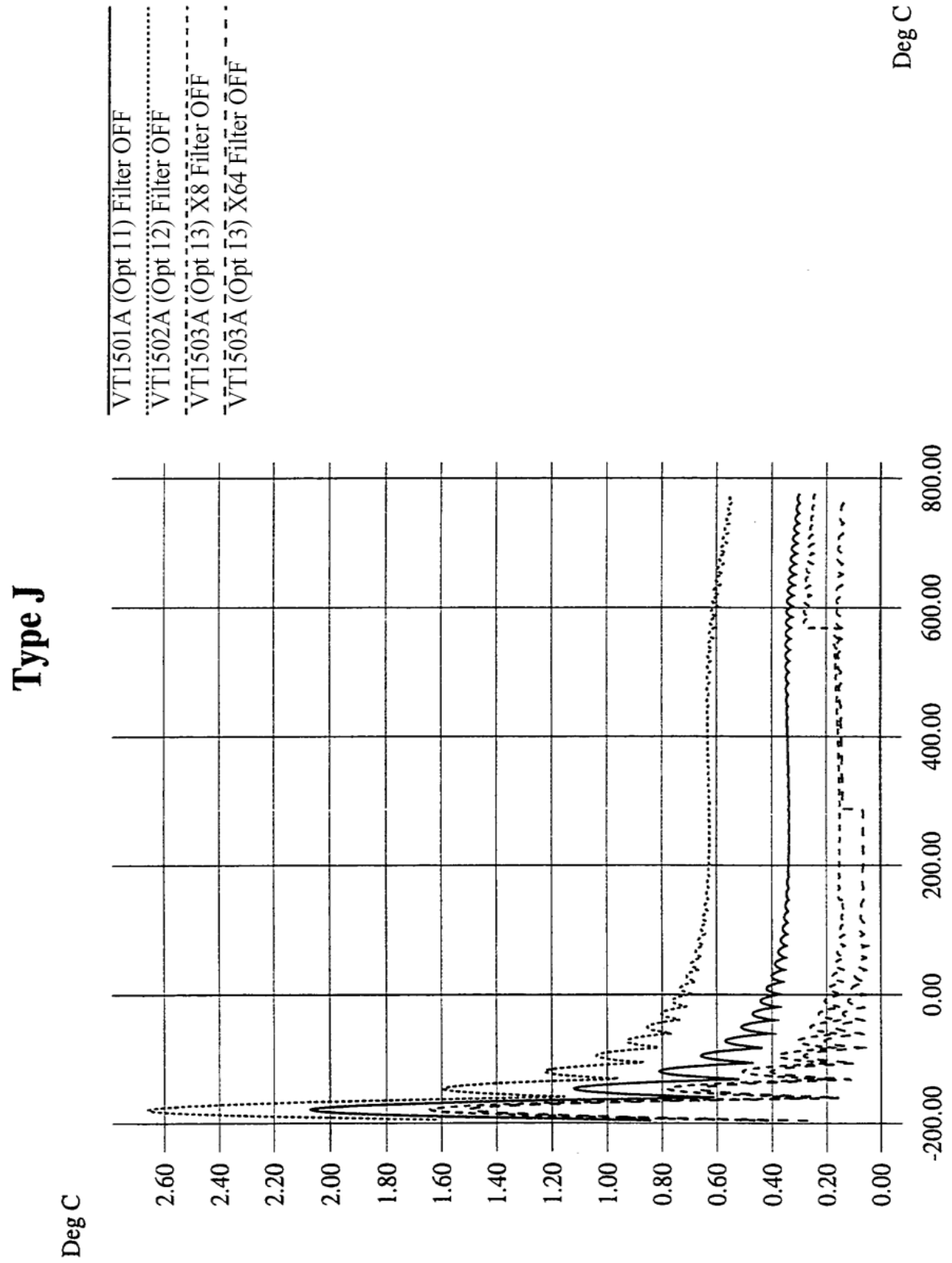


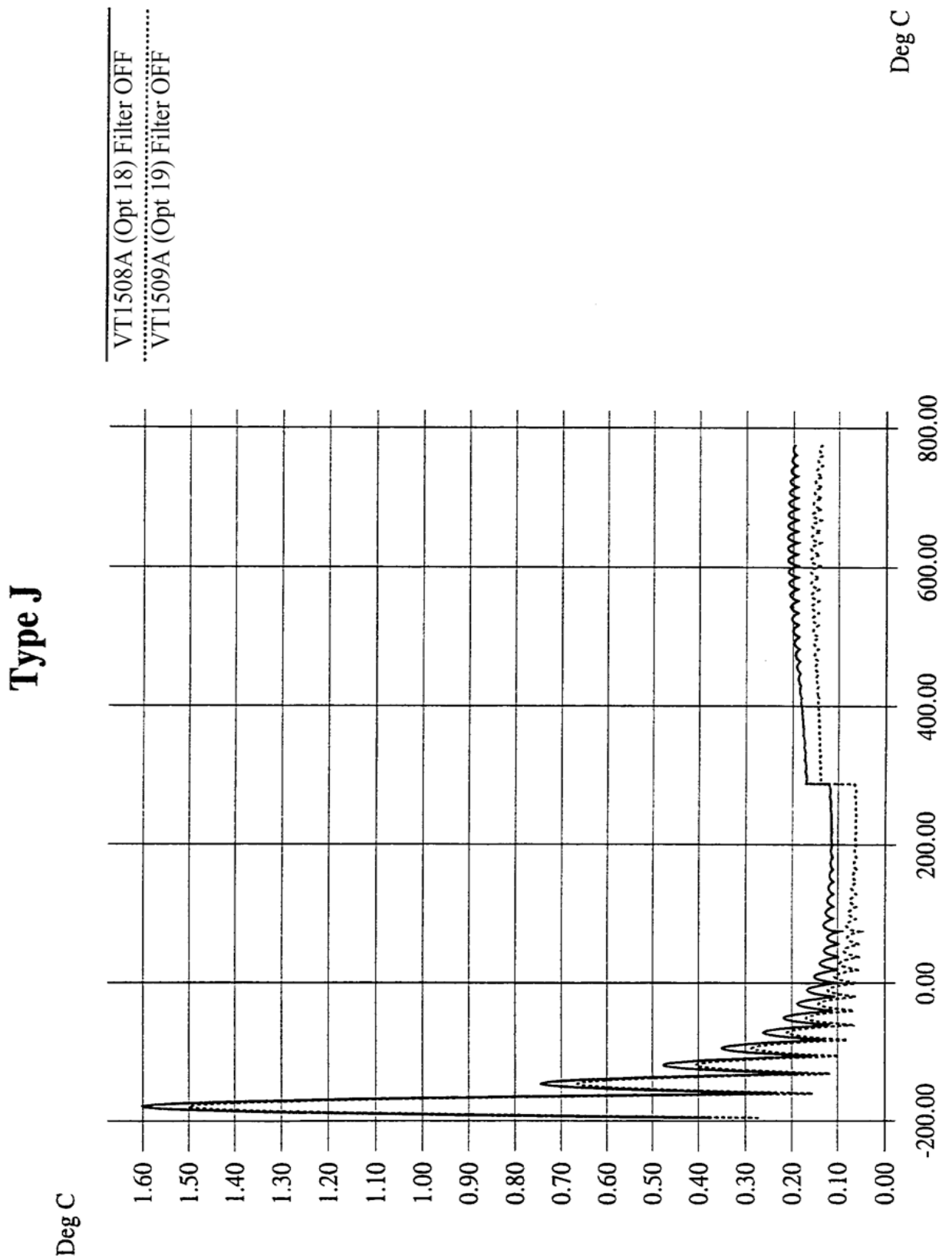


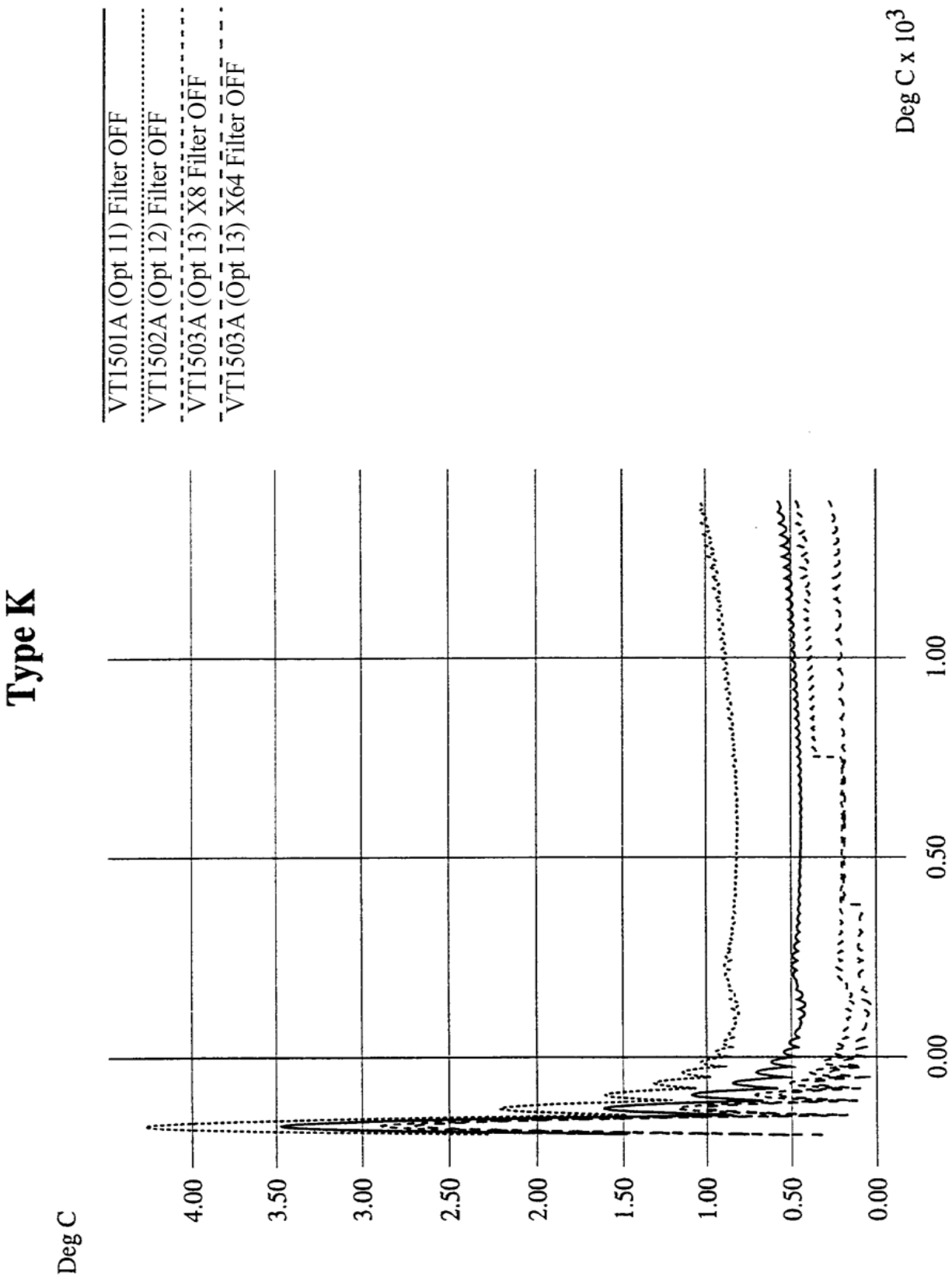


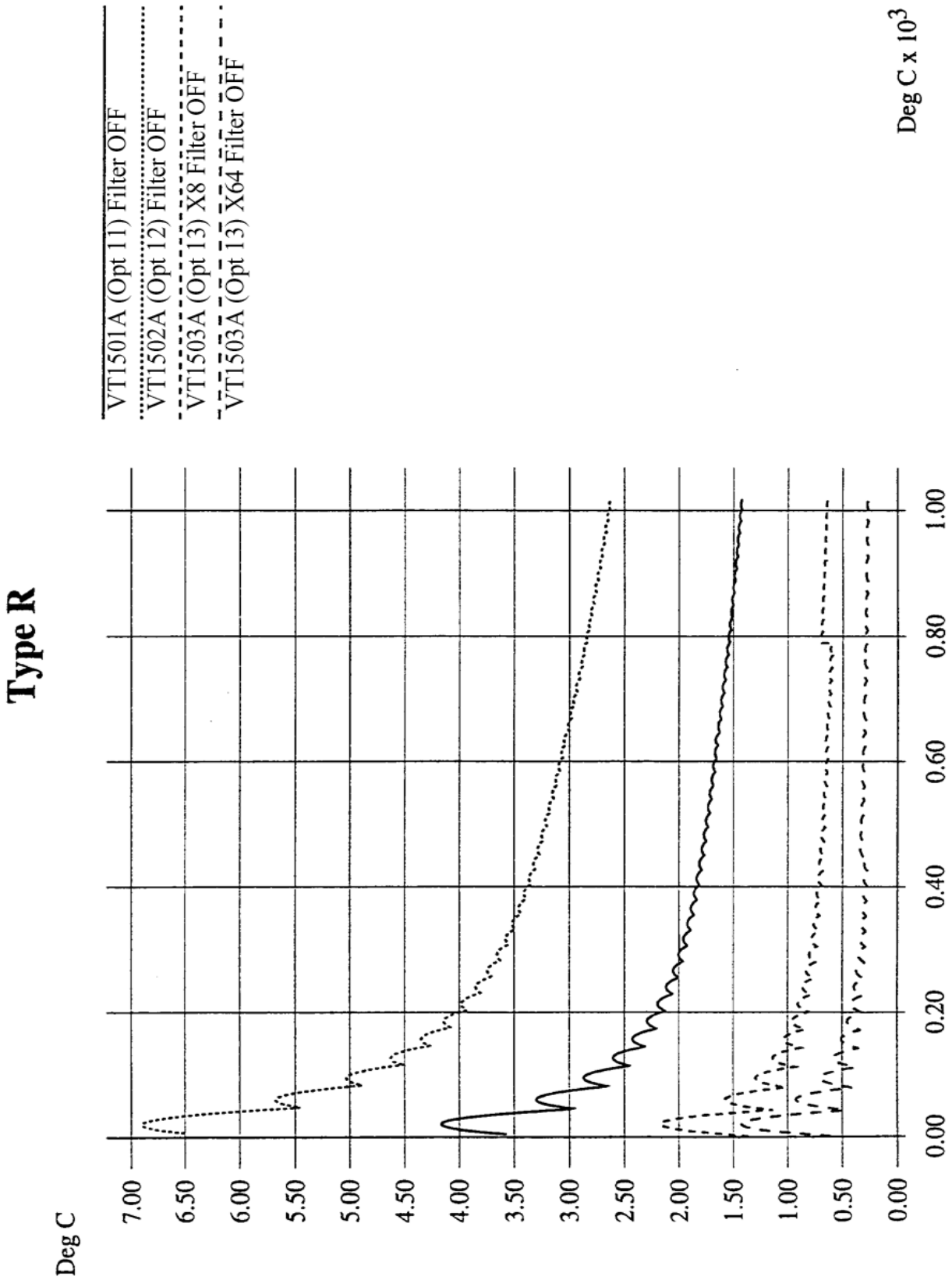


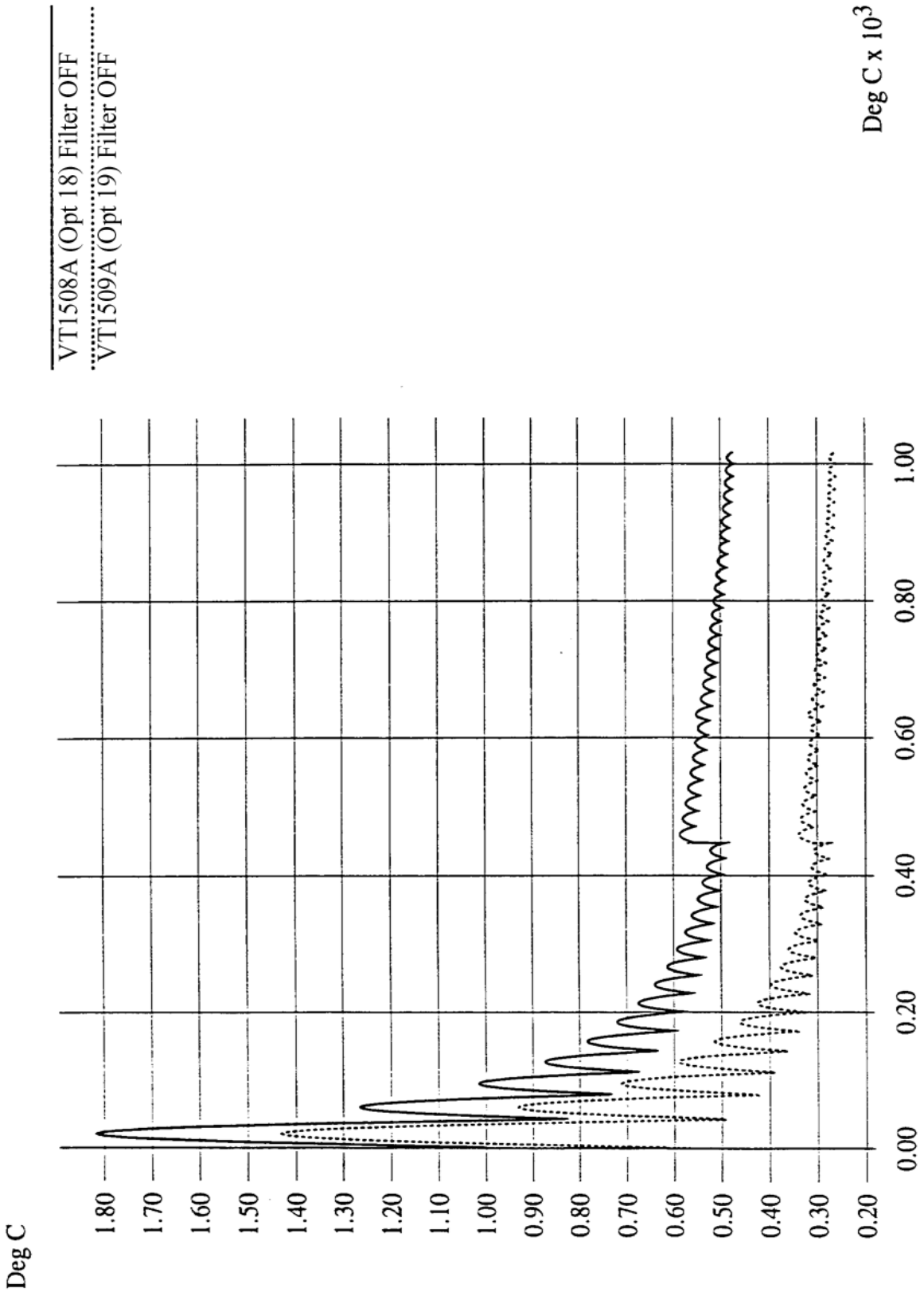


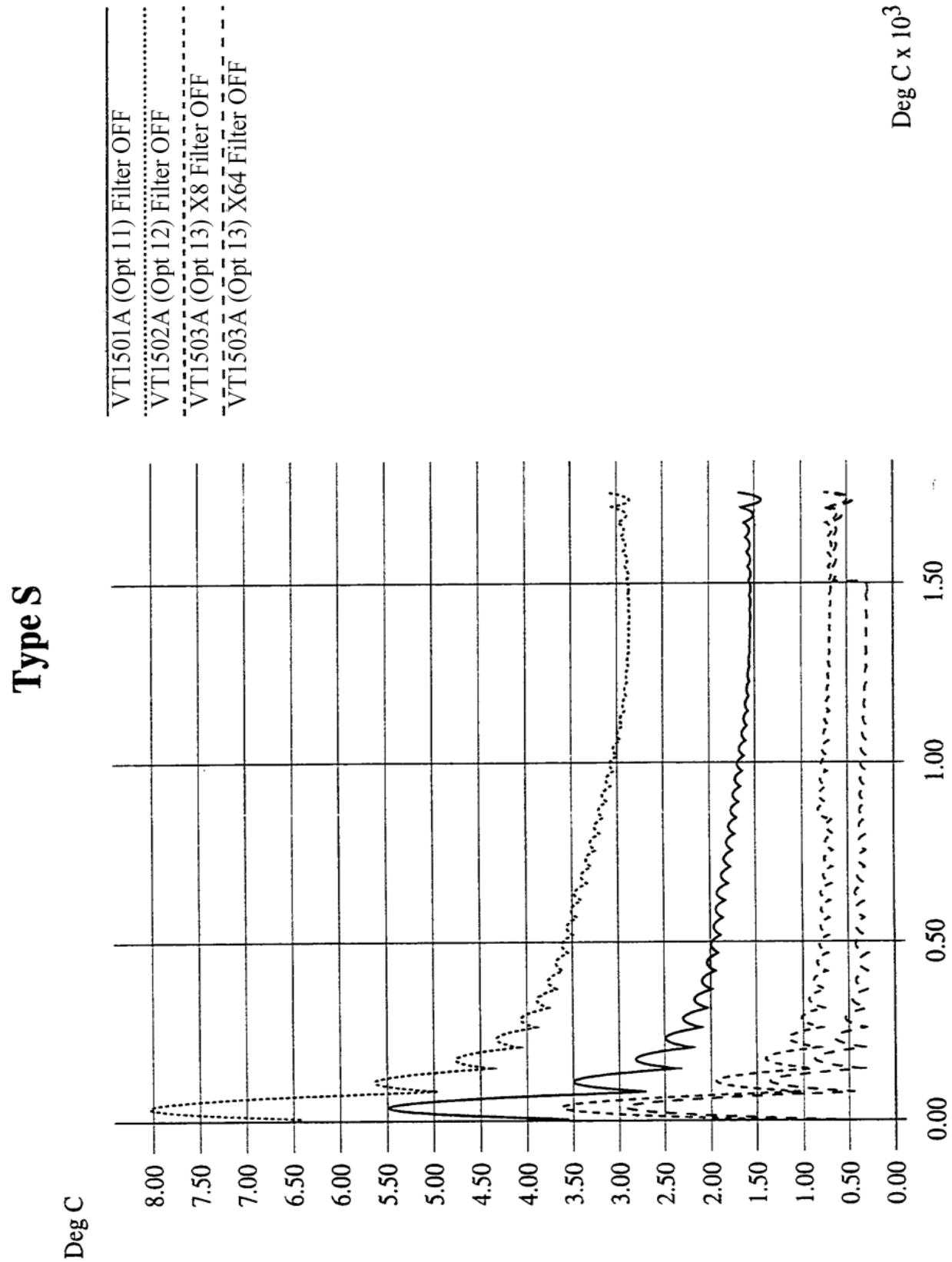


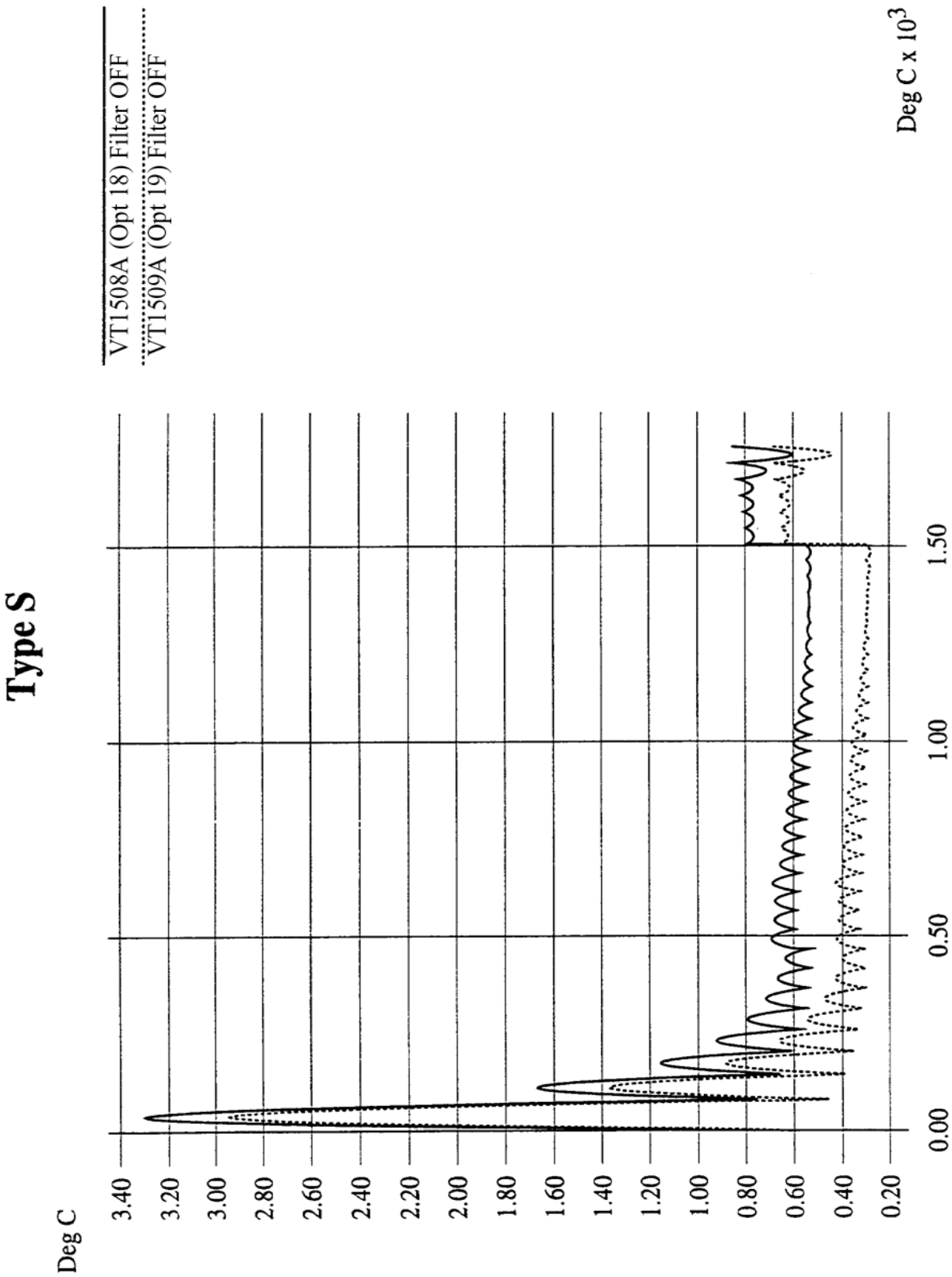


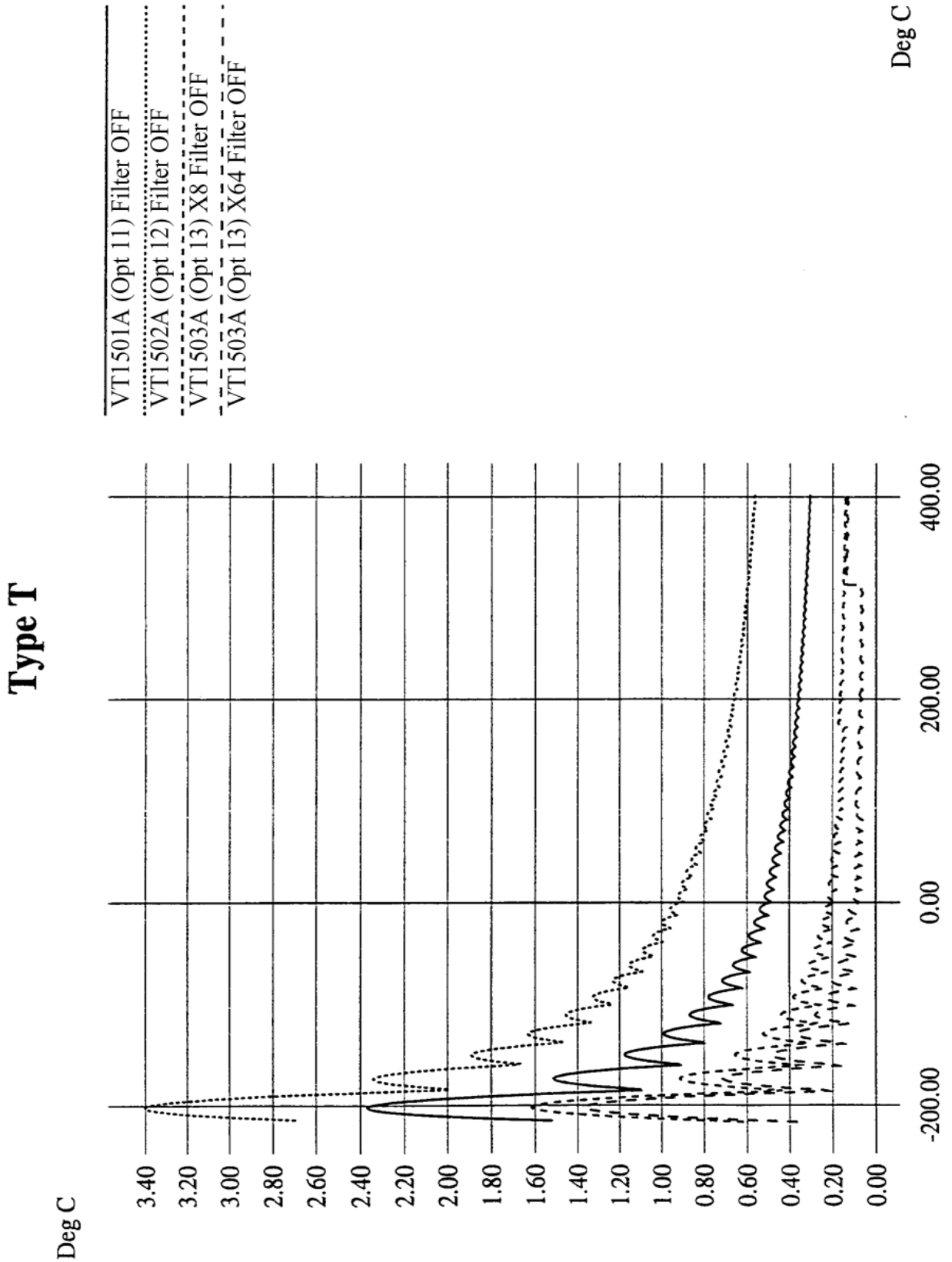


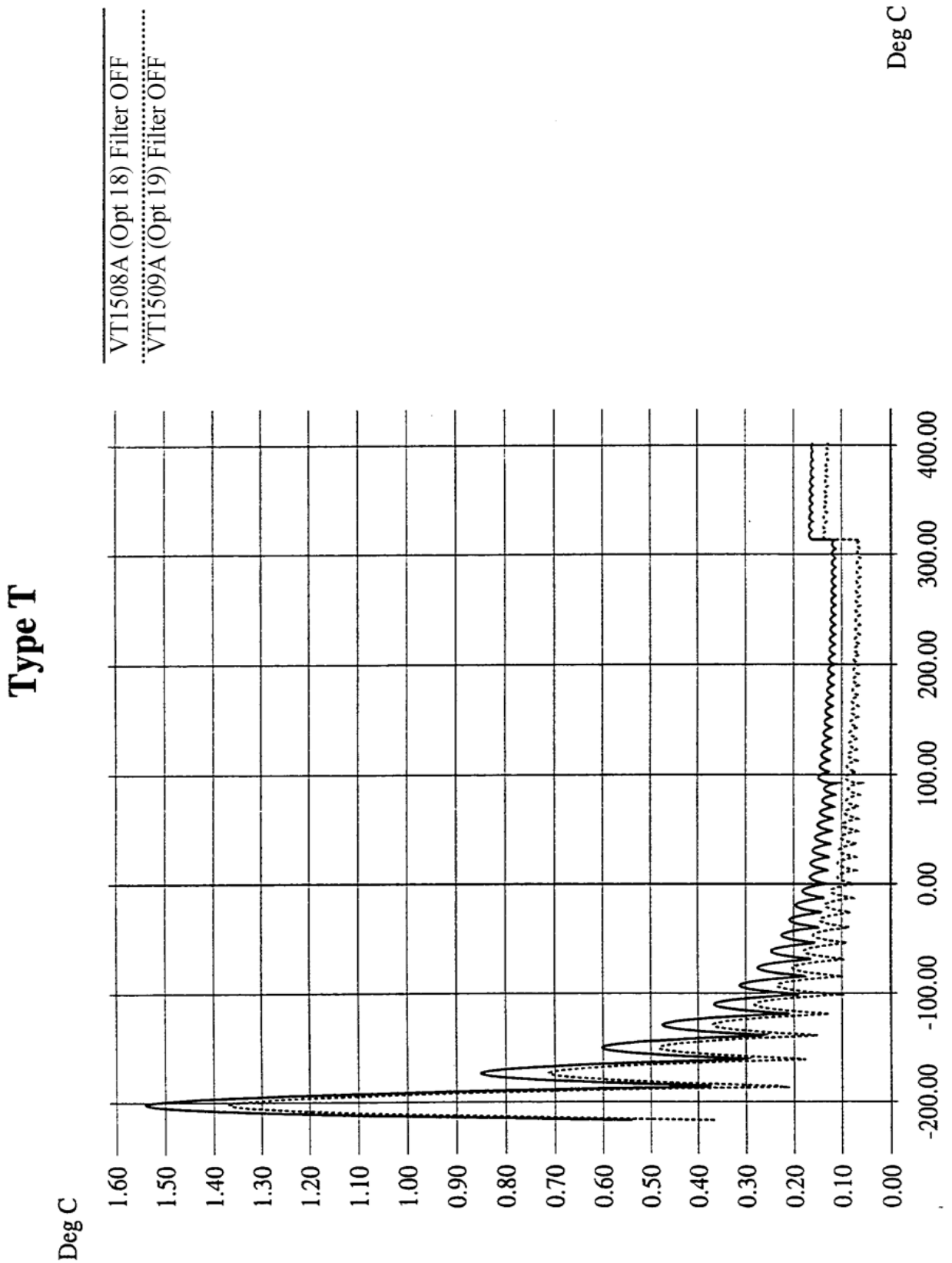


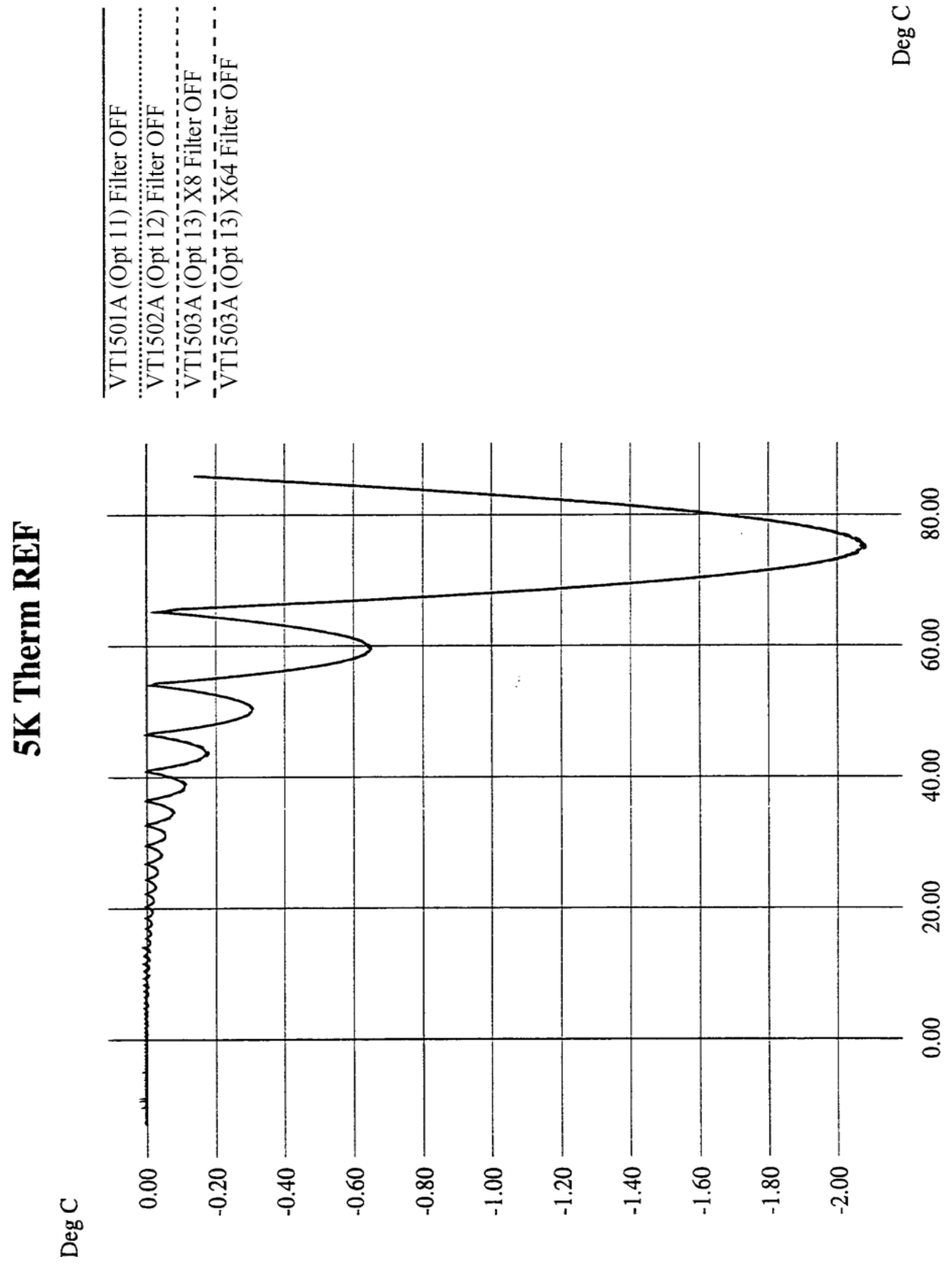


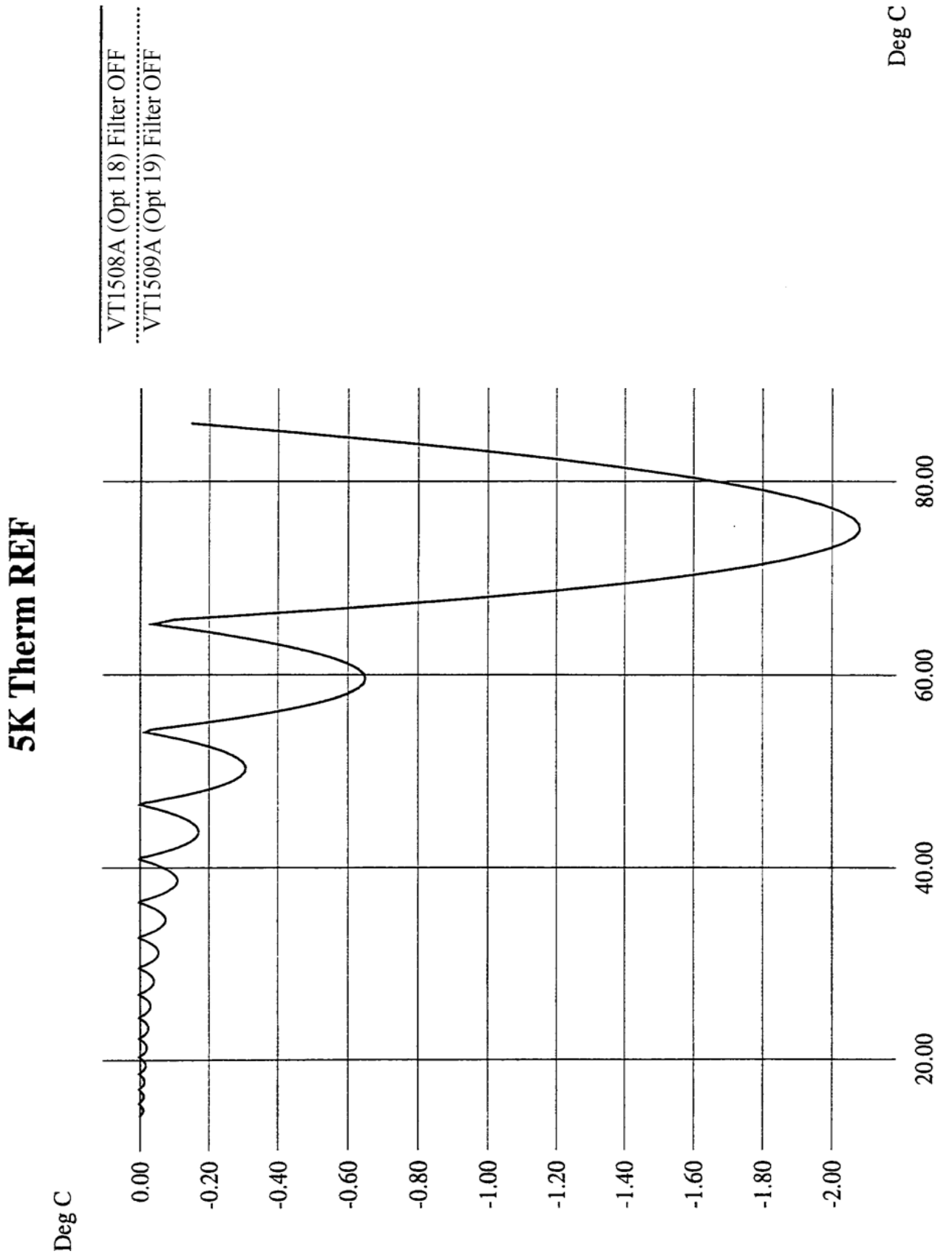


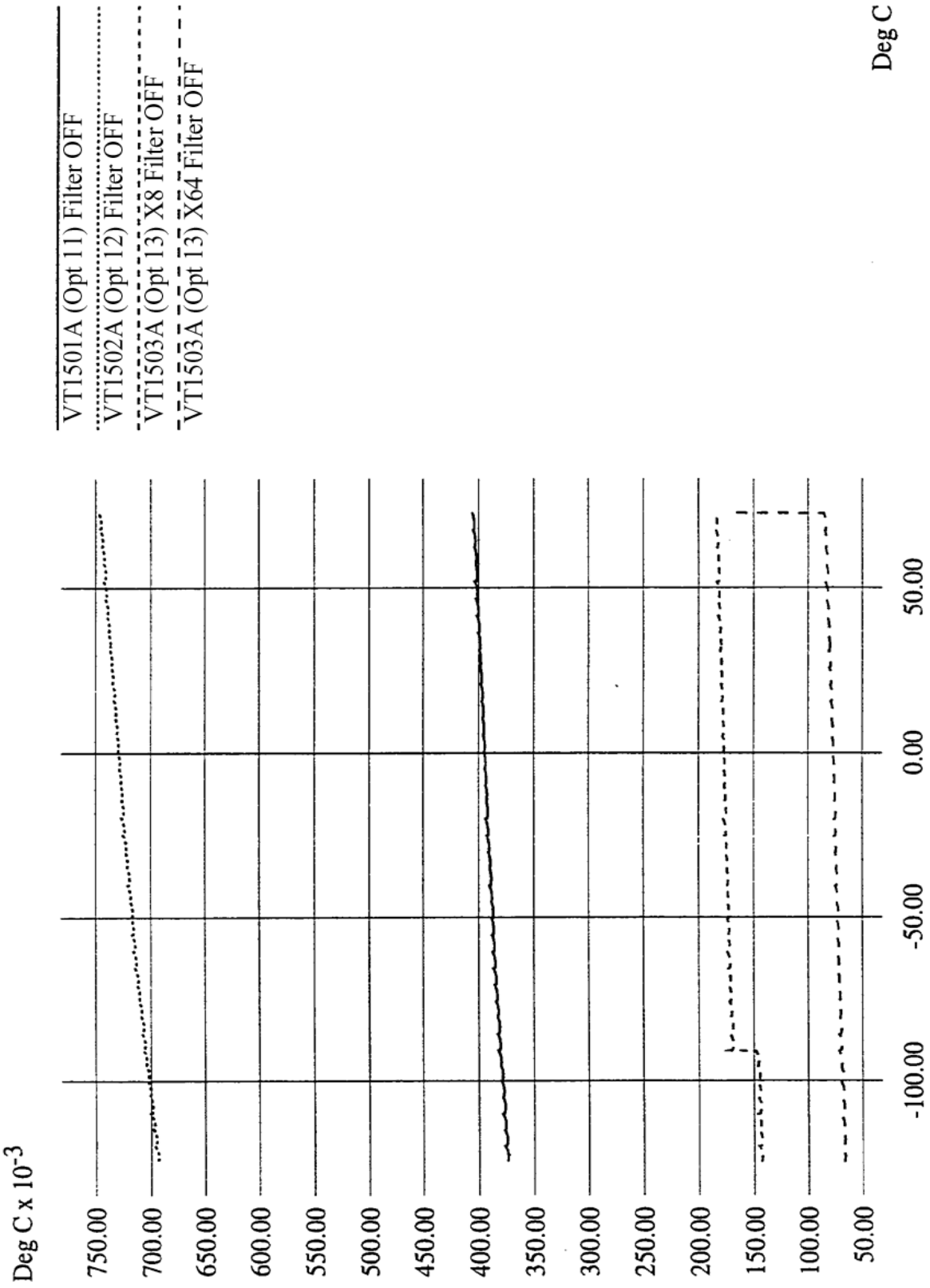


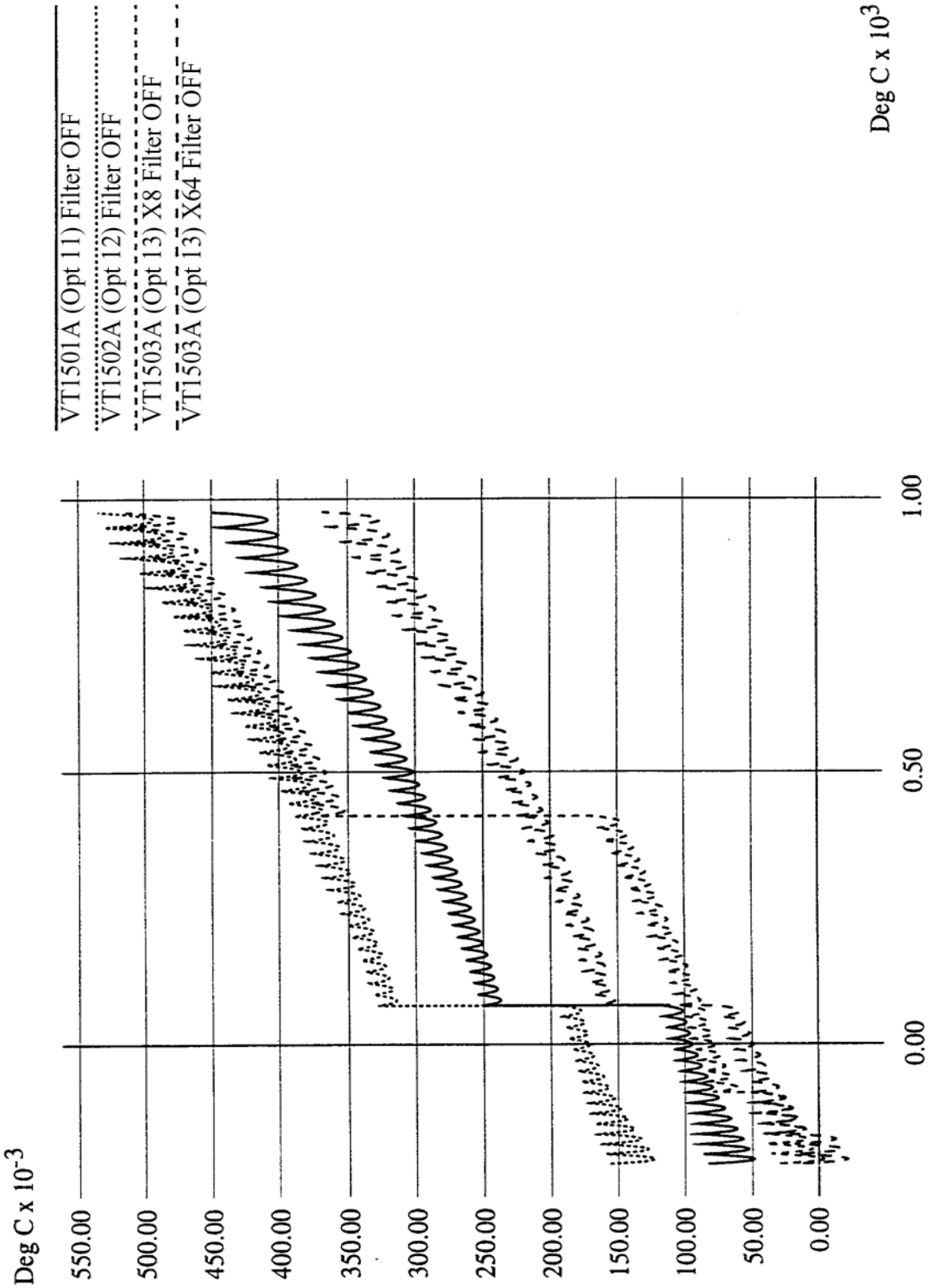


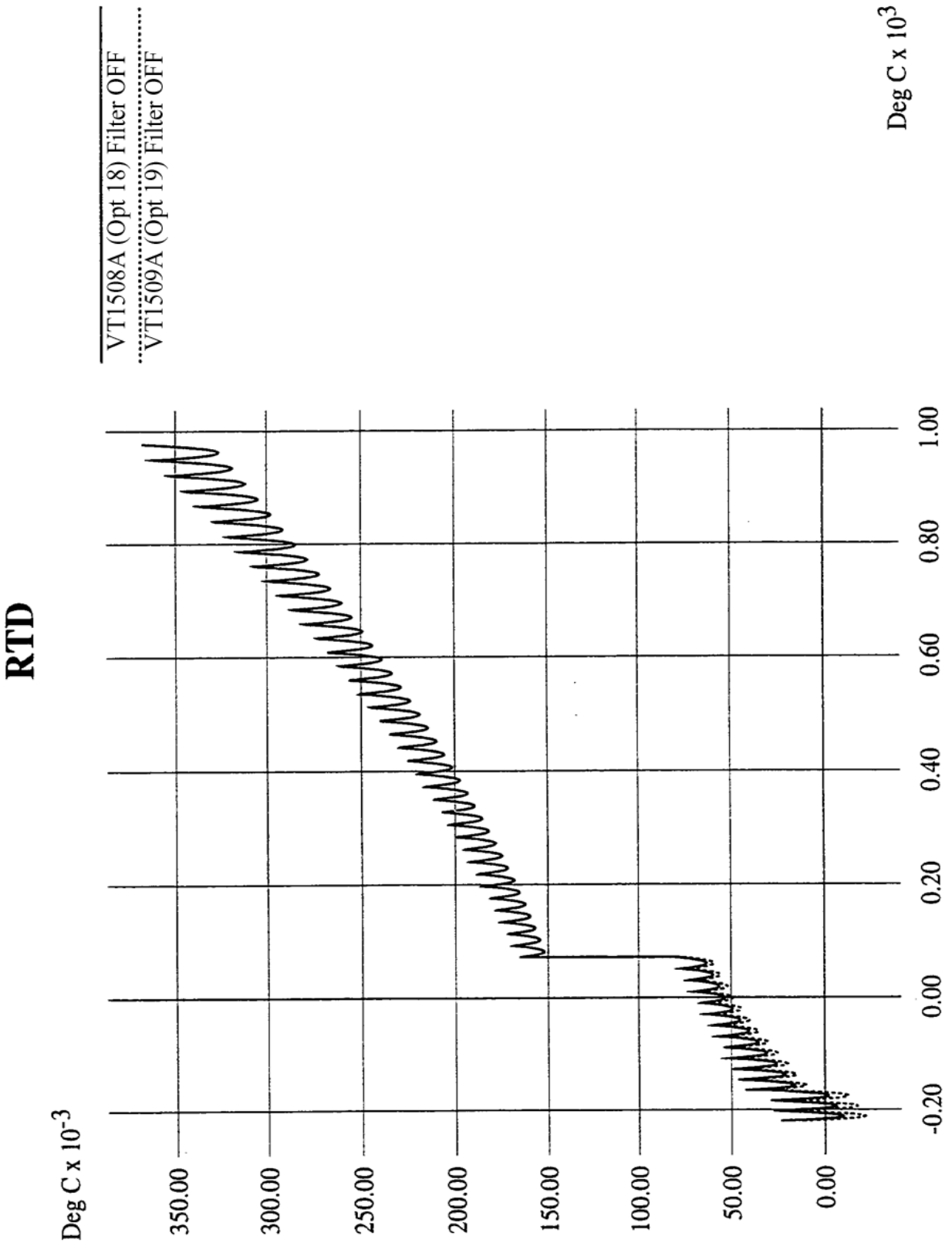


