Contents

Certification

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

Warranty

This Hewlett-Packard product is warranted against defects in materials and workmanship for a period of three years from date of shipment. Duration and conditions of warranty for this product may be superseded when the product is integrated into (becomes a part of) other HP products. During the warranty period, Hewlett-Packard Company will, at its option, either repair or replace products which prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Hewlett-Packard (HP). Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country.

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

Limitation Of Warranty

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

The design and implementation of any circuit on this product is the sole responsibility of the Buyer. HP does not warrant the Buyer's circuitry or malfunctions of HP products that result from the Buyer's circuitry. In addition, HP does not warrant any damage that occurs as a result of the Buyer's circuit or any defects that result from Buyer-supplied products.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED. HP SPECIFICALLY DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Exclusive Remedies

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

Notice

The information contained in this document is subject to change without notice. HEWLETT-PACKARD (HP) MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HP shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. HP assumes no responsibility for the use or reliability of its software on equipment that is not furnished by HP.

Restricted Rights Legend

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

HP E1429A/B 20 MSa/s 2-Channel Digitizer User's Manual
Edition 2

Documentation History

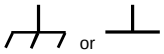
All Editions and Updates of this manual and their creation date are listed below. The first Edition of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct or add additional information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

Edition 1 . . . . . March 1993
Edition 2 . . . . . June 1993

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 frame.

Alternating current (AC).

Direct current (DC).

Indicates hazardous voltages.

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

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

WARNINGS

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

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

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

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

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

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

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

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

Declaration of Conformity

according to ISO/IEC Guide 22 and EN 45014

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

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

declares, that the product:

Product Name: 20MS a/s Digitizer

Model Number: HP E1429A

Product Options: All

conforms to the following Product Specifications:

Safety: IEC 1010-1 (1990) Incl. Amend 1 (1992)/EN61010-1 (1993)

EMC: CISPR 11:1990/EN55011 (1991): Group1 Class A
IEC 801-2:1991/EN50082-1 (1992): 4kVCD, 8kVAD
IEC 801-3:1984/EN50082-1 (1992): 3 V/m
IEC 801-4:1988/EN50082-1 (1992): 1kV Power Line

Supplementary Information: The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC.

Tested in a typical configuration in an HP C-Size VXI mainframe.

March 1, 1993 **Jim White, QA Manager**

European contact: Your local Hewlett-Packard Sales and Service Office or Hewlett-Packard GmbH, Department ZQ/Standards Europe, Herrenberger Straße 130, D-7030 Böblingen, Germany (FAX +49-7031-143143).

Notes

Chapter 1
Getting Started

Chapter Contents

This chapter covers the features, configuration, and programming procedures for the HP E1429A/B 2-Channel, 20 MSa/s Digitizer. The main sections of this chapter include:

HP E1429A/B Features and VXIbus Configuration

The HP E1429A/B is a 2-Channel, 20 MSample/second digitizer. The HP E1429A/B digitizers are VXI message-based instruments, but can also be programmed at the register level (register programming is covered in Appendix C). The features of the HP E1429A and HP E1429B are the same, except that the HP E1429B also has VXI Local bus data transfer capability. This manual covers the use of both digitizers.

Front Panel Description

Figure 1-1 describes the front panels of the HP E1429A/B digitizers.