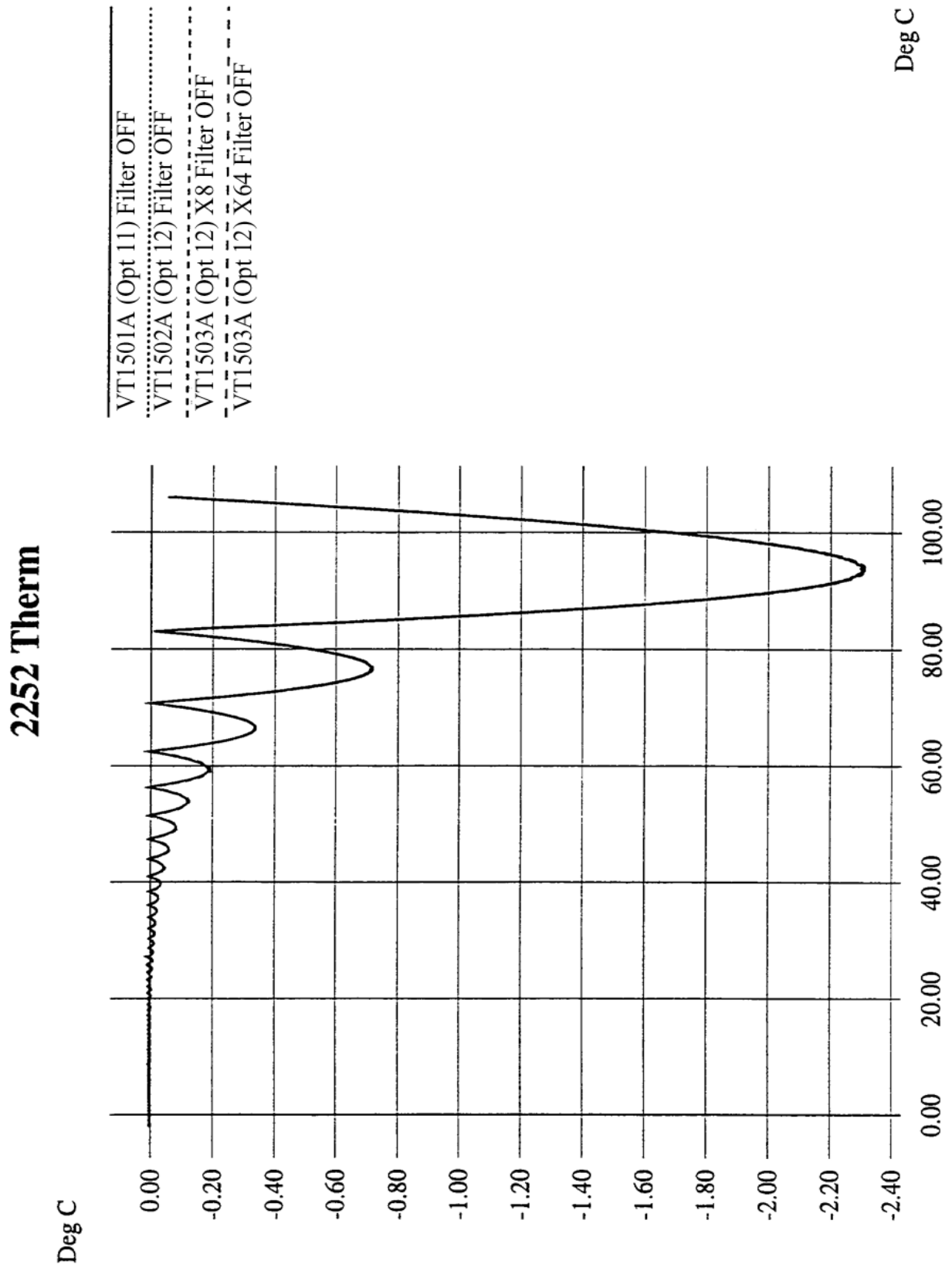


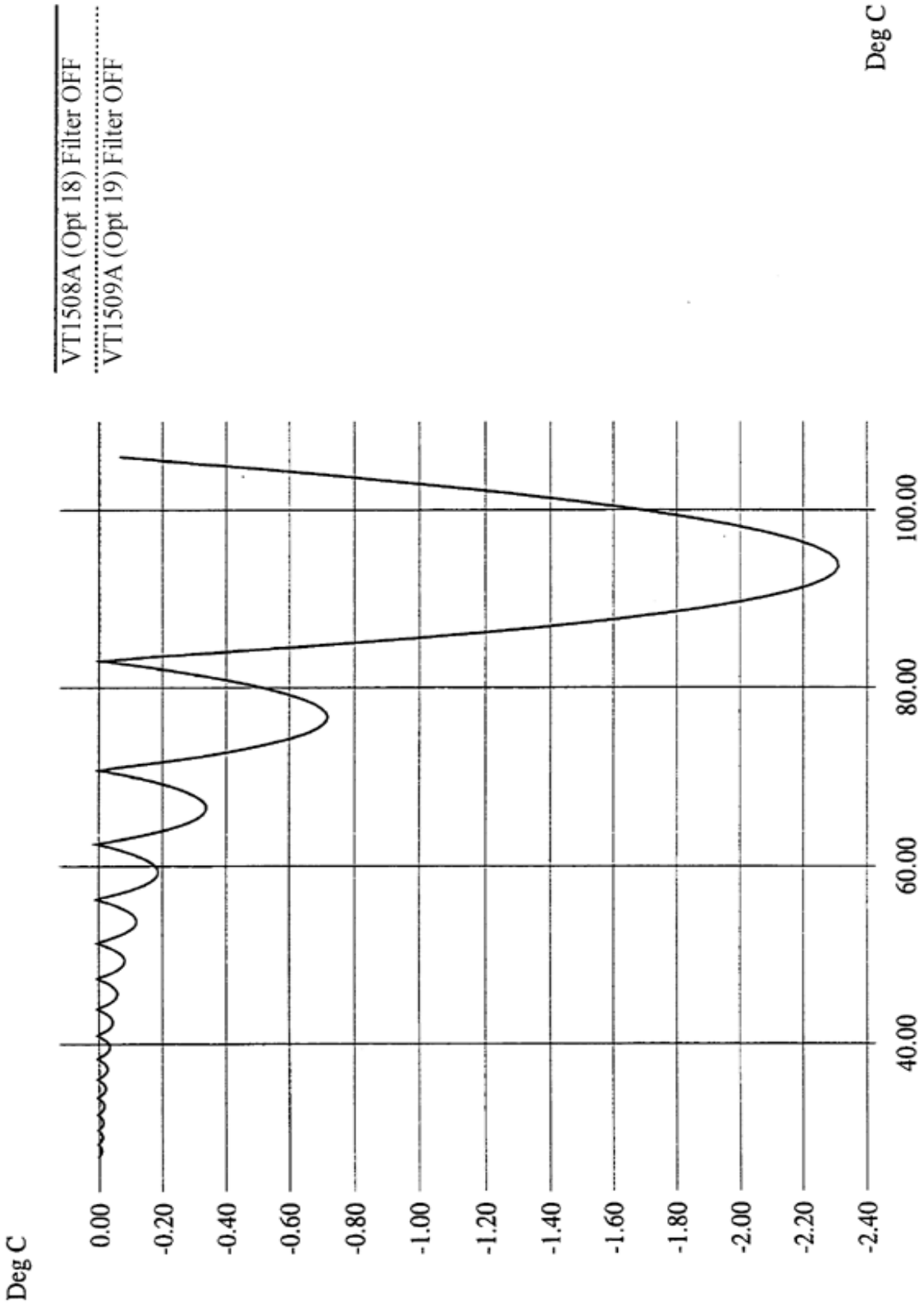


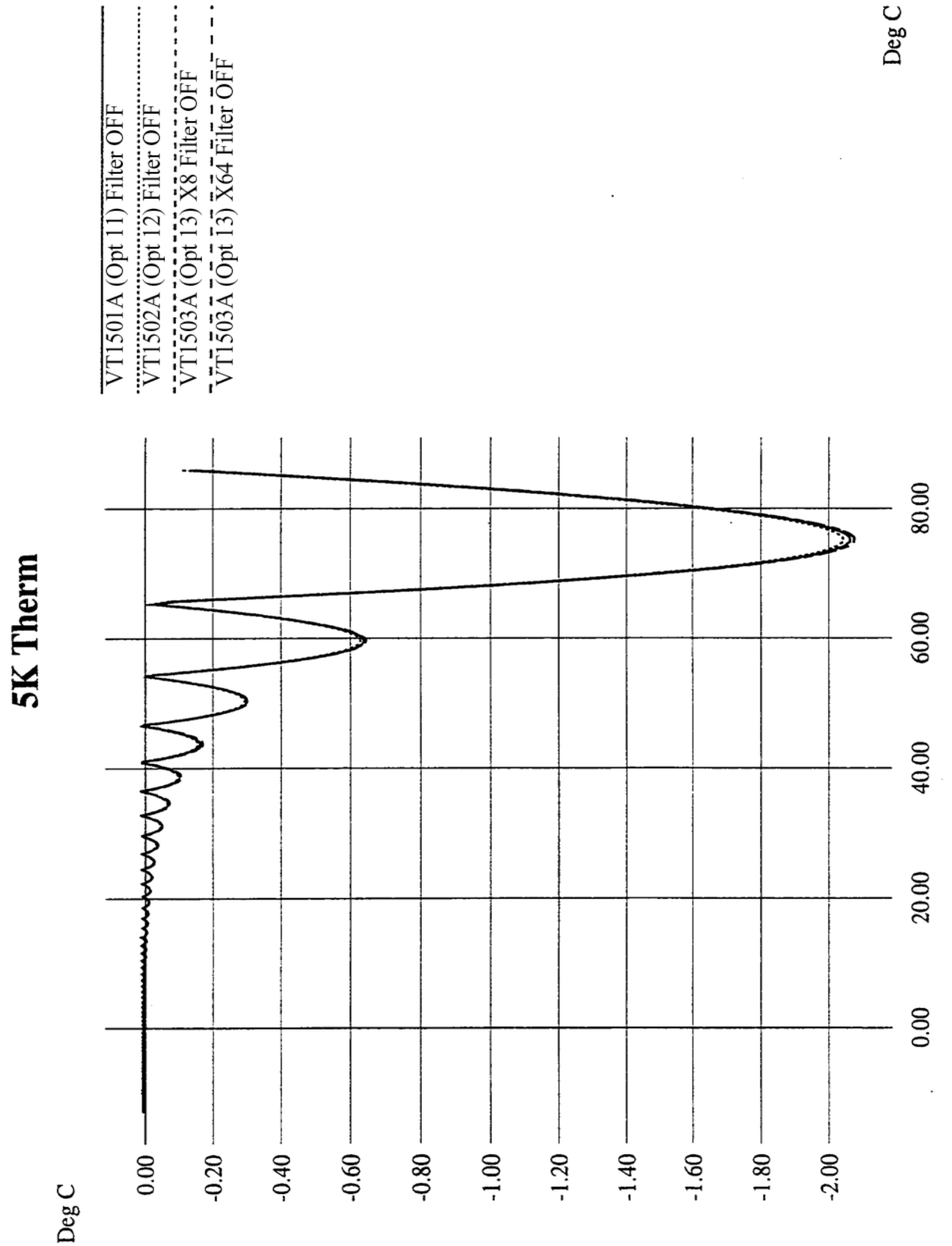


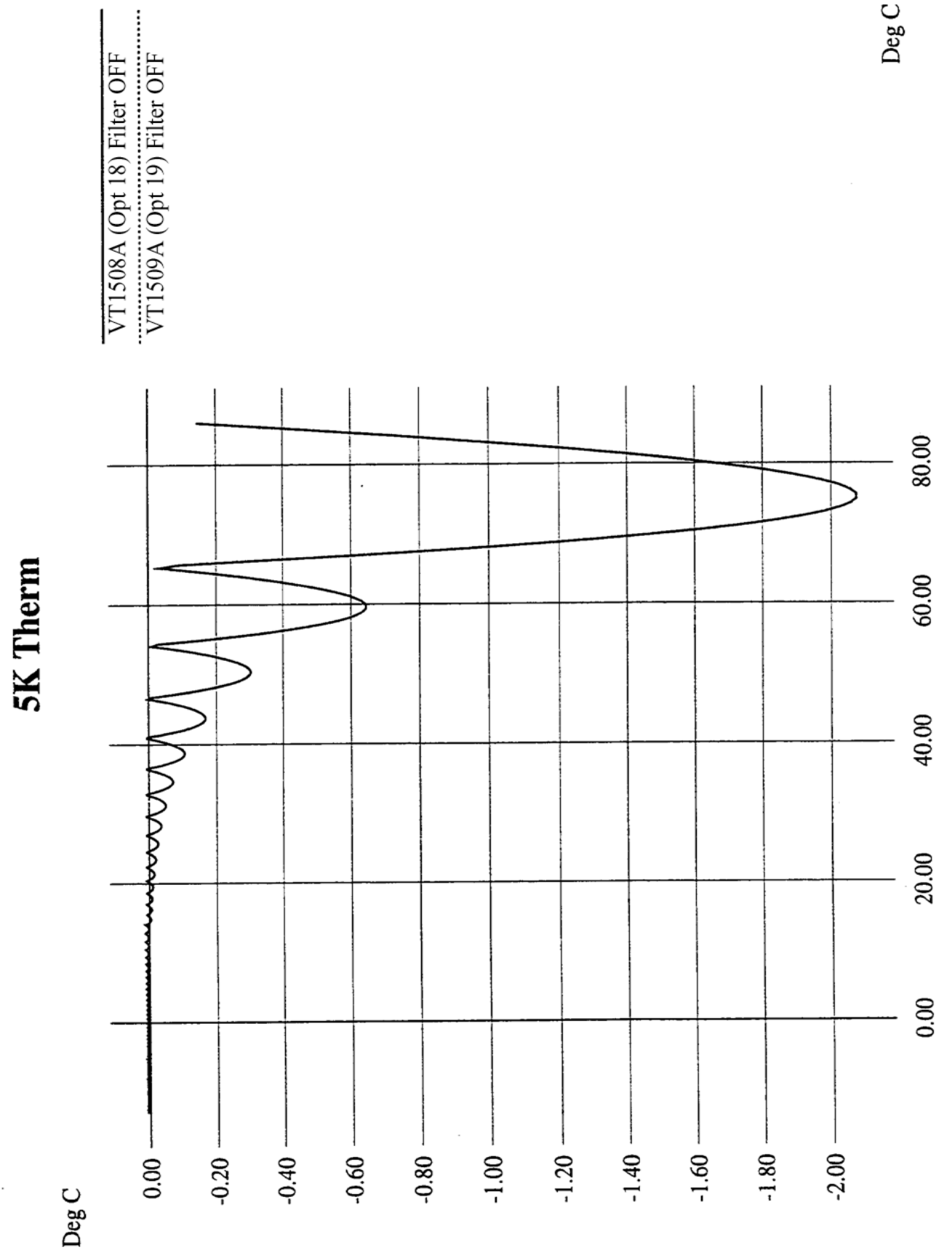


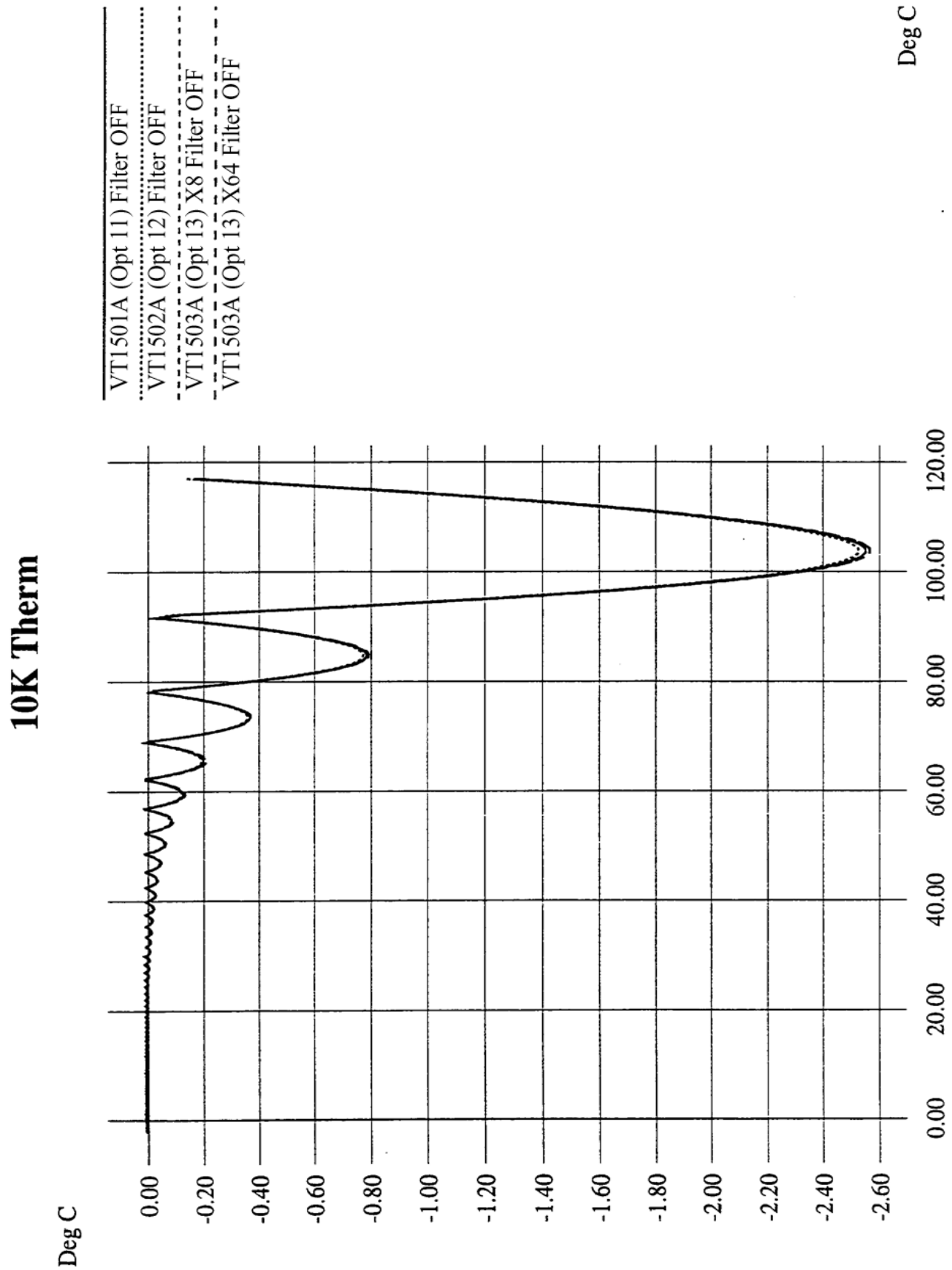


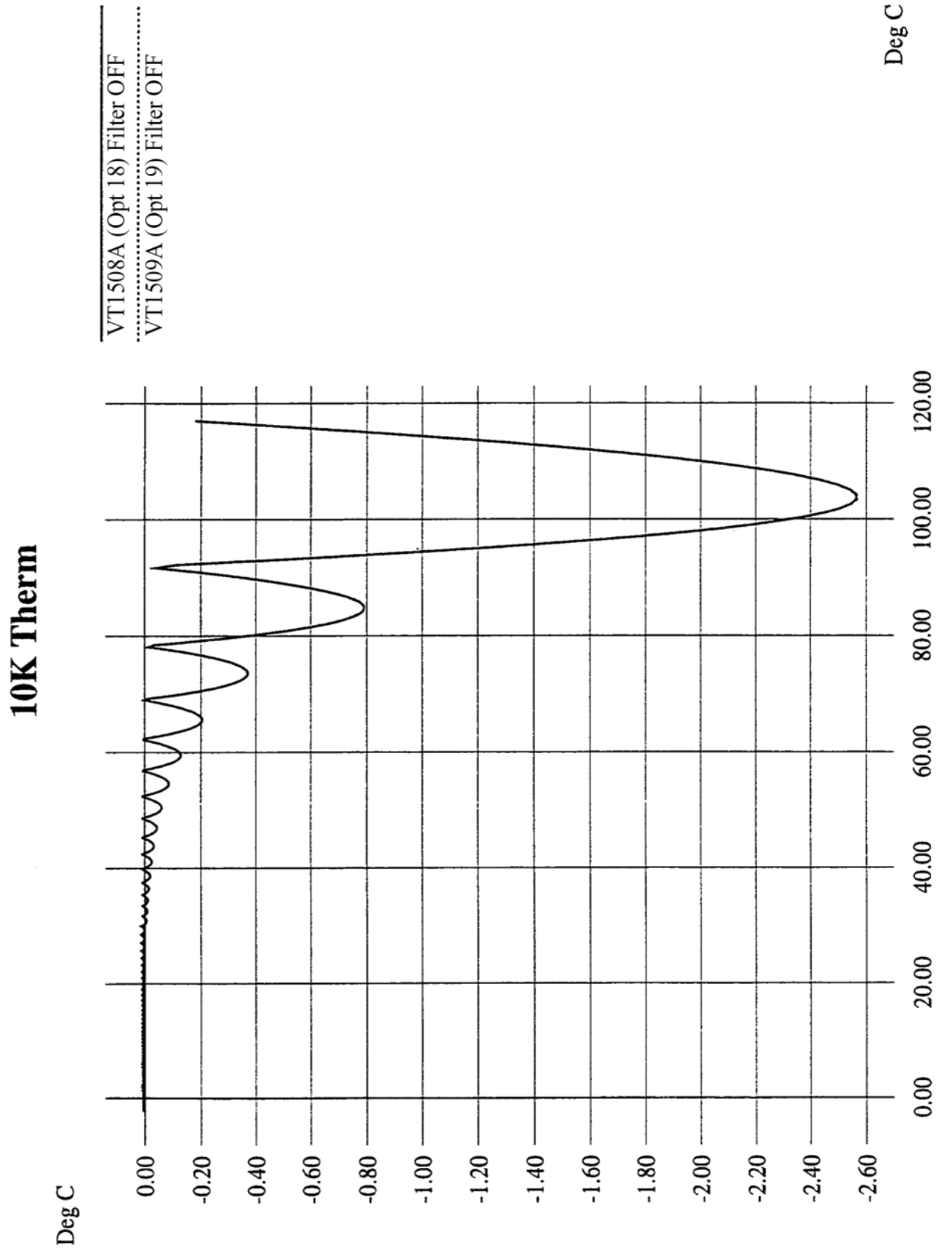












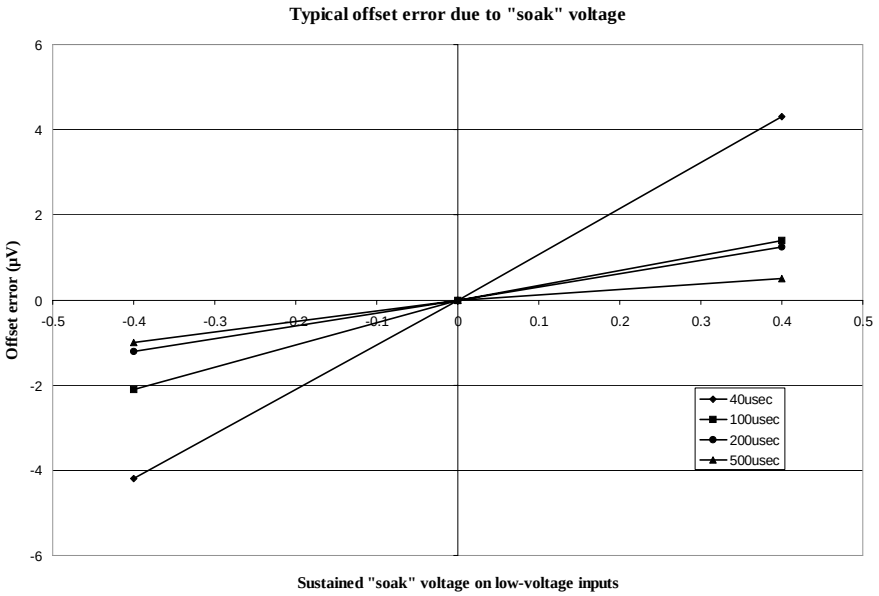
VT1529A/B Specifications

General:	
VT1529A/B outputs:	Single static output from 32:1 multiplexer 32 individually buffered dynamic outputs
Bridge completion:	120, 350 Ω and user installed, program selectable
Bridge configurations:	Full, half and quarter