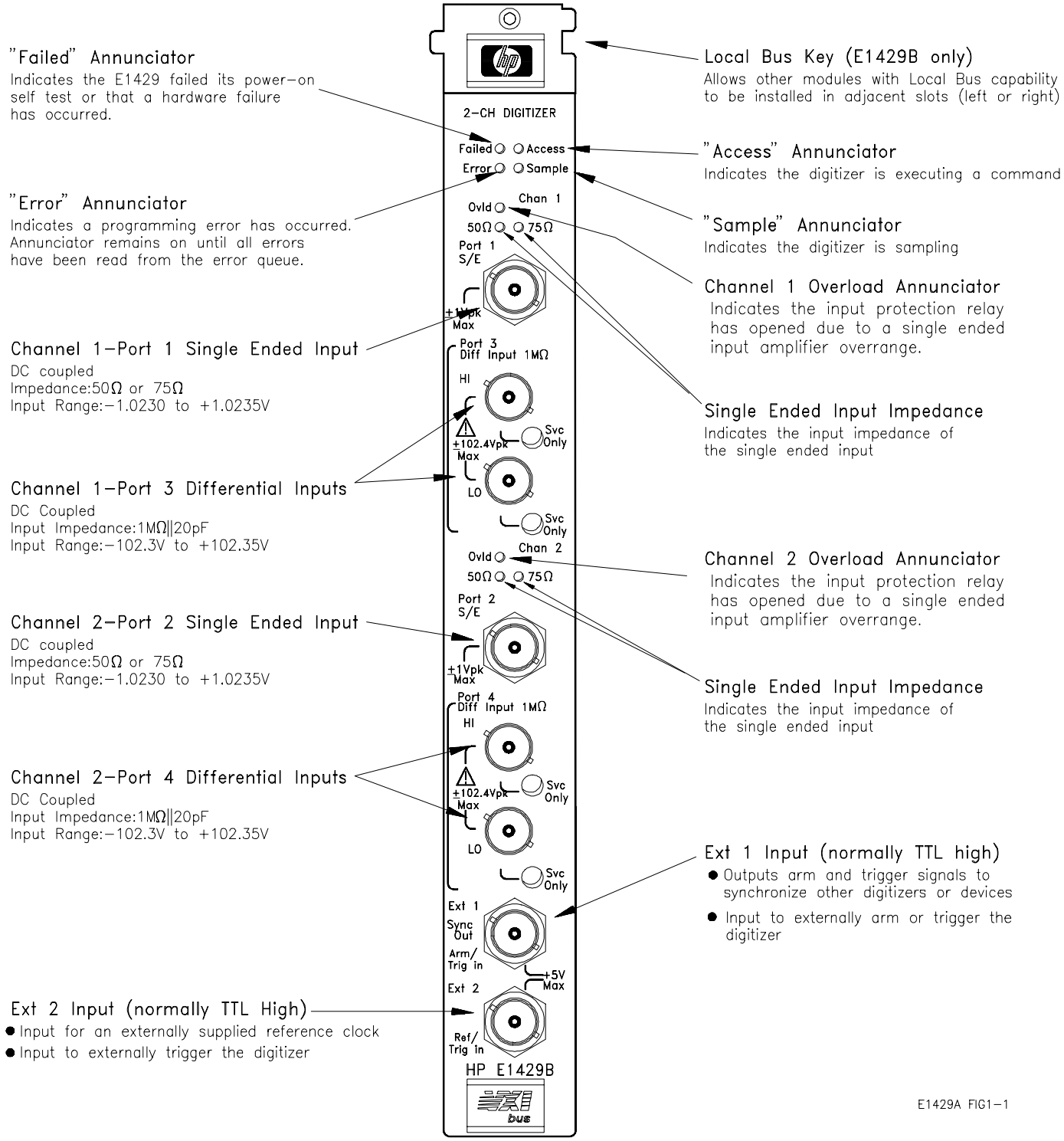


Figure 1-1. The HP E1429A/B Digitizer

HP E1429A/B VXIbus Configuration

Table 1-1 lists the digitizer's VXIbus device information and factory settings. Appendix A has the complete list of HP E1429A/B operating specifications.

Table 1-1. HP E1429A/B VXIbus Configuration

VXIbus Device Information
Device type: message-based servant
C-size (1 slot)
Connectors: P1 and P2
Addressing modes: A16/A24
Data transfer modes: D08/D16/D32 slave
A24 size: 4096 bytes
Dynamically Configurable
Non-interrupter/non-interrupt handler
VXIbus Revision Compliance: 1.4
SCPI Revision: 1992.0
See side of module for power/cooling requirements

HP E1429A/B Factory Settings	
Parameter	Setting
Logical Address	40
Servant Area	0 (not used)
Bus Request Level	3

Input signals (DC and AC) which may be connected to this module are likely to include occasional overvoltage transients. These overvoltages may be caused by motor inductances, switching circuits, lightning, etc.

If the input signal is likely to exhibit transients greater than 800 Vpk, add external transient suppression circuitry to reduce transients to 800 Vpk or less.

Caution **The 800 Vpk level is a product safety test specification and does not assure correct product operation if 800 Vpk transients have been applied. To maintain product functionality and performance, do not exceed ±42 Vpk on the single-ended inputs, or ±102.4 Vpk on the differential inputs.**

Preparation for Use

This section contains configuration information specific to the HP E1429A/B digitizer.

Note For more (VXIbus) system configuration information, refer to the C-Size VXIbus Systems "Installation and Getting Started Guide".

The Digitizer Logical Address

The HP E1429A/B digitizer logical address is used:

- to place the digitizer in the servant area of a commander (e.g. HP E1406 Command Module, embedded controller, or another instrument).
- to address the digitizer (see "Addressing the Digitizer" or " Using an Embedded Controller" later in this chapter.)

Assigning the Digitizer to a Commander

In a VXIbus system, every device must be in the servant area of a commander (with the exception of the top-level comander).

Note the following when assigning the digitizer to a commander:

- A commander's servant area is defined as:

  Servant area = (logical address + 1) through (logical address + servant area switch setting)

- The HP E1429A/B digitizer is a message-based device. If an embedded controller and an HP E1406 Command Module are part of your VXIbus system, put the digitizer in the servant area of the controller. This enables you to program the digitizer at higher speeds across the VXIbus backplane, rather than over the Hewlett-Packard Interface Bus (HP-IB*) via the Command Module.

* HP-IB is Hewlett-Packard's implementation of IEEE Std. 488.1-1978

- If your system uses an external controller and the HP E1406 Command Module, put the digitizer in the servant area of the Command Module. This enables the module to function as the HP-IB interface to the digitizer.

  The HP E1406 Command Module has a factory set logical address of 0 and a servant area switch setting of 255. Using the factory settings, it is not necessary to change the logical address of the digitizer (40) to place it in the servant area of the Command Module.

- If the digitizer is used with the HP E1485 Digital Signal Processing (DSP) module, the digitizer must be in the servant area of the DSP module.

The digitizer's logical address switch is shown in Figure 1-2.