Remote operation:	330 m (1000 ft) from multiplexed output 100 m (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 kΩ and user-installed shunt calibration resistor, program selectable
Measurement rate:	25 kSamples/s via multiplexed output, up to 196 kSamples/s dynamic
Static (multiplexed) outputs:	
Gain (VT1529A/B only)	32 V per V
Gain (VT1529A/B + VT1422A)	5000 V per V
Resolution (1 LSB @ VT1422A)	0.06 mV (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 VXI Technology's CT-400 VXI mainframe.
Static strain measurements	VT1422A Remote Channel DAC Module plus up to eight VT1539A SCPs
Dynamic strain measurements	VT1432A 16 Chan 51.2 kSamples/s Digitizer VT1433A 8 Chan 196 kSamples/s Digitizer

Bridge Specifications	
Completion resistors:	
Values	120/350 Ω ± 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 mV), ±4 °C of tare cal
Excitation sense & high-level DCV:	
Maximum input	±8 V dc (16 V centered about Gnd terminal)
Gain accuracy	±0.01% of reading
Offset	<1 mV

Strain Measurement (Combined VT1422A + VT1529A/B)		
Measurement range (με)		
(Quarter bridge, ±5 V excitation):	**Resolution (με)**	**rms noise (με)**
±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: Typical offset depends on previous channel value as shown in figure below.