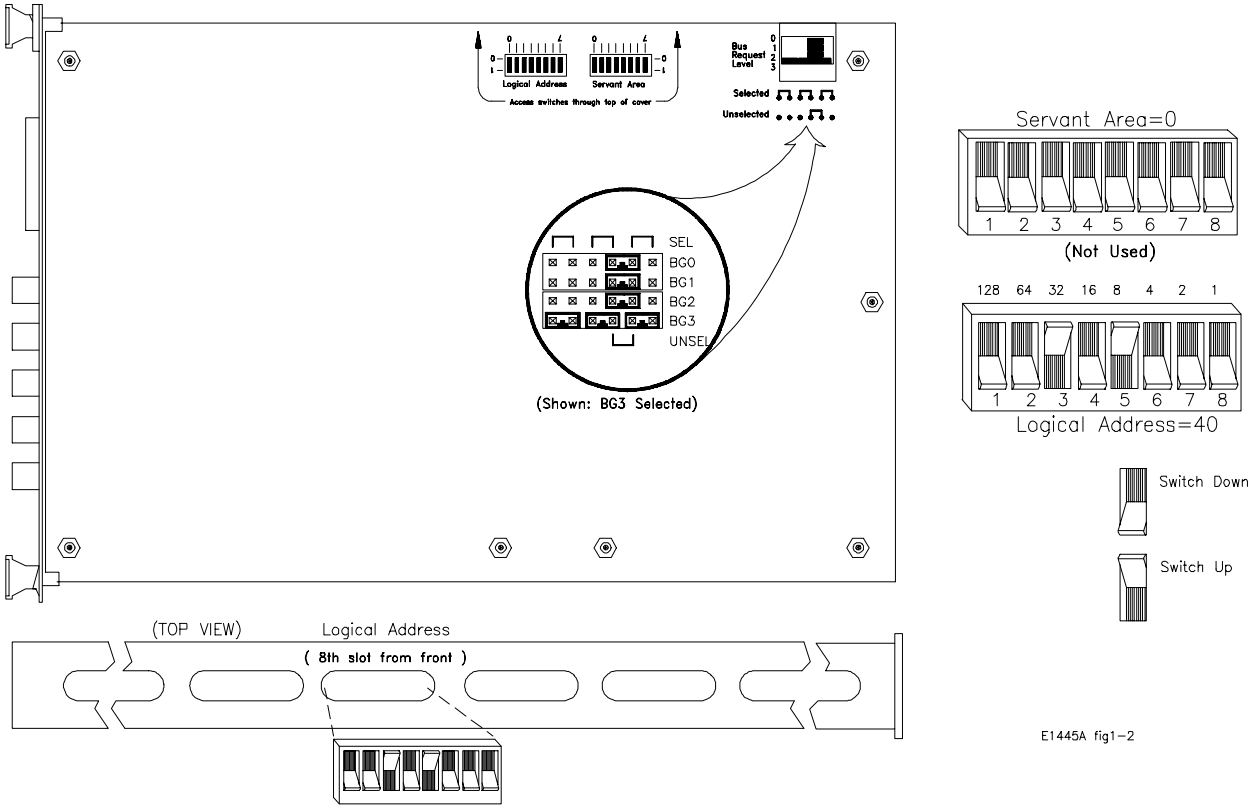


Figure 1-2. HP E1429A/B Logical Address Switch Location

Note The digitizer's servant area switches are not used and should be left in their factory-set (0) position.

The Digitizer Bus Request Level

The bus request level is a priority at which the HP E1429A/B digitizer can request the use of the VME (VXI Data Transfer) Bus.

Bus Request Level Guidelines

- There are four bus request lines (BG0 - BG3) from which one is selected (Figure 1-2). Bus request line 3 has the highest priority, bus request line 0 has the lowest priority. **It is not necessary to change the bus request level setting (BG3) on the digitizer**.
- More information on the Data Transfer Bus can be found in the C-Size VXIbus Systems "Installation and Getting Started Guide".

Installing the Digitizer

The HP E1429A/B digitizer can be installed in any mainframe slot except slot 0. However, in applications where the HP E1429B is generating data over the Local bus, install the digitizer in the left slot adjacent to the module(s) receiving the data. Figure 1-3 shows the installation of a sample Local bus configuration.

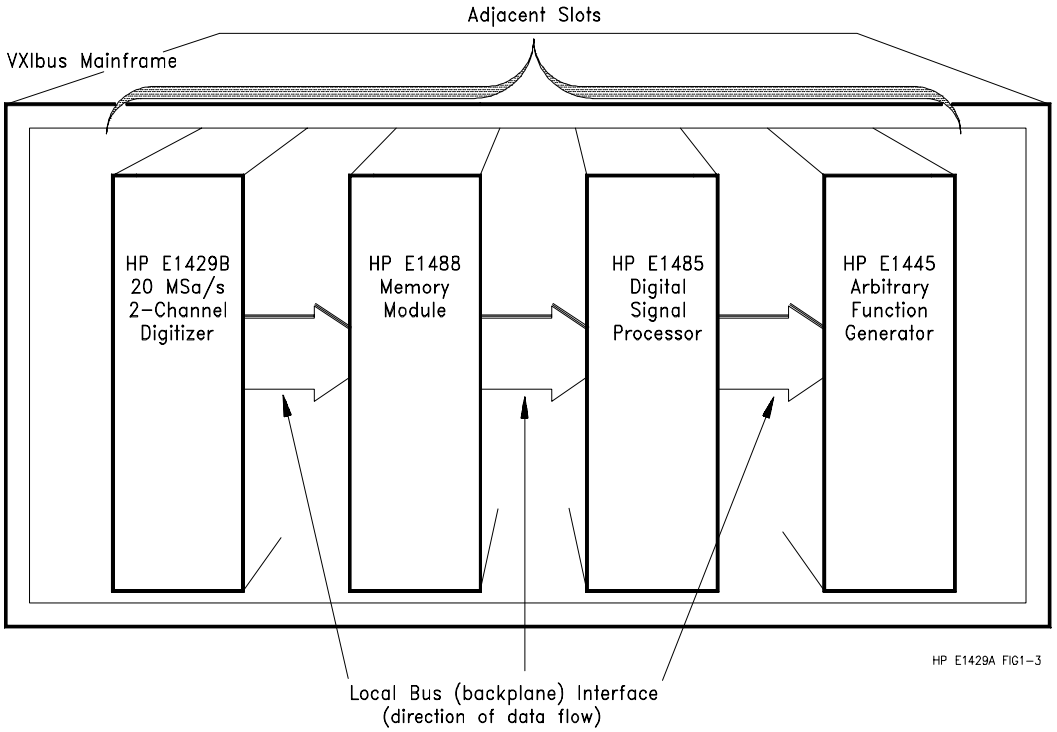


Figure 1-3. Installing the HP E1429B Digitizer in a Local Bus Configuration

Note For compliance with European EMC standards EN 55011 and EN 50082-1, backplane connector shields are included with each HP E1429A/B digitizer ordered. The shields need only be installed in the VXI mainframe if it is necessary to comply with these standards.

Addressing the Digitizer over HP-IB

Devices in the Series C mainframe and in the servant area of the HP E1406 Command Module are located by an HP-IB address. The HP-IB address is a combination of the controller's interface select code, the Command Module's primary HP-IB address, and the device's secondary HP-IB address. An address in this form appears as:

70905

Interface Select Code (**7**): Determined by the address of the HP-IB interface card in the controller. In most Hewlett-Packard controllers, this card has a factory set address of 7, including the HP 82335 HP-IB Interface Card (this card was used with an HP Vectra AT compatible personal computer to create the C programs on the example programs disk).

Primary HP-IB Address (**09**): This is the address of the HP-IB port on the Command Module. Valid addresses are 0 to 30. The Command Module has a factory set address of 9.

Secondary HP-IB Address (**05**): This address is derived from the logical address of the digitizer by dividing the logical address by 8. Thus, for the HP E1429A/B digitizer factory set logical address of 40, the secondary address is 05.

Addressing the Digitizer using an Embedded Controller

As a message-based device, the HP E1429A/B digitizer can easily be programmed across the VXIbus backplane from a HP E1499A V/382 embedded controller. The select code of the VXI interface board in embedded controllers is typically 16. Since no secondary HP-IB address is required when programming over the backplane, the logical address of the HP E1429A/B digitizer is combined with the VXI interface select code:

1640 (for device logical address 40; range = 01 to 99)

If the digitizer's logical address is changed to a value greater than 99, the address becomes:

160xxx

Introductory Programs

The introductory programs in this section include:

- Sending the *IDN? Command
- Digitizer Self-Test
- Resetting the digitizer and clearing the status registers
- Querying the digitizer configuration

HP BASIC and C language versions of the introductory programs follow. C language versions of these and all programs in the manual are contained on the following disk which ships with the manual:

- **HP E1429A/B Example Programs: C Language**
  3.5" 720 KByte disk (HP E1429-10302)

Other than the introductory programs and selected programs throughout the manual, the program listings show only the digitizer commands.