Gain Error:	<0.015% of reading
rms Noise:	<1 mV rms
CMRR:	>100 dB, dc-10 MHz (common mode range ± 10 V)
Drift:	Note: drift errors can be removed by running CAL routine
Offset drift	<1 mV/°C <1 mV/month
Gain drift	<30 ppm/°C
Dynamic Outputs:	
Gain:	32 V per V ±0.1% of reading
Offset:	<250 mV
Bandwidth:	>100 kHz
Equivalent Input Noise (E.I.N.):	<20 nv/sqr(Hz)
Quarter Bridge Bending Errors:	(5 V Excitation, GF=2)

$\mu\in$	**Error ($\mu\in$)**
-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)	
	$\mu\in$	**Error ($\mu\in$)**
	-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)	
	$\mu\in$	**Error ($\mu\in$)**
	-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 V ac ± 10%	CAT II (2500 V_{PEAK} transients)
Line frequency:	50 - 60 Hz
Input power:	16 VA

Environmental	
Pollution 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	
VT1529A/B:	3 year return to VXI Technology

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 7, 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.' Indicates 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 VT1422A 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.

FIFO 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 0.0625 V associated with failed test
201	A/D range 0.25 V associated with failed test
202	A/D range 1 V associated with failed test
203	A/D range 4 V associated with failed test
204	A/D range 16 V associated with failed test

Test ID	Corrective Actions
1 - 19, 21 - 29	(VXI Technology Service)*
20, 30 -37	Remove all SCPs and see if *TST? passes. If so, replace SCPs one at a time until the one causing the problem is found.
38 - 71	(VXI Technology 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	(VXI Technology Service)*

*Must send module to an VXI Technology Service Center for repair. Record information found in FIFO to assist the VXI Technology Service Center in repairing the problem.

Refer to the Command Reference under *TST? for a list of module functions tested.

Note During the first five 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 SCPs enabled.' Don't set TRIG:SOUR or ARM:SOUR to SCP with VT1510A or VT1511A 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 VT1506A or VT1507A.
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 7.

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 the 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.' The algorithm's constructs are using too much translator memory. The Agilent/HP E1406 needs more memory. Try breaking the 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 VXI Technology 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 be received if 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	'E1536A 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	'E1536A debounce.' Each referenced 4 Ch debounce bank must contain at least one input channel.
3110	'Channel specified is invalid for RVELocity function.' See the VT1538A SCP manual.
3111	'Multiple channels are specified in the reference channel list.' See the VT1538A SCP manual.
3112	'Channel specified is invalid for RPULse reference channel.' See the VT1538A SCP manual.
3113	'Channel specified is not on the same SCP as reference channel.' See the VT1538A SCP manual.
3114	'First channel on SCP cannot be used in RPULse output channel list.' See the VT1538A SCP manual.
3115	'Channels specified are not in ascending order.' See the VT1538A SCP manual.
3116	'Multiple channels specified are not grouped correctly.' See the VT1538A SCP manual.
3117	'Grouped channels are not adjacent.' See the VT1538A SCP manual.
3118	'Incomplete setup information for RPULse function.' See the VT1538A SCP manual.
3119	'RPULse reference channel must be defined as RVELocity type.' See the VT1538A SCP manual.
3120	'Minimum velocity parameter must not exceed maximum velocity parameter.' See the VT1538A SCP manual.
3121	'Query invalid for current channel usage with the configured RPULse mode.' See the VT1538A SCP manual.

3122	'This multiple channel function must not span multiple SCPs.' See the VT1538A SCP manual.
3123	'E1538A OE switch ON conflicts with this command.' See the VT1538A SCP manual.
3124	'E1538A OE switch OFF conflicts with this command.' See the VT1538A SCP manual.
3125	'E1538A VRS switch setting conflicts with OE switch setting.' See the VT1538A SCP manual.
3126	'E1538A VRS switch setting conflicts with PU switch setting.' See the VT1538A SCP manual.
3127	'Undefined E1538A stepper motor mode.' See the VT1538A SCP manual.
3128	'E1538A Input threshold calibration failure.' See the VT1538A 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=neither CVT nor 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 323.
3136	'Only 32 channels may be scanned on each VT1529A/B.' Maximum Scan List entries per VT1529A/B is 32, so if a channel is used more than once, some other channel must be left out on that VT1529A/B.
3137	'VT1539A SCP channel 0 not responding.' Indicates a communications error between the VT1539A 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 'VT1539A SCP channel 1 not responding.' Indicates a communications error between the VT1539A 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 'VT1529A/B data not received properly, cable connected?'

3141 'Gain of 0.0 not allowed for VT1529A/B 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 323.

3145 'Shunt ON/OFF allows only a single channel per VT1529A/B.' The shunt cal resistor in a VT1529A/B can only be connected to one channel at a time. The <*ch_list*> in the OUTPut:SHUNT contained references to more than one channel on the same VT1529A/B.

3146 'Communication error reading VT1529A/B calibration constants, defaults used.'

3147 'Checksum error reading VT1529A/B calibration constants, defaults used.' Do CAL:REMote? and CAL:REM:STORe on the affected VT1529A/B.

Appendix C
VT1529A/B Verification & Calibration

Introduction

This appendix describes the verification and calibration procedure for the VT1529A/B Remote Strain Conditioning Unit when used with the VT1422A and the VT1539A SCP.

Note The verification and calibration procedure for the VT1422A and all SCPs other than the VT1539A is identical to the procedures for the VT1415A Algorithmic Closed Loop Controller. The procedures and programs described in the VT1415A and VT1419A Service Manual should be used to verify and calibrate the VT1422A and the other SCPs. Please contact VXI Technology for service manual availability.

Recommended Equipment

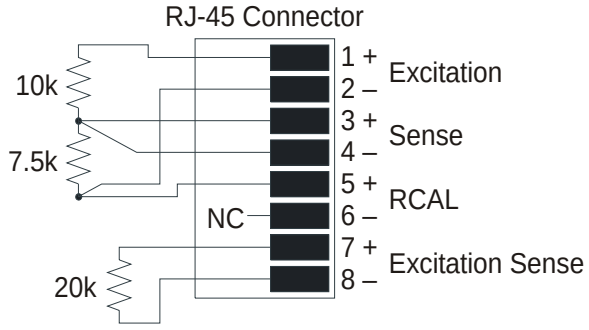
See the table below for test equipment recommended to test and service the VT1529A/B. Essential requirements for each piece of test equipment are listed in the Requirements column. Other equipment may be substituted if it meets the requirements listed in the table.

Instrument	Requirements	Recommended model	Use*
GPIB Controller	Pentium II 450 MHz or faster PC with GPIB card	GPIB = 82350A or 82357A Agilent I/O Libraries	V, A
VXI card cage and slot 0	Compatible with Agilent/HP E1406A Command Module	Agilent/HP E1406A	V, A
Programmable D/A Converter	Frequency ≥ 2 kHz	VT1531A SCP for VT1422A	V
Digital Multimeter	Voltage Range: ±20 V dc Resistance Range: 1 MΩ Current Range: 100 μA to 1 mA dc	Agilent/HP 3458A	V, A
Precision Calibration Source	Voltage Range: 5.000 V dc	Fluke 5700	V, A
Dummy Load	See “Dummy Load” below		V
*A = Adjustments, V= Verification Tests			

Dummy Load

Many of the tests require that a dummy load be connected to the VT1529A/B channel(s) being tested. With only one dummy load, each channel must be tested separately. This will take a long time, so it is recommended that one load be created for each channel on a VT1529A/B so that all channels can be tested simultaneously.

The circuit diagram for the dummy load is:



Notes:
Resistors are 0.1% tolerance
Connector is L-COM TPS3088 or equivalent
Wire is stranded 24 AWG, L-COM TDB8-X00 or equivalent

Figure C-1. Dummy Load Circuit

Tests

Remote calibration (CAL:REMOTE? command) performs calibration against the Remote Cal signal from the VT1422A which is traceable to a standard. The procedures in the Verification section test functional performance. They do *not* test the performance compared to specifications.

The assumed conditions for the performance verification tests are based on the following specifications/conditions:

- Minimum 1 hr warm-up for the VT1422A, VT1529A/B and Voltage Source.
- Test environment temperature 23 °C ± 1 °C.
- Linearity = ± 0.01% of reading (90-day specifications)
- *CAL? command executed before measurements. Executing CAL:TARE automatically executes *CAL?.
- CAL:ZERO? executed within five minutes after *CAL?.

Verification

The purpose of the verification tests is to verify that the VT1529A/B is functioning properly. Functional verification tests include:

Note For a quick functional check of the VT1529A/B, perform only the VT1422A Self-Test.

Note The VT1422A VXI*plug&play* Soft Front Panel can be used to enter the commands and queries described in the test procedures.

Test V-1: Self-Test

Description This test performs a self-test on the VT1529A/B by running the VT1422A self-test. This self-test checks much of the basic functionality of the VT1529A/B.

Setup The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The dummy load is not required for this test, but may be connected.

Procedure Send the DIAG:TEST:REM:SELF? (@<*ch_list*>) query to the VT1422A module. The expected return value is "+0".

Troubleshooting Failures One problem with the current VT1422A drivers is that a failure on one channel will can cause the VT1422A to report that all channels failed. This can make it difficult to find the channel that is failing. It may be easiest to let all the tests run and then look at the channels with the most failures.

See "DIAGnostic:TEST:REMote:SELFtest?" on page 288 for details on the self-test, return values and possible causes.

Test V-2: Cal Remote

Description This test performs a calibration of the VT1529A/B by running the VT1422A remote calibration command. The calibration sequence checks the results for reasonable values and reports errors.

This test requires that the time since the last *CAL? calibration be less than 8 hours. The test program (*perf_e1529.c*) insures this by forcing a *CAL at the beginning of the program.

Setup The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The dummy load is not required for this test, but may be connected.

Procedure If necessary, send the *CAL? query to the VT1422A (required if more than 8 hours has elapsed since the last *CAL? command). The expected return value is "+0".

Send the CAL:REMote? (@<*ch_list*>) query to the VT1422A module. The expected return value is "+0".

Troubleshooting Failures

A failure in the *CAL? command indicates a problem with the VT1422A or with one of the installed SCPs.

A failure in the CAL:REMote? command indicates a problem with the VT1529A/B, the VT1539A SCP or the cable between the VT1529A/B and the VT1539A.

Test V-3: Sense Out

Description

This test verifies the operation of the excitation sense circuitry of the VT1529A/B. It measures the excitation voltage with the dummy load attached to each channel. This voltage should be one-half the applied excitation voltage due to the voltage divider effect between the 20 kΩ resistor in the dummy load and the 10 kΩ resistors between the excitation bus and the Excitation Sense + and - terminals on channel input in the VT1529A/B.

Note that the actual measurements will vary based on the actual resistor values in the dummy load, so these values should be considered approximate when running this test.

Setup

The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The Dummy Load must be connected to the channel(s) being tested.

The Voltage Source must be connected to the appropriate channel bank terminals of the Bridge Excitation input on the VT1529A/B.

The signal path through the VT1529A/B is highlighted in Figure C-2.

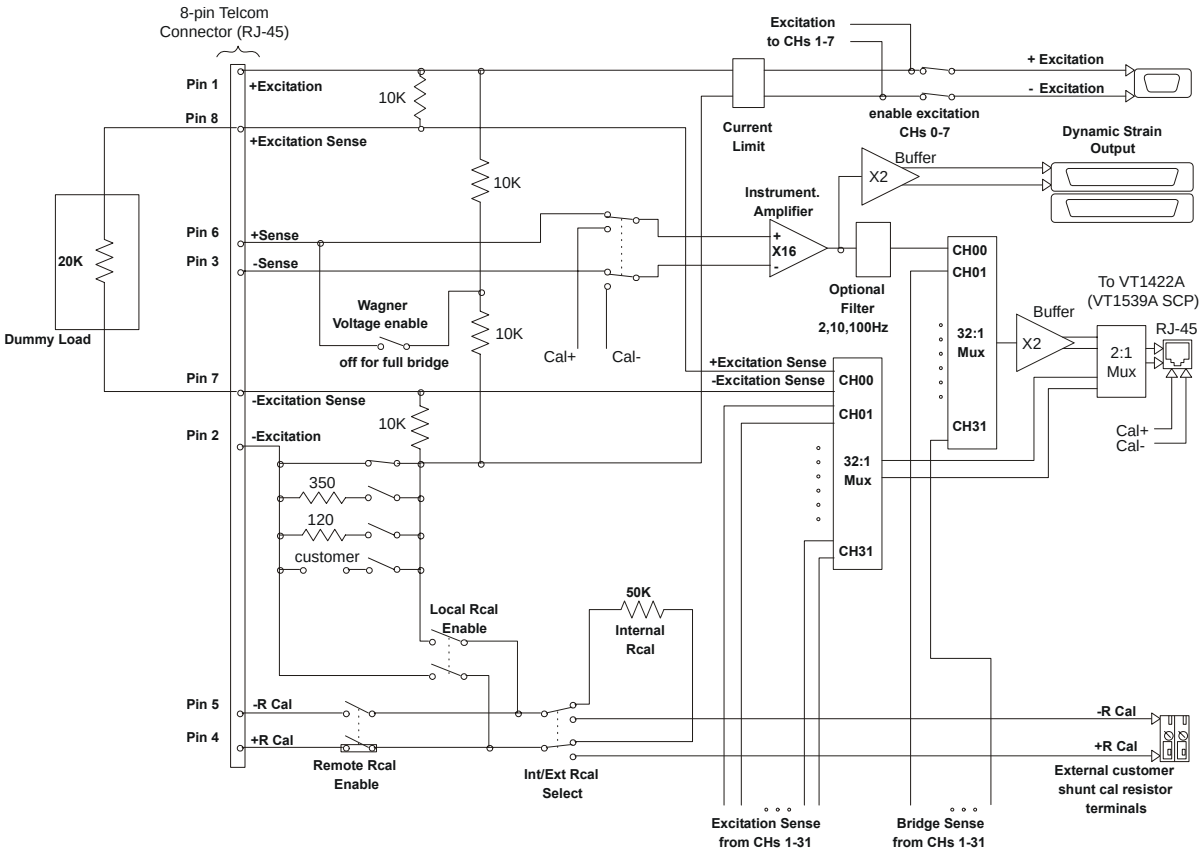


Figure C-2. Signal Path for Sense Out Test

Procedure

Set the Voltage Source to +5.000 V. Send the MEAS:VOLT:EXCitation? (@<*ch_list*>) command to the VT1422A. The expected measurement value is approximately +2.5 V on each channel.

Troubleshooting Failures

Check that the Dummy Load is wired and working correctly and firmly seated in the RJ-45 input connector. If all 8 channels of one block of VT1529A/B channels are failing, check that the Voltage Source is connected to the correct terminals on the Bridge Excitation input on the VT1529A/B.

Test V-4: Bridge Resistors

Description

This test verifies the operation of the internal bridge completion resistors and associated switches in the VT1529A/B.

The operation of the internal bridge resistors can be measured by unbalancing the bridge circuit. The bridge output voltage is a function of the resistor value as given by the following formula:

$$V_{sense} = V_{excitation} * (10.0 / (10.0 + 7.5 + R) - 0.5)$$

where R is the completion resistor value in kΩ.

With $V_{excitation}$ = 5.000 V, the nominal output voltages can be computed:

for R = 0 Ω, Vsense = 0.357143 V
for R = 120 Ω, Vsense = 0.337684 V
for R = 350 Ω, Vsense = 0.301120 V

Note that the actual measurements will vary based on the actual resistor values in the dummy load, so these values should be considered approximate when running this test.

Setup

The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The Dummy Load must be connected to the channel(s) being tested.

The Voltage Source must be connected to the appropriate channel bank terminals of the Bridge Excitation input on the VT1529A/B.

The signal path through the VT1529A/B is highlighted in Figure C-3.

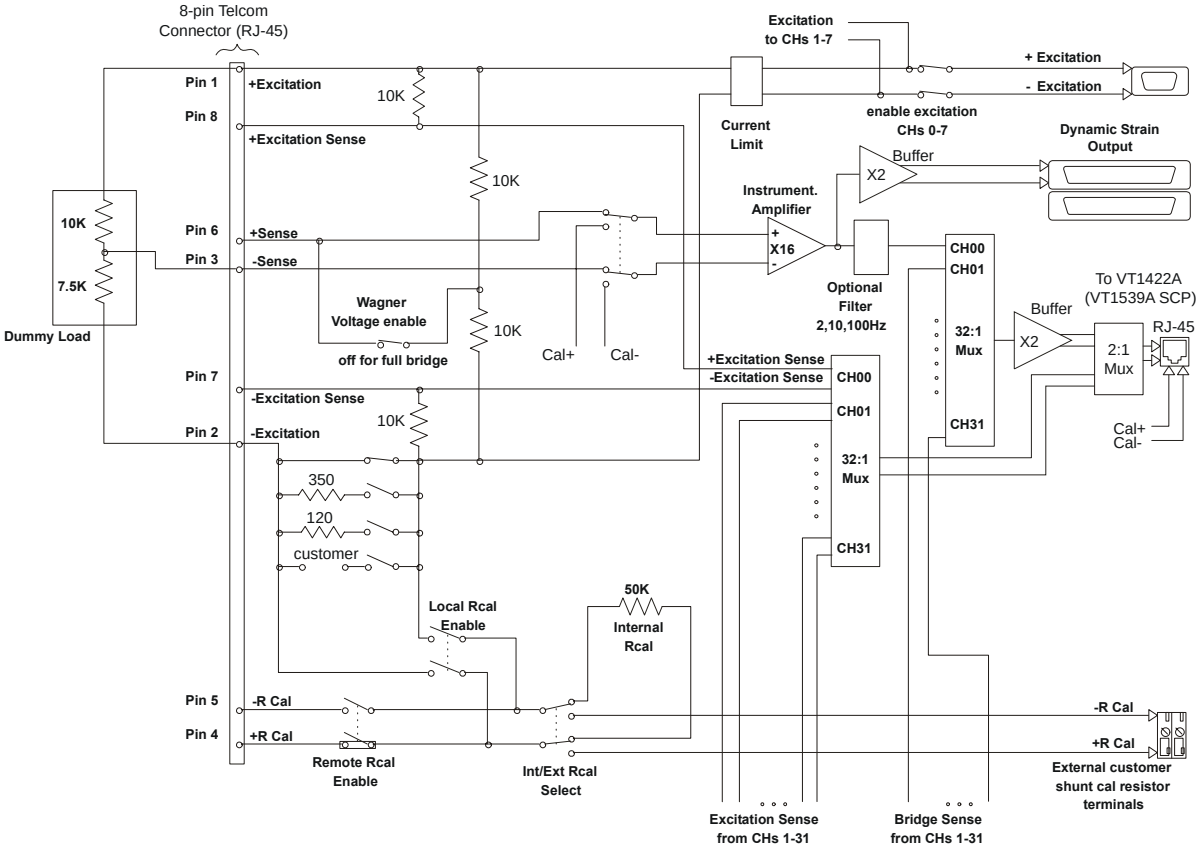


Figure C-3. Signal Path for Bridge Resistor Test

Procedure

If necessary, send the *CAL? query to the VT1422A (required if more than eight hours has elapsed since the last *CAL? command). The expected return value is “+0”.

Set the Voltage Source to 5.000 V.

Configure the VT1529A/B for bending half-bridge operation using the SENS:FUNC:STR:HBEN (@<*ch_list*>) command. This switches in the 0 Ω

resistor path. Send the MEAS:VOLT:UNStrained? (@<*ch_list*>) query to the VT1422A. The expected return value is ~0.357 V on each channel.

Configure the VT1529A/B for 120 Ω quarter-bridge operation using the SENS:FUNC:STR:Q120 (@<*ch_list*>) command. Send the MEAS:VOLT:UNStrained? (@<*ch_list*>) query to the VT1422A. The expected return value is ~0.338 V on each channel.

Configure the VT1529A/B for 350Ω quarter-bridge operation using the SENS:FUNC:STR:Q350 (@<*ch_list*>) command. Send the MEAS:VOLT:UNStrained? (@<*ch_list*>) query to the VT1422A. The expected return value is ~0.301 V on each channel.

Troubleshooting Failures

Check that the Dummy Load is wired and working correctly and firmly seated in the RJ-45 input connector. If all 8 channels of one block of VT1529A/B channels are failing, check that the Voltage Source is connected to the correct terminals on the Bridge Excitation input on the VT1529A/B.

Test V-5: Dynamic Strain Output Port

Description

Test the dynamic output gain and offset of the VT1529A/B.

The dynamic strain output voltage should be equal to V_{SENSE} times the gain of the output amplifier, which is fixed at 32.

With $V_{EXCITATION}$ = 5.000 V, the Dummy Load attached and the VT1529A/B configured for the 0 Ω sense path, the nominal dynamic output voltage can be computed:

for R = 0 Ω, V_{sense} = 0.357143 V, $V_{dynamic}$ = 11.4286 V

Setting Vexcitation to 0 V, verify the operation of the dynamic output offset control DAC by setting the offset to +1.0 V. The dynamic output voltage should be the same as the DAC value, +1.0 V.

Note that the actual measurements will vary based on the actual resistor values in the dummy load, so these values should be considered approximate when running this test.

Note

This test does *not* test the offset or bandwidth specifications.

Setup

The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The Dummy Load must be connected to the channel(s) being tested.

The Voltage Source must be connected to the appropriate channel bank terminals of the Bridge Excitation input on the VT1529A/B.

The Digital Multimeter should be connected to the first channel on the Dynamic Strain Output Port.

The signal path through the VT1529A/B is highlighted in Figure C-4.

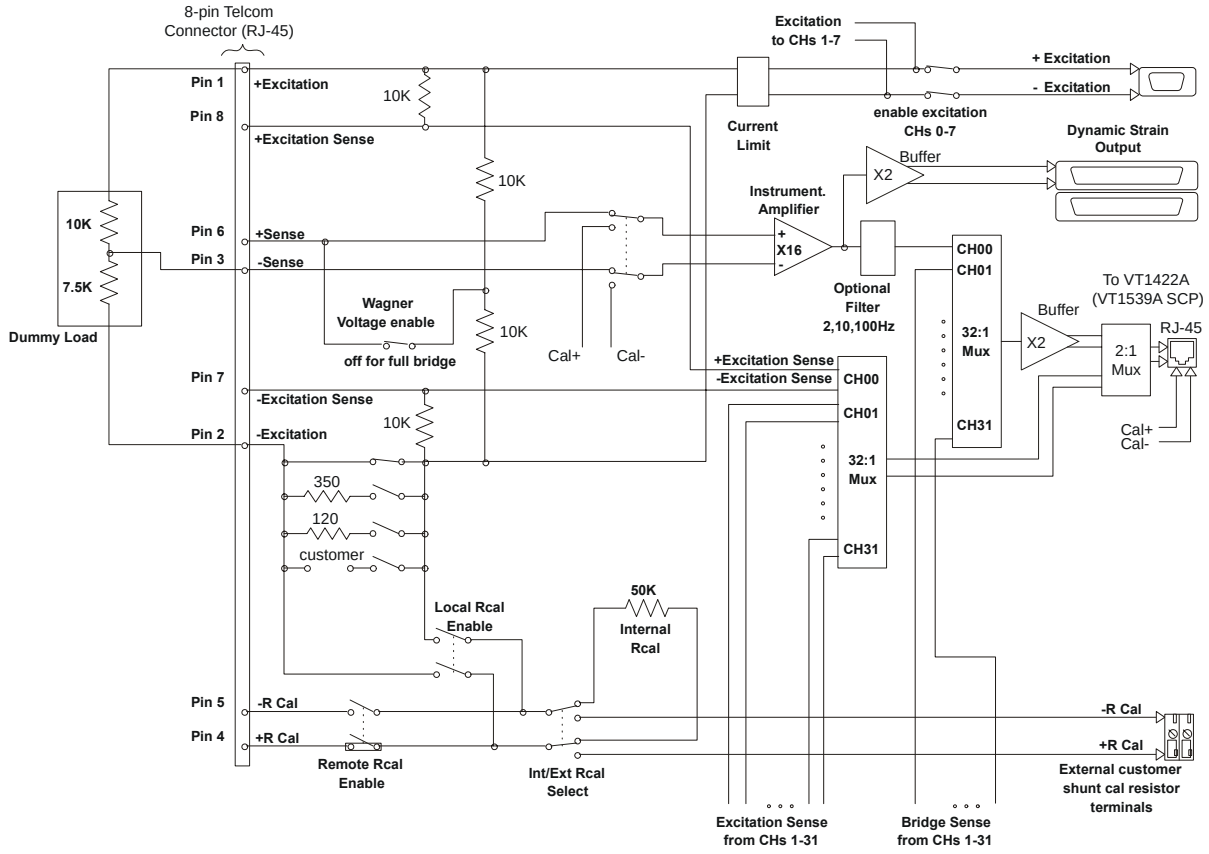


Figure C-4. Signal Path for Dynamic Strain Output Port Test

Procedure

If necessary, send the *CAL? query to the VT1422A (required if more than 8 hours has elapsed since the last *CAL? command). The expected return value is “+0”.

Set the Voltage Source to 5.000 V.

Configure the VT1529A/B for bending half-bridge operation using the SENS:FUNC:STR:HBEN (@<*ch_list*>) command. This switches in the 0 Ω resistor path. Send the MEAS:VOLT:UNStrained? (@<*ch_list*>) query to the VT1422A. The expected return value is ~0.357 V on each channel.

Use the Digital Multimeter to measure the output voltage on each channel on the Dynamic Strain Output Port. The expected value is ~11.43 V.

After measuring all channels, set the Voltage Source to 0.0 V. Send the SOUR:VOLT:AMPL 1.0,<*ch_list*> command to set the offset DAC for each channel to 1.0 V.

Use the Digital Multimeter to measure the output voltage on each channel on the Dynamic Strain Output Port. The expected value is 1.0 V.

Troubleshooting Failures

Check all connections and the Dummy Load.

Test V-6: Filters

Description

Tests the low-pass filters for each channel in the VT1529A/B. The nominal filter bandwidths are 2 Hz, 10 Hz, 100 Hz, and 20 kHz.

The filter bandwidth is calculated by measuring the rise time of the step response. The bandwidth is related to the rise time by the equation:

Bandwidth = 0.35 / Rise Time

The VT1422A cannot sample fast enough to measure the rise time of the 20 kHz filter. The test procedure for this filter checks that the bandwidth is greater than 800 Hz, which is sufficient to verify that none of the other filters are being used.

Setup

The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The Dummy Load must be connected to the channel(s) being tested.

The VT1531A Voltage Output SCP should be installed on the VT1422A. If needed, remove another SCP so that the VT1531A can be installed.

Make the following connections between the VT1531A and the Bridge Excitation input port on the VT1529A/B front panel. This connection table assumes that RJ-45 terminal block for the VT1422A is being used.

Bridge Excitation Input	Bridge Excitation Pin	VT1531A Output	RJ-45 connector as labeled on terminal block	RJ-45 pin
Ch 0-7 +	1	Ch 0 +	8	1
Ch 0-7 -	2	Ch 0 -	8	2
Gnd	3	Shield Gnd	8 or 9	Shield
Ch 8-15 +	4	Ch 1 +	9	1
Ch 8-15 -	5	Ch 1 -	9	2
Ch 16-23 +	6	Ch 2 +	8	3
Ch 16-23 -	7	Ch 2 -	8	6
Ch 24-31 +	8	Ch 3 +	9	3
Ch 24-31 -	9	Ch 3 -	9	6

See Figure 3-7, “Excitation Supply Connections”, on page 68 and Figure 2-4, “RJ-45 Connector Module and Pin-out”, on page 41 for pin-out information for the Bridge Excitation port on the VT1529A/B and the RJ-45 connector ports for the VT1531A.

Procedure If necessary, send the *CAL? query to the VT1422A (required if more than eight hours has elapsed since the last *CAL? command). The expected return value is “+0”.

Each channel of the VT1529A/B must be tested individually. Follow the steps below for each channel to be tested.

Use the VT1422A VXI*plug&play* Soft Front Panel to select Reset from the System menu. Then, select Algorithm from the Panels menu. Create the algorithm below. Note that this algorithm assumes the VT1531A SCP is in SCP position 1; it is necessary to modify the VT1531A channel numbers (O108, O109, O110, O111) if the VT1531A is in a different SCP location. It is also necessary to modify the VT1529A/B channel (I10000) depending on the channel to be tested. When algorithm creation is finished, click on the Down Load... button to download the algorithm.

```
static float loop = 0.0;
static float wrote_fifo = 0.0;

if (First_loop)
{
   loop = 0;
   O108 = 0;
   O109 = 0;
   O110 = 0;
   O111 = 0;
}
else
{
    loop = loop + 1.0;
    if (loop == 1000.0)
    {
        if (I10000 > 0.001)
        {
             writefifo(-2);
             wrote_fifo = 1;
        }
        O108 = -1.0;
        O109 = -1.0;
        O110 = -1.0;
        O111 = -1.0;
    }
    if (!wrote_fifo)
    {
        if (loop > 1000.0 && I10000 > .39)
        {
            writefifo(1093.75 / (loop - 1000));
            wrote_fifo = 1.0;
        }
        else if (loop == 1999.0)
            writefifo(-1.0);
    }
}
```

From the Panels menu, select Main Panel, then enter the following commands on the Enter Command line in the Interactive Communications box to test the 100 Hz filter. Note that it is necessary to change the channel number (10000) in these commands depending on the channel being testing:

```
samp:tim min
trig:sour tim;tim 400e-6;count 2000
rout:seq:def (@0(10000))
func:volt (@10000)
inp:filt:freq 100,(@10000);stat on,(@10000)
init
data:fifo?
```

This should display a value between about 85 and 100, which is the computed bandwidth of the 100 Hz filter. This value should be considered only as an approximate value given the limited resolution on the measurement.

To test the 10 Hz filter, now enter the following commands (again, modify the channel number as appropriate):

```
inp:filt:freq 10,(@10000)
init
data:fifo?
```

This should display a value between about 9 and 10, which is the approximate bandwidth of the 10 Hz filter.

To test the 2 Hz filter, now enter the following commands (again, modify the channel number as appropriate):

```
inp:filt:freq 2,(@10000)
init
data:fifo?
```

This should display a value of approximately 1.9 to 2.2.

Finally, to test the 20 kHz filter, enter (again, modify the channel number as appropriate):

```
inp:filt:stat off,(@10000)
init
data:fifo?
```

This should display a value greater than 1000.

Troubleshooting Failures

Use the Digital Multimeter to verify that the VT1531A outputs transition from 0 V to 5 V after the INITiate:IMMediate command is sent.

Test V-7: Shunt Cal Resistor Port

Description

This test verifies the operation of the switches associated with the front panel Shunt Cal Resistor port. The operation of these switches can be verified by observing the changes in the bridge output voltage when external shunt calibration is disabled and enabled.

A 1 kΩ external shunt resistor is connected to the Shunt Cal Resistor port and the Dummy Load is attached. The 350 Ω bridge completion resistor path is configured. With the external shunt disabled, the bridge output voltage should be:

$$V_{sense} = V_{excitation} * (10 / (10 + 7.5 + .350) - 0.5)$$

With $V_{excitation}$ = 5.0 V, the expected value of V_{sense} is 0.301120 V.

With the external shunt enabled, the bridge output voltage should be:

$$V_{sense} = V_{excitation} * (10 / (10 + 7.5 + R) - 0.5)$$

Where R equals the parallel combination of the 350 Ω resistor and the 1 kΩ external shunt or 259.26 Ω. Again, with $V_{EXCITATION}$ = 5.0 V, the expected value of V_{sense} is 0.315433 V.

Note that the actual measurements will vary based on the actual resistor values in the dummy load, so these values should be considered approximate when running this test.

Setup

The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The Dummy Load must be connected to the channel(s) being tested.

The Voltage Source must be connected to the appropriate channel bank terminals of the Bridge Excitation input on the VT1529A/B.

The signal path through the VT1529A/B is highlighted in Figure C-5.

Procedure

If necessary, send the *CAL? query to the VT1422A (required if more than 8 hours has elapsed since the last *CAL? command). The expected return value is “+0”.

Set the Voltage Source to 5.000 V.

Configure the VT1529A/B for 350 Ω quarter-bridge operation using the SENS:FUNC:STR:Q350 (@<*ch_list*>) command. Disable shunt calibration by sending the OUTP:SHUNt OFF,(@<*ch_list*>) command. Send the MEAS:VOLT:UNStrained? (@<*ch_list*>) query to the VT1422A. The expected return value is ~0.301 V on each channel. Now, configure for external shunt using OUTP:SHUNt:SOURce EXTernal,(@<*ch_list*>).

For each channel (each must be tested separately), enable external shunt calibration by sending the OUTP:SHUNt ON,(@<*channel*>) command.
Note: *this turns off the shunt on the previous channel.*
Send the MEAS:VOLT:UNStrained? (@<*channel*>) query to the VT1422A. The expected return value is ~0.315 V on each channel.

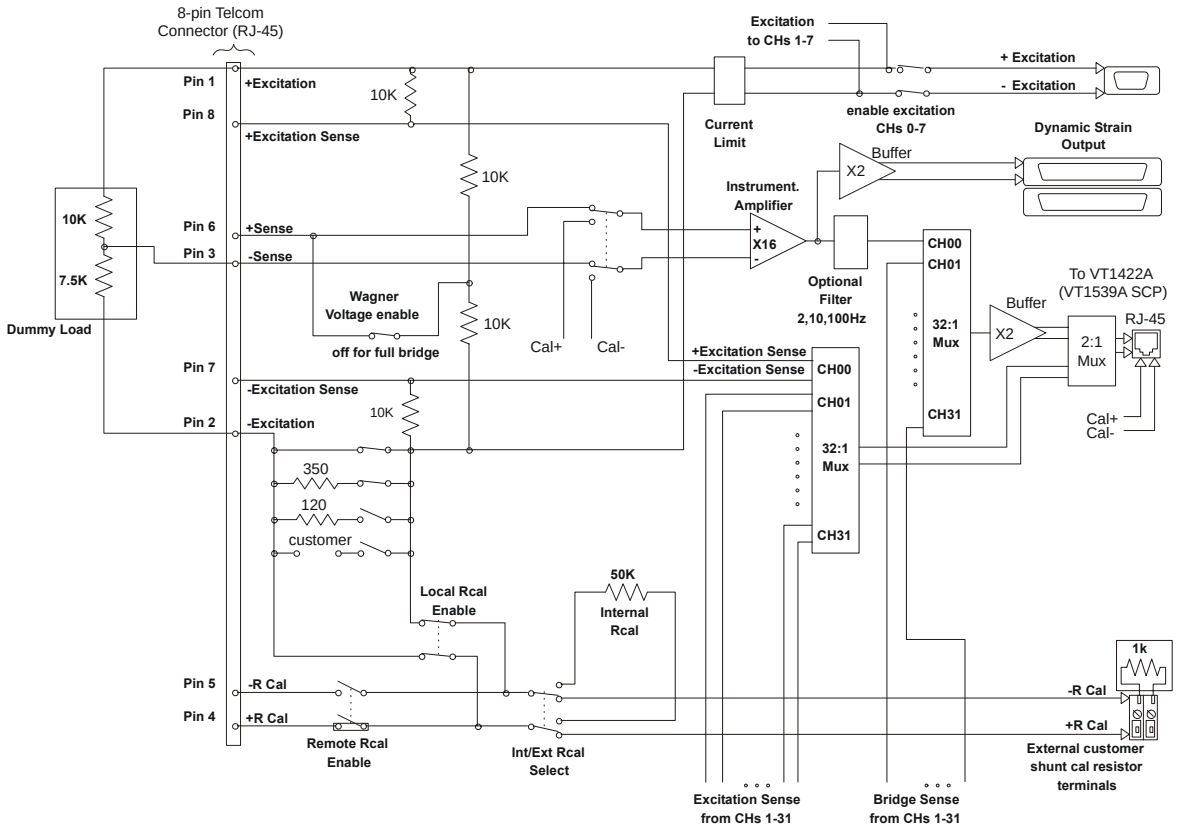


Figure C-5. Signal Path for Shunt Cal Resistor Port Test

Test V-8: Internal Shunt

Description

This test verifies the operation of the internal 50 kΩ shunt resistor and the associated switches in the VT1529A/B.

Switch operation can be verified by observing the changes in the bridge output voltage when internal shunt calibration is disabled and enabled.

The bending half-bridge strain path is configured and the Dummy Load is attached. With the internal shunt disabled, the bridge output voltage should be:

$$V_{sense} = V_{excitation} * (10 / (10 + 7.5) - 0.5)$$

With $V_{excitation}$ = 2.0 V, the expected value of V_{sense} is 0.142857 V.

With the internal shunt enabled, the bridge output voltage should be:

$$V_{sense} = V_{excitation} * (10 / (10 + R) - 0.5)$$

Where R equals the parallel combination of the 7.5 kΩ resistor in the Dummy Load and the 50 kΩ internal shunt or 6.522 kΩ. Again, with $V_{excitation}$ = 2.0 V, the expected value of V_{sense} is 0.210526 V.

Note that the actual measurements will vary based on the actual resistor values in the dummy load, so these values should be considered approximate when running this test.

Setup

The VT1529A/B must be connected to the VT1422A through the VT1539A SCP. The Dummy Load must be connected to the channel(s) being tested.

The Voltage Source must be connected to the appropriate channel bank terminals of the Bridge Excitation input on the VT1529A/B.

The signal path through the VT1529A/B is highlighted in Figure C-6.