Sending the *IDN? Command

The following programs are a fast method for determining if the digitizer is set to the intended address and is communicating with the computer. The programs send the *IDN? command which returns:

HEWLETT-PACKARD,E1429A,0,A.02.00

HP BASIC

```
10   !Send the *IDN? command, enter and display the result.
20   DIM Message$[80]
30   OUTPUT 70905;"*IDN?"
40   ENTER 70905;Message$
50   PRINT Message$
60   END
```

IDN.C

```
/* IDN.C - This program sends the *IDN? command to the digitizer as a  */
/* way to determine if the computer is communicating with the digitizer */

        /* Include the following header files */

#include   <stdio.h>
#include   <cfunc.h>       /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905    /* I/O path from the PC to the digitizer */
```

Continued on Next Page

```
/*****************************************************************************/
void main(void)                    /* Run the program */
{
        char      message[80];
        intlength = 80;

        IOOUTPUTS(ADDR, "*IDN?", 5);              /* send *IDN? command */
        IOENTERS(ADDR, message, &length);         /* enter *IDN? response */

        printf("%s\n", message);                  /* print*IDN? response */
}
```

Digitizer Self-Test

The digitizer self-test is executed with the command:

*TST?

The digitizer parameters tested include:

- internal interrupt lines
- measurement range integrity
- measurement RAM integrity
- battery charge
- timebase integrity

The self-test takes approximately 30 seconds to complete. Upon completion, one of the self-test codes listed in Table 1-2 is returned.

Table 1-2. HP E1429A/B Self-Test Codes.

Self-Test Code	Description
0	Test passed
1	Test failed. An error message describes the failure.

Caution **Executing the self-test erases the readings in the digitizer's non-volatile memory.**

If the self-test fails, the command:

DIAGnostic:TEST?

can be executed to obtain additional information on the failure. Note that DIAGnostic:TEST can return a string up to 40 characters.

HP BASIC

```
10   !Send the self-test command, enter and display the result.
20   DIM Message$[256],Diagnostic$[40]
30   OUTPUT 70905;"*TST?"
40   ENTER 70905;Rslt
50   IF Rslt <>0 THEN
60      REPEAT
70            OUTPUT 70905;"SYST:ERR?"
80            ENTER 70905;Code,Message$
90            PRINT Code,Message$
100     UNTIL Code =0
110     OUTPUT 70905;"DIAG:TEST?"
120     ENTER 70905;Diagnostic$
130     PRINT Diagnostic$
140  END IF
150  PRINT Rslt
160  END
```

SLFTST.C

```
/* SLFTST.C - This program performs a self-test on the digitizer and prints */
/* out the resulting self-test code */

      /* Include the following header files */
#include   <stdio.h>
#include   <cfunc.h>            /* This file is from the HP-IB Command Library Disk */

#define   ADDR 70905L          /* I/O path from the PC to the digitizer, via the E1406 */
```

Continued on Next Page

```
/*****************************************************************************/
void main(void)                    /* Run the program */
{
        char      message[256], diagnostic[80];
        int length = 256;
        float     tst;

        IOOUTPUTS(ADDR, "*TST?", 5);          /* send the self-test command */
        IOENTER(ADDR, &tst);                  /* enter the code */

        printf("%d\n\n", (int) tst);          /* display the code */

        if (tst != 0)
        {
           IOOUTPUTS(ADDR, "SYST:ERR?", 9);          /* query error register */
           IOENTERS(ADDR, message, &length);         /* enter error message */

           printf("Error: %s\n\n", message);         /* print   error message */

           IOOUTPUTS(ADDR, "DIAG:TEST?", 10);    /* get diagnostic information */
           IOENTERS(ADDR, diagnostic, &length);  /* on self-test error */
           printf("Diagnostic information: %s\n", diagnostic);

        }
}
```

Resetting and Clearing the Digitizer

The commands used to reset and clear the digitizer are:

*RST
*CLS

*OPC? (OPeration Complete) is often executed after *RST and *CLS to allow the reset and clear to complete before program execution continues.

Resetting the digitizer sets it to its power-on configuration, and clearing the digitizer clears its status registers. Additional information on the status registers is located in Chapter 3.

HP BASIC

```
10    !Assign an I/O path between the computer and digitizer.
20    ASSIGN @A_d TO 70905
30    COM @A_d
40    !Call the subprogram
50    Rst_cls
60    END
70    !
80    SUB Rst_cls
90 Rst_cls:   !subprogram which resets and clears the digitizer.
100       COM @A_d
110       OUTPUT @A_d;"*RST;*CLS;*OPC?"  !reset and clear
120       ENTER @A_d;Complete
130   SUBEND
```

RSTCLS.C

```
/* RSTCLS.C - This program resets the digitizer and clears its status register */

/* Include the following header files */
#include    <stdio.h>
#include    <cfunc.h>              /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L             /* I/O path from PC to the digitizer, via the E1406 */

/* Function Prototypes */

void rst_clr(void);

/*****************************************************************************/
void main(void)              /* Run the program */
{
rst_clr();                   /* Reset and clear the digitizer */
}
/*****************************************************************************/
void rst_clr(void)
{
        IOOUTPUTS(ADDR, "*RST;*CLS", 9);       /* reset and clear the digitizer*/
}
```

Querying the Digitizer Configuration

After resetting the digitizer or cycling power, the digitizer parameters are set to their power-on values. These values are listed in Appendix B, Table B-2. You can determine the digitizer's reset settings or its current configuration using the command:

*LRN?

The data returned by *LRN? is a semicolon (;) separated list of each parameter setting.

HP BASIC

```
10    !Assign an I/O path between the computer and the A/D.
20    ASSIGN @A_d TO 70905
30    !Call the subprogram
40    Lrn_conf(@A_d)
50    END
60    !
70    SUB Lrn_conf(@A_d)
80 Lrn_conf:  !subprogram which queries the digitizer configuration
90        DIM Lrn$[2000]
100       INTEGER I
110       OUTPUT @A_d;"*LRN?"
120       ENTER @A_d;Lrn$
130       Lrn$=Lrn$&";"
140       REPEAT
150             I=POS(Lrn$,";")
160             PRINT Lrn$[1;I-1]
170             Lrn$=Lrn$[I+1]
180       UNTIL Lrn$=""
190   SUBEND
```

LRN.C

This program uses a 2,000 element character array. To prevent stack overflow errors when compiling and running this program using Microsoft® QuickC©, change the stack size using the /F option of the "qcl" command. An example of how this program might be compiled is:

qcl /AL /F 8192 b:\lrn.c c:\qc2\lib\clhpib.lib

```
/* LRN.C - This program queries the digitizer's reset conditions */

/* Include the following header files */
#include   <stdio.h>
#include   <string.h>
#include   <cfunc.h>              /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L            /* I/O path from PC to the digitizer, via the E1406 */

/*****************************************************************************/
void main(void)                 /* Run the program */
{
char static *codes[] = {"*RST","*LRN?;*OPC?"};
char  lrndata[2000], *prt, ch;
int     loop,
        length = 2000;

        /* Execute each command group using a loop */

for (loop = 0; loop < (sizeof(codes) / sizeof(char*)); loop++ )
        IOOUTPUTS( ADDR, codes[loop], strlen(codes[loop]));

        /* Enter data returned by *LRN into string */

IOENTERS(ADDR, lrndata, &length);

        /* Start line counter */
loop = 1;

        /* Separate *LRN? data into tokens delimited by ";". Read and */
        /* print the first *LRN? data point */
```