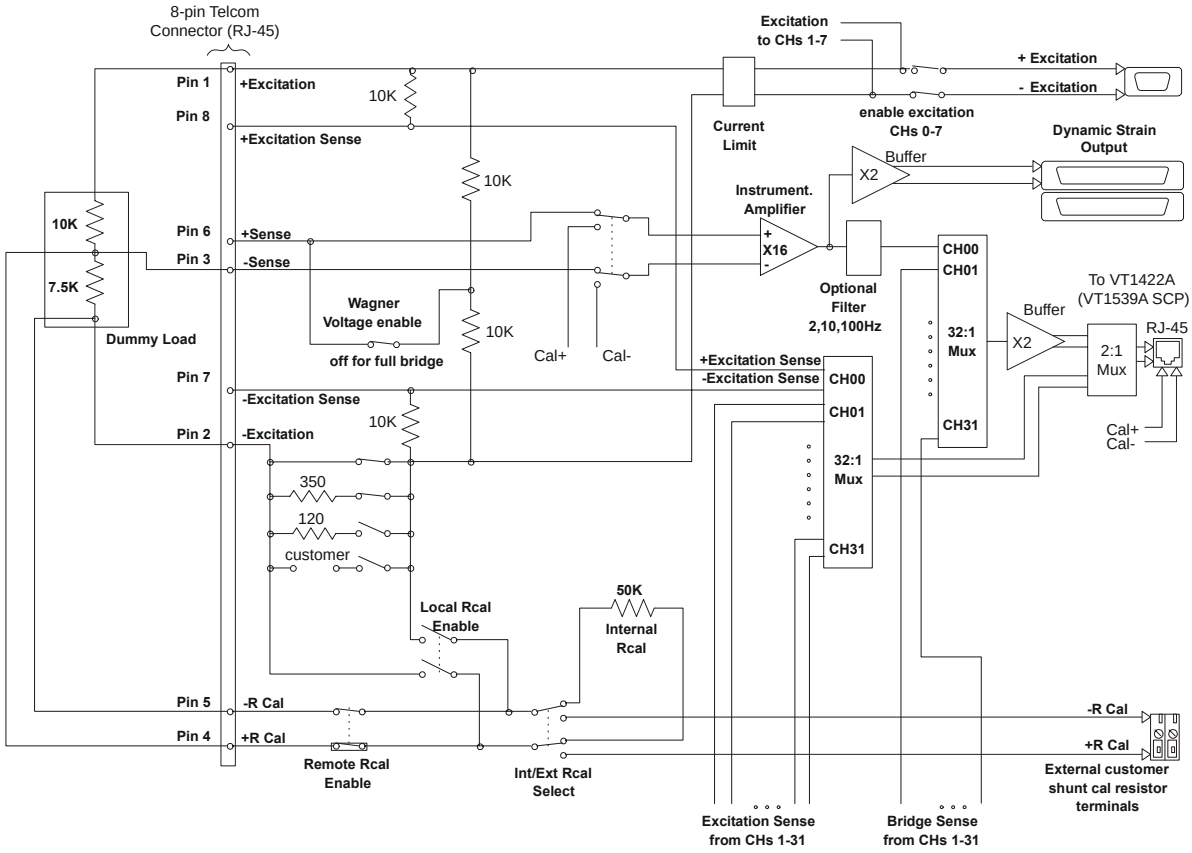


Figure C-6. Signal Path for Internal Shunt Test

Procedure

If necessary, send the *CAL? query to the VT1422A (required if more than 8 hours has elapsed since the last *CAL? command). The expected return value is “+0”.

Set the Voltage Source to 2.000 V.

Configure the VT1529A/B for bending half-bridge operation using the SENS:FUNC:STR:HBEN (@<*ch_list*>) command. Disable shunt calibration by sending the OUTP:SHUNt OFF,(@<*ch_list*>) command. Send the MEAS:VOLT:UNStrained? (@<*ch_list*>) query to the VT1422A. The expected return value is ~0.142 V on each channel. Now, configure for internal shunt using OUTP:SHUNt:SOURce INTernal,(@<*ch_list*>).

For each channel (each must be tested separately), enable internal shunt calibration by sending the OUTP:SHUNt ON,(@<*channel*>) command. Note: this turns off the shunt on the previous channel.
Send the MEAS:VOLT:UNStrained? (@<*channel*>) query to the VT1422A. The expected return value is ~0.211 V on each channel.

Calibration

Calibration is performed via the VT1422A CAL:REMote? command. The VT1422A will automatically calibrate the specified VT1529A/B using the Remote Calibration signal from the VT1422A. This signal is calibrated via the Adjustment Procedure described in Chapter 3 of the VT1415A and VT1419A Service Manual.

No other calibration procedure is needed.

Appendix D
Glossary

The following terms have special meaning when related to the VT1422A.

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 VT1422A that implements a data acquisition and control algorithm.

Algorithm Language — The algorithm programming language specific to the VT1422A. 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 VT1422A, define its algorithms, then start the algorithms running. Typically, once the VT1422A 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 the 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 VT1422A'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 VT1422A 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 VT1422A to store its control firmware and calibration constants
RSC unit or RSCU	This stands for Remote Signal Conditioning Unit. The VT1529A/B 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 VT1422A'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 connected to the system field wiring. The terminal blocks are inside the Terminal Module.

Terminal Module The plastic encased module which contains the terminal blocks connected to the field wiring. The Terminal Module then is plugged into the VT1422A'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 E
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, 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, these types of signals are kept separated by the width of the VT1422A module. The signals are further isolated because they remain separated on the connector module as well. Note that in Figure E-1, even though only seven 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 it is necessary 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.

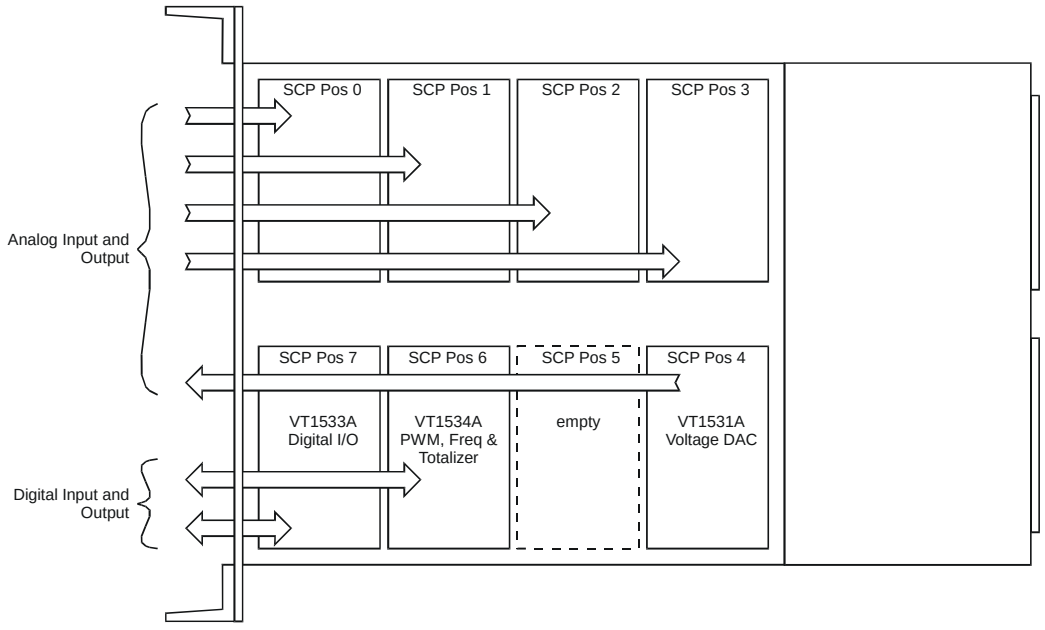


Figure E-1. Separating Analog and Digital Signals

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 VT1422A. Floating DUTs or transducers are an exception. Connect the shield to VT1422A 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 50/60 Hz) 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.

VT1422A Guard Connections

The VT1422A guard connection provides a 10 kΩ current limiting resistor between the guard terminals (G) and VT1422A 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 VT1422A 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

Be very careful not to exceed the maximum common mode voltage referenced to the card chassis ground of ±16 volts (±60 volts with the VT1513A Attenuator SCP). There is an exception to this when high frequency (1 kHz - 20 kHz) common mode noise is present (see "VT1422A Noise Rejection" below). Also, if the DUT is not grounded, then the shield should be connected to the VT1422A 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 the particular 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 VT1422A and Agilent/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.

VT1422A 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 VT1502A, VT1503A, VT1508A, and VT1509A 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 (VT1501A). However, high-frequency Ecm is rectified and generates an offset with the amplifier and filter SCPs (such as VT1502A, VT1503A, VT1508A and VT1509A). 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 VT1422A 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 VT1422A chassis ground through a very low impedance (not via the guard terminal). The VT1422A guard terminal connection shown in the VT1422A 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. It is recommended that this practice be followed and if high frequency common mode noise is seen (or suspected), tie the shield to the VT1422A ground through a 0.1 μF capacitor. At high frequencies, this drives the shield voltage to 0 volts at the VT1422A input. Due to inductive coupling to the signal leads, the Ecm voltage on the signal leads is also driven to zero.

Appendix F
Generating User Defined Functions

Introduction

The VT1422A 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 1.5 μs to 4 μs for each floating point calculation, a complex waveform such as sine(x) could take more than 100 μs to get the desired result. A faster solution is desirable and available.

The VT1422A provides a solution to approximating such complex waveforms by using a piece-wise linearization of virtually any complex waveform. The technique is simple. The VXI*plug&play* Drivers & Product Manuals CD-ROM supplied with the VT1422A contains a 'C' program which calculates 128 Mx+B segments over a specified range of values for the desired function. The user supplies the function; the program generates the segments in a table. The resulting table can be downloaded into the VT1422A's RAM with the ALG:FUNC:DEF command where any desired name of the function (i.e., sin(x), tan(x), etc.) can be selected. 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 18 μs.

The VT1422A actually uses this technique to convert volts to temperature, strain, etc. The accuracy of the approximation is really based upon how well the range is selected over which the table will be built. For thermocouple temperature conversion, the VT1422A fixes the range to the lowest A/D range (±64 mV) so that small microvolt measurements yield the proper resolution of the actual temperature for a non-linear transducer. In addition, the VT1422A permits the creation of Custom Engineering Unit conversion for a 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, it may be necessary to break up the waveform into several functions in order to get the desired accuracy. For example, suppose the generation a square root function is required for both voltage and strain calculations. The voltages are only going to range from 0 to ±16 volts, 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 M*x*+B operations. If accuracy is desired, the range MUST be limited 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 2π 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 -π/2 to π/2). The resulting function represents a fairly accurate approximation of this non-linear waveform when the range is limited as indicated. Since the tables must be built upon binary boundaries (i.e., 0.125, 0.25, 0.5, 1, 2, 4, etc.) and since π/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, the last couple sentences can be ignored if they are not understood. This is only brought up because the accuracy may suffer the farther away from the X=0 point the center is unless the resolution available is understood and how much non-linearity is present in the waveform. This is discussed later in the "Limitations" section.

Figure F-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 -π/2 and the ending point would be at π/2. With the desired range spread over ±π/2, the 128 segments are actually divided over the range of ±2. Therefore, the 128 M*x*+B line segments are divided equally on both sides of X=0: 64 segments for 0..2 and 64 segments for -2..0.

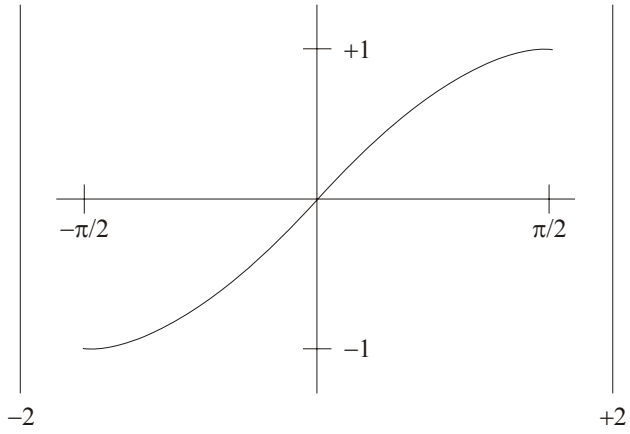


Figure F-1. A Haversine Function

A typical use of this function would be to output an analog voltage or current at each Scan Trigger of the VT1422A and over the range of the haversine. For example, suppose a new position is desired for an analog output and it is moved from 1 mA to 3 mA over a period of 100 ms. If the TRIG:TIMER setting or the EXTernal trigger was set to 2 ms, then forcing fifty intervals over the range of the haversine is desired. 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 -π/2 to π/2 range. 3 mA is multiplied by the custom function result over each interval which will yield the shape of the haversine (0.003*sin(*x*)+0.001).

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. Simply 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 VT1422A 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 F-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 M*x*+B segments are located on boundaries specified by 2/64 on each side of X=0. This means that if the exact input value that was used for the beginning of each segment is selected, the exact calculated value of that function at that point will be determined. 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	'VT1422A' sin(-1.570798)	-0.999905
'C' sin(-1.256639)	-0.951057	'VT1422A' sin(-1.256639)	-0.950965
'C' sin(-0.942479)	-0.809018	'VT1422A' sin(-0.942479)	-0.808944
'C' sin(-0.628319)	-0.587786	'VT1422A' sin(-0.628319)	-0.587740
'C' sin(-0.314160)	-0.309017	'VT1422A' sin(-0.314160)	-0.308998
'C' sin(0.000000)	0.000000	'VT1422A' sin(0.000000)	0.000000
'C' sin(0.314160)	0.309017	'VT1422A' sin(0.314160)	0.308998
'C' sin(0.628319)	0.587786	'VT1422A' sin(0.628319)	0.587740
'C' sin(0.942479)	0.809018	'VT1422A' sin(0.942479)	0.808944
'C' sin(1.256639)	0.951057	'VT1422A' sin(1.256639)	0.950965
'C' sin(1.570798)	1.000000	'VT1422A' sin(1.570798)	0.999905

Table F-1. 'C' Sin(x) Vs. VT1422A 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 the function X*X*X over a range of ±1000 is to represented. 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 M*x*+B calculation over that segment. As long as the numbers input are VERY close to those cardinal points, good results will be attained. Strictly speaking, "perfect" results will only be attained if calculated at the cardinal points, which may be reasonable for an application if the input values are limited to those exact 128 points.

The waveform may also be shifted 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 the results dependent upon the linearity of the waveform used. Suffice it to say that the best results will be attained and it will be easiest to grasp what is being done if the center stays near the X=0 point since most of the results of the measurements will have 1e-6..16 values for volts.

Truncation errors may be seen in the fourth digit of the results. This is because only 15 bits of the 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 F-1 where the first and last entries return ±0.9999 rather than ±1. For most applications, this accuracy should be more than adequate.

Appendix G
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 downloaded as a custom algorithm.

PIDA Algorithm

Figure G-1 shows the block diagram of the PIDA algorithm.

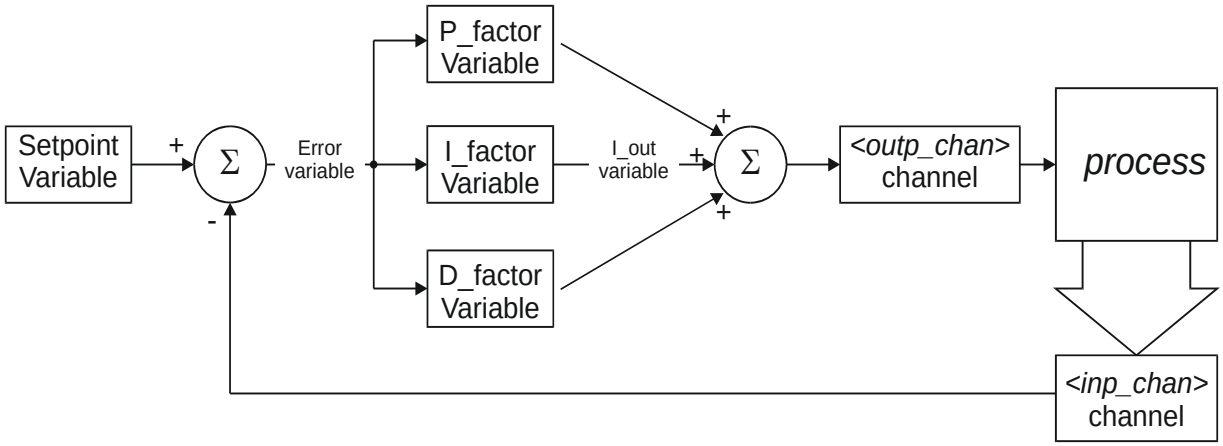


Figure G-1. The Simple PID Algorithm "PIDA"

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 VT1422A 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 G-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.

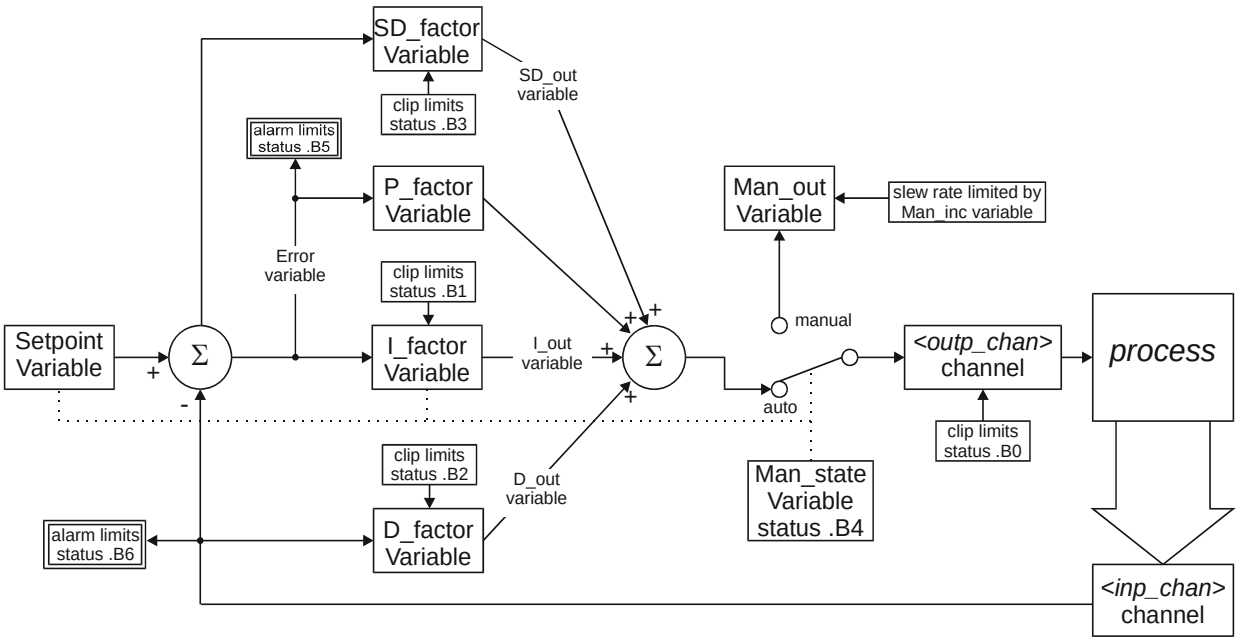


Figure G-2. The Advanced Algorithm "PIDB"

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 the values are changed 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 <*History_mode*> is set 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 five 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 VT1422A 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 output 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 VT1422A 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  );
   }
```

Index

VT1422A Remote Channel Multi-function DAC Module

Symbols

Numerics

A

D

E

F

G

H

I

N

Q

R

S

T

U

V

W

Z

Algorithmic Closed Loop Controller and Remote Channel Multifunction DAC

Overview

The VT1415A and VT1422A are C-size, single-slot, VXI modules capable of either multi-function input/output (data acquisition) or powerful control capabilities. They serve as powerful data acquisition modules that handle analog input/output and digital input/output in both static and dynamic modes. The digital capability includes the ability to set or sense static states, to measure input frequency and period, to totalize, and to input or output PWM and FM signals. Refer to the VXI Technology Website for instrument driver availability and downloading instructions, as well as for recent product updates, if applicable.

Algorithmic Closed Loop Controller - VT1415A

More powerful than PID controllers and easier to implement than large custom control systems, the VT1415A fills a unique niche in the data acquisition and control field, providing both control and precise data acquisition. Applications include:

- PID control of stimulus loops such as hydraulic actuators, levers, rotational devices as in structural test
- PID control of temperature, position, velocity, acceleration and more
- Complex control such as cascade loops in thermal cooling jackets, ratio
- Independent loops with multi-level alarms.

The design of the on-board, DSP firmware assures the user that all inputs, all calculations, and all outputs can be completed between scan triggers. This means there is no drift, or jitter in the critical time intervals that are used to calculate integrals and derivatives in control algorithms.

The firmware allows a user to employ pre-written PID control algorithms, modify them for specific application needs, or to write an application from scratch. Low duty-cycle connection to the host computer allows interaction between the host and real-time DSP so the user can update algorithms, change tuning constants, or do envelope control. Limited host computer interaction leads to very high performance (8-loops, update rate 1000/second per loop with simple PID calculation included).

Multi-function Data Acquisition & Control Module - VT1422A

The VT1422A is a module that is essentially the same as the VT1415A and has all of the same data acquisition and control capabilities as the VT1415A, plus some additional features.

Features

- Powerful Data Acquisition Capability
- Powerful Control Capability
- Comprehensive On-board Signal Conditioning
- Custom On-board DSP Program Development
- Wide Choice of Input/Output Signal Types
- Large Channel-count Strain Signal Conditioning and Measurement

Algorithmic Closed Loop Controller and Remote Channel Multifunction DAC

The VT1422A Remote Channel Multi-Function DAC Module supports the VT1539A Remote Channel Signal Conditioning Plug-on and the VT1529B Remote Strain Signal Conditioning Unit to form a high-performance, but economical strain measurement system.

The VT1422A serves as the controller in this system, managing all the configuration, calibration, triggering of measurements, EU conversion, and calibration processes.

The main differences between the VT1415A and VT1422A are:

- The VT1422A has 40 kB of memory available for user algorithms; the VT1415A has 48 kB.
- If the only thing being done in an application is collection of strain data, the VT1422A user doesn't have to write an algorithm, as for the VT1415A.
- The VT1422A offers the same two Terminal Blocks as does the VT1415A.
  (Option 011 screw terminals and Option 013 spring clamp)

Automated Calibration for Better Measurements

The VT1415A and VT1422A offer superior calibration capabilities that provide more accurate measurements. Periodic calibration of the module's measurement inputs is accomplished by connecting an external voltage measurement standard (such as a highly accurate multimeter) to the inputs of the module. This external standard first calibrates the on-board calibration source. Then built-in calibration routines use the on-board calibration source and on-board switching to calibrate the entire signal path from the closed loop controller's input, through the signal conditioning plug-ons (SCPs) and FET MUX, to the A/D itself. Subsequent daily or short-term calibrations of this same signal path can be quickly and automatically done using the internal calibration source to eliminate errors introduced by the signal path through the SCPs and FET MUX or by ambient temperature changes. All input channels can be quickly and productively calibrated to assure continued high-accuracy measurements.

In addition to the calibration of the signal paths within the modules, the VT1415A and VT1422A allow you to perform a "Tare Cal" to reduce the effects of voltage offsets and IR voltage drops in your signal wiring that is external to the module. The Tare Cal uses an on-board D/A to eliminate these voltage offsets. By placing a short circuit across the signal or transducer being measured, the residual offset can be automatically measured and eliminated by the D/A. Tare Cal should not be used to eliminate the thermoelectric voltage of thermocouple wire on thermocouple channels.

Flexibility with Deterministic Control

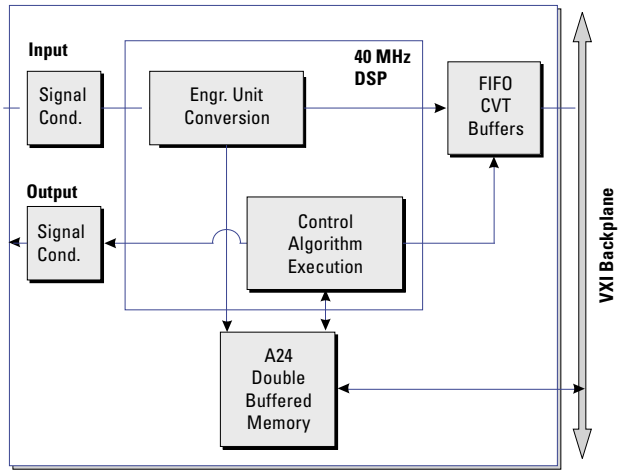
The VT1415A and VT1422A are digital sampling closed loop control systems that are complete in a single VXI module. All signal conditioning, process monitoring, control calculations, and control signals are handled on-board without the need for computer supervision. Once setup is done, the module is essentially free-running.

The inputs are updated at the beginning of each cycle and the outputs are updated at a later deterministic time in the cycle so that various paths in the control algorithm do not affect the loop timing. These steps are executed automatically and deterministically without need for intervention from a system computer.

Other Features

Digital Sampling Closed Loop Control System

The VT1415A/VT1422A combine flexibility with deterministic control. Control algorithms for each of the loops can be the default PID calculation or a user-defined, downloaded, custom algorithm. The loop update rate is deterministically controlled by an internal clock so that variations in the algorithm execution times do not affect the loop cycle time.



Digital Sampling Closed Loop System

Powerful Control Capability

The control algorithm for each loop is easily developed by the user from a list of algebraic expressions and flow constructs such as IF, THEN, ELSE. Tuning is simplified because all of the constants in the algorithm as well as the algorithm itself can be updated on-the-fly. New values are double-buffered so there is no need to stop scanning the inputs or halt the algorithm execution.

Algorithmic Closed Loop Controller and Remote Channel Multifunction DAC

The on-board 40 MHz pipelined DSP provides highly deterministic execution, making it easy to accurately predict cycle times. Engineering unit conversions for temperature, strain, resistance, and voltage measurements are made automatically without slowing down the algorithm execution speed.

Wide Choice of Inputs/outputs

The inputs to the loop algorithm can be measured values from multiple channels, operator input values, outputs from other loops, or values from other subsystems. The VT1415A/VT1422A have a variety of signal conditioning plug-ons for making measurements of:

- Temperature, strain
- Voltage, current, resistance
- RPM, frequency, totalize
- Discrete levels, TTL, contact closures

In addition, the measured input values and the calculated output values can be stored in a 64,000-sample FIFO buffer and efficiently transferred to the controlling computer in blocks of data. With this feature, it is no longer necessary to waste resources by dedicating a data acquisition channel to monitor each control loop input and output. The result of any algorithm calculation can be an input for use by another loop or subsystem, or it can be a direct output of several different types. Among the choices of output are:

- Analog voltage
- Analog current
- Discrete levels (TTL)
- Pulse width modulation (TTL)

As an example of output flexibility, the pulse width modulation output has several modes. In the PWM free-run mode, the frequency or pulse width output rate is independent of the loop update rate and can be changed once per loop update cycle. The square wave mode provides a variable frequency, fixed 50% duty cycle output signal. The pulse-per-update mode provides a variable width pulse synchronized to the loop update cycle.

Operator Control

Manual control can be implemented through a user software interface or external hardware, such as a potentiometer. Seamless transfer from auto to manual mode, or manual to auto is handled automatically by a set-point-tracking routine in the default PID algorithm code.

Signal Conditioning Plug-Ons

A Signal Conditioning Plug-on (SCP) is a small daughter board that mounts on VXI Technology's VXI scanning measurement and control modules. These SCPs provide a number of input and output functions. Several include gain and filtered analog inputs for measuring electrical and sensor-based signals, as well as frequency, total event count, pulse-width modulation, toothed-wheel velocity, and digital state. Output functions include analog voltage and current D/As, 8- or 16-bit digital outputs, pulse output with variable frequency and PWM, and stepper motor control.

Refer to the information on each individual SCP for more details.

Voltage Measurements

Use any of the following SCPs with the VT1415A/VT1422A to make voltage measurements: VT1501A, VT1502A, VT1503A, VT1508A, VT1509A, VT1512A, or VT1513A.

Temperature Measurements

Any of the input SCPs can be used to make temperature measurements with thermocouples, thermistors, or RTDs, but the VT1503A/VT1508A/VT1509A SCPs provide higher accuracy with thermocouples.

Resistance Measurements

Resistance is measured using either the VT1505A 8-channel Current Source SCP and an input SCP or the VT1518A 4-wire Resistance Measurement SCP. Measurements are made by applying a dc current to the unknown and measuring the voltage drop across the unknown.

Static Strain Measurements

There are two ways to make static strain measurements.

The VT1506A and VT1507A SCPs provide a convenient way to measure a few channels of static strain. When using the VT1506A/VT1507A for bridge completion, a second SCP is required to make the measurement connection. You can use the following SCPs for this type of static strain measurements:

- VT1503A 8-channel Programmable Filter/Gain
- VT1506A 8-channel 120 Ω Strain Completion & Excitation
- VT1507A 8-channel 350 Ω Strain Completion & Excitation
- VT1508A 8-channel 7 Hz Fixed Filter & x16 Gain
- VT1509A 8-channel 7 Hz Fixed Filter & x64 Gain

Algorithmic Closed Loop Controller and Remote Channel Multifunction DAC

For applications requiring large channel counts of strain measurement, the EX1629 provides a more cost effective approach to static (and dynamic) strain measurements.

Dynamic strain measurements are implemented by connecting the EX1629 to high-speed digitizers, such as the VXI Technology VT1432B and VT1433B.

Note: SCPs are also available for making dynamic strain measurements (VXI Technology VT1510A, VT1511A, and VT1521).

Transient Measurements

When making higher speed measurements, a vital issue often is the time skew between channels. Ideally, in many applications, the sampled data is needed at essentially the same instant in time. While the intrinsic design of the VT1415A/VT1422A provides scanning of 64 channels, with maximum skew of 640 µs between the first and last channel (far less than most sampled data systems), this still may not be small enough skew for some applications.

Transient Voltage Measurements

The VT1510A provides basic sample-and-hold capabilities on four channels. Six-pole Bessel filters provide alias and alias-based noise reduction while giving excellent transient response without overshoot or ringing. The VT1510A can be used in strain applications primarily where the bridge is external.

Transient Strain Measurements

The VT1511A, a double-wide SCP, has all the capabilities of the VT1510A but adds on-board bridge excitation and completion functions. The four direct input channels are used for monitoring the bridge excitation. A maximum of four SCPs (16 channels) can be installed on a VT1415A/VT1422A.

Analog Output

Use the VT1531A for voltage outputs, and the VT1532A for current outputs. The VT1531A and VT1532A have eight (8) output channels available on each SCP.

A maximum of seven (7) VT1532A SCPs can be installed on each VT1415A/VT1422A due to power limitations. There are no power restrictions on the VT1531A.

Digital I/O

Use the VT1533A Digital I/O SCP to provide two 8-bit input/output words.

Frequency/Totalize/PWM

The VT1538A Enhanced Frequency/Totalize/PWM SCP provides eight (8) channels which can be individually configured as a frequency or totalizer input, or as a pulse width modulated output.

Compact Packaging with Signal Conditioning

The VT1415A/VT1422A provide for configurable signal conditioned I/O with up to eight individual plug-ons for analog, digital, and frequency needs. The SCPs are:

- **VT1501A 8-channel Direct Input SCP**
- **VT1502A 8-channel 7 Hz Low-pass Filter SCP**
- **VT1503A 8-channel Programmable Filter and Gain SCP**
- **VT1505A 8-channel Current Source SCP**
- **VT1506A 8-channel 120 Ω Strain Completion & Excitation SCP**
- **VT1507A 8-channel 350 Ω Strain Completion & Excitation SCP**
- **VT1508A 8-channel x16 Gain & 7 Hz Fixed Filter SCP**
- **VT1509A 8-channel x64 Gain & 7 Hz Fixed Filter SCP**
- **VT1510A 4-channel Sample & Hold Input SCP**
- **VT1511A 4-channel Transient Strain SCP**
- **VT1512A 8-channel 25 Hz Fixed Filter SCP**
- **VT1513A 8-channel Divide-by-16 Fixed Attenuator & 7 Hz Low-pass Filter SCP**
- **VT1518A 4-wire Resistance Measurement SCP**
- **VT1521 4-channel High-speed Bridge SCP**
- **VT1531A 8-channel Voltage Output SCP**
- **VT1532A 8-channel Current Output SCP**
- **VT1533A 16-bit Digital I/O SCP**
- **VT1536A 8-bit Isolated Digital I/O SCP**
- **VT1538A Enhanced Frequency/Totalize/PWM SCP**
- **VT1539A Remote Channel SCP (VT1422A only)**

Product Specifications

VXI Technology™

Algorithmic Closed Loop Controller and Remote Channel Multifunction

Timing Signals

Timing: Scan-to-scan timing and sample-to-sample timing can be set independently.

Scan triggers: Can be derived from a software command or a TTL level from other VXI modules, internal timer, or external hardware. Typical latency 17.5 µs.

Synchronization: Multiple VT1415A/VT1422A modules can be synchronized at the same rate using the TTL trigger output from one VT1415A/VT1422A to trigger the others.

Alternate synchronization: Multiple VT1415A/VT1422A modules can be synchronized at different integer-related rates using the ALG:SCAN:RATIO command and the TTL trigger output from one VT1415A/VT1422A module to trigger the others.

Scan Triggers

Internal: 100 µs to 6.5536 s

Resolution: 100 µs

Trigger count: 1 to 65535 or infinite

Sample Timer Range: VT1415A: 10 µs to 32768 ms
VT1422A: 40 µs to 32768 ms

Resolution: 0.5 µs

Measurement Specifications

The following specifications include the SCP and scanning A/D performance together as a unit. Accuracy is stated for a single sample. Averaging multiple samples will improve accuracy by reducing noise of the signal. The basic VT1415A scanning A/D has a full-scale range of ±16 V and five auto-ranging gains of x1, x4, x16, x64, and x256. An SCP must be used with each eight channel input block to provide input protection and signal conditioning.

Note: For field wiring, the use of shielded twisted pair wiring is highly recommended.

Measurement resolution: 16 bits (including sign)

Maximum reading rate: VT1415A: Up to 56 kSa/s dependent upon configuration
VT1422A: Up to 25 kSa/s dependent upon configuration

Memory: 64 kSamples

Maximum input voltage: Normal mode plus common mode

Operating:	±16 V peak
Damage level:	±42 V peak

Maximum common mode voltage:

Operating:	±16 V peak
Damage level:	±42 V peak

SCP input impedance: 100 MΩ differential

Maximum tare cal offset: 62.5 mV range ±75% of full scale, other ranges ±25% of full-scale

Jitter:

Phase jitter scan-to-scan: 80 ps rms

Phase jitter card-to-card: 41 ns peak 12 ns rms

Measurement Accuracy

Typically ±0.01% of input level; varies with the SCP used. Specifications are 90 days, 23 °C ± 1 °C, with *CAL done after a 1hr warm-up and CAL:ZERO done within 5 minutes. Note: Beyond the 5min. limitation and CAL:ZERO not done, apply the following drift error: Drift = 10 µV/°C ÷ SCP gain, per °C change from CAL:ZERO temperature.

Accuracy Data

Measurement accuracy is dependent upon the SCP module used. Refer to the accuracy tables and graphs for the individual SCP to determine the overall measurement accuracy.

Many definitions of accuracy are possible. Here we use single-shot with 3 sigma noise. To calculate accuracy assuming temperature is held constant within ±1 °C of the temperature at calibration, the following formula applies:

$$\text{Single Shot } 3\sigma = \pm\left(\sqrt{((\text{GainError})^2 + (\text{OffsetError})^2 + (3\sigma \text{ noise})^2}\right)$$

Correcting for Temperature

Algorithmic Closed Loop Controller and Remote Channel Multifunction

To calculate accuracy over temperature range outside the ±1 °C range, results after *CAL are given by replacing each of the above error terms as follows:

Replace
$(GainError)^2$
with
$(GainError)^2 + (GainTempco)^2$

Replace
$(OffsetError)^2$
with
$(OffsetError)^2 + (OffsetTempco)^2$

Loop Control Specifications

Number of loops: 1 to 32

Default control algorithm type: PID

Maximum VT1415A loop update rate for default PID algorithm:

(Note: VT1422A maximum sample rate is 25 kSamples/s, compared to 56 kSa/s for the VT1415A. The loop speeds of the VT1422A are reduced in same ratio.)

1 loop:	3 kHz
8 loops:	1 kHz
32 loops:	250 Hz

Custom algorithm development:

Language:	Subset of C programming language including if-then-else, most math and comparison operations.
Variable types:	Scalar local and global variables, array local and global variables.
Intrinsic functions:	interrupt( ), writefifo( ), writecvt( ),writeboth( ), min( ), max( ), abs( ).
Other functions:	Create own custom functions to handle transcendental operations.

I/O General

A total of eight (8) Signal Conditioning Plug-ons (SCPs) can be installed in most combinations of input or output configurations on a single VT1415A/VT1422A.

Power Available for SCPs

±24 V:	1.0 A
5 V:	3.5 A

General Specifications

VXI device type: A16, slave only, Register based

Size: C

Slots: 1

Connectors: P1/2

Shared memory: n/a

VXI buses: TTL Trigger bus

Drivers: VXI*plug&play* with Source Code

Instrument Drivers - See the VXI Technology Website www.vxitech.com for driver availability and downloading.

VT1415A Algorithmic Closed Loop Controller,

VXI Technology™

Algorithmic Closed Loop Controller and Remote Channel Multifunction

Ordering Information

Includes Spring Clamp Terminal Block	**VT1415A-02** Algorithmic Closed Loop Controller, Includes Screw Connector Terminal Block
VT1415A-A3F	Interface to rackmount terminal panel
VT1422A	Remote Channel Multi-function Data Acquisition & Control Module
VT1422A-001	16-Port RJ-45 Connector Block (supports VT1415A also)
VT1422A-011	Screw Terminal Connector Block (supports VT1415A also)
VT1422A-013	Spring Clamp Terminal Connector (supports VT1415A also)
VT1501A	8-channel Direct Input SCP
VT1502A	8-channel 7 Hz Low-pass Filter SCP
VT1503A	8-channel Programmable Filter/Gain SCP
VT1505A	8-channel Current Source SCP
VT1506A	8-channel 120 Ω Strain Completion & Excitation SCP
VT1507A	8-channel 350 Ω Strain Completion & Excitation SCP
VT1508A	8-channel x16 Gain & 7 Hz Fixed Filter SCP
VT1509A	8-channel x64 Gain & 7 Hz Fixed Filter SCP
VT1510A	4-channel Sample & Hold Input SCP
VT1511A	4-channel Transient Strain SCP
VT1512A	8-channel 25 Hz Fixed Filter SCP
VT1513A	8-channel ÷ 16 Fixed Attenuator & 7 Hz Low-pass Filter SCP
VT1518A	4-wire Resistance Measurement SCP
VT1521	4-channel High-Speed Bridge SCP
VT1531A	8-channel Voltage Output SCP
VT1532A	8-channel Current Output SCP
VT1533A	16-bit Digital I/O SCP
VT1536A	8-bit Isolated Digital I/O SCP
VT1538A	Enhanced Frequency/Totalize/PWM SCP
VT1539A	Remote Channel Signal Conditioning Plug-on (VT1422A only)

ACCESSORIES

73-0025-002 Option 011 Screw Terminal Connector Block

73-0025-003 Option 013 Spring Clamp Terminal Connector Block

73-0025-004 Option A3F Interface to Rackmount Terminal Panel