Continued on Next Page

```
prt = strtok(lrndata,";");
printf("\n\t%s",prt);

        /* Print out each (*LRN? data) token */
while (prt != NULL)
{
        prt = strtok(NULL,";");

        /* Exit when data returned by *OPC? (1) is reached */
        if (atoi(prt) == 1)
            break;

        /* Print one user screen's worth of *LRN? data, have user */
        /* press 'Enter' to see the next screen of data */

        if (loop >= 23)
        {
            printf("\n\nPress \'Enter\' to continue");
            scanf("%c", &ch);
            fflush(stdin);
            loop = 0;
        }
        printf("\n\t%s",prt);

loop ++;/* increment counter */
}
}
```

Instrument and Programming Languages

The purpose of this manual is to teach you how to use the HP E1429A/B digitizer. To do this, the manual uses block diagrams, flowcharts, and example programs. In most cases, the manual's example programs list only the digitizer's SCPI commands. The I/O (input/output) constructs depend on the programming language you use.

SCPI Programming

SCPI (Standard Commands for Programmable Instruments) is an ASCII-based instrument command language designed for test and measurement instruments. The message-based digitizer has an on-board microprocessor which interprets the ASCII command strings and returns ASCII formatted results.

Command Listings

The typical format of commands listed in the command reference and throughout this manual is:

```
TRIGger[:STARt]:TIMer1 <period>
```

To aid in learning the digitizer command set, all headers are included in the example programs; however, the headers are abbreviated. In an example program, the previous statement with a *period* parameter of 10 µs would appear as:

```
TRIG:STAR:TIM1 10E-6
```

Note Chapter 4 contains more information on the structure and execution of SCPI commands.

Coupled Commands

Some of the digitizer SCPI commands are functional or value coupled. Functionally coupled commands are those that for one command to have affect, another command must be set to a particular value. Value coupled commands are those where changing the value of one command, changes the value of the others.

Coupled commands can cause “Settings conflict” errors when the program executes. When a coupled command is executed, the command setting is evaluated by the digitizer processor. If the setting causes an illegal digitizer configuration, a "Settings conflict" error occurs. The error message lists the conflicting settings, and then reports the values set by the digitizer processor.

The "Comments" section of each command reference entry (Chapter 4) indicates if a command is coupled, and if it is, what the coupling constraints are.

How to Execute Coupled Commands

To prevent possible "Settings conflict" errors, coupled commands must be contiguous and executed in the same program statement. This is done by placing the commands in the same program line, or for HP BASIC programs, by suppressing the EOL terminator until the last (coupled) command has been sent.

To send multiple commands in a single line or in a single statement, the commands are linked with a semicolon (;) and a colon (:). This is illustrated in the following lines:

```
OUTP:EXT1:STAT ON;:TRIG:SOUR EXT1;:OUTP:EXT1:STAT OFF
```

or

```
OUTP:EXT1:STAT ON;
:TRIG:SOUR EXT1;
:OUTP:EXT1:STAT OFF
```

Notice that the semicolon (;) **and** colon (:) link commands within different subsystems. Only a semicolon (;) is required to link commands at the same level within the same subsystem.

Sending the commands as shown prevents "Settings conflict" errors. The command settings are not evaluated until the EOL terminator is received after the last command. If these commands were sent individually (an EOL terminator after each command), a "Settings conflict" error would occur because of the coupling between OUTP:EXT1:STAT ON and TRIG:SOUR EXT1.

Terminating Commands

A SCPI command string is terminated with a line feed (LF) and/or with an End Or Identify (EOI) message. The carriage return (CR) is ignored.

Suppressing the End-Of-Line Terminator

Suppressing the end-of-line (EOL) terminator on a command line allows coupled commands to be sent on separate lines, yet as a single program statement. In HP BASIC programs, the EOL terminator is suppressed by

placing a semicolon (;) following the quotation mark (") which closes the command string:

```
OUTPUT 70905;"OUTP:EXT1:STAT ON;";
OUTPUT 70905;":TRIG:SOUR EXT1;";
OUTPUT 70905;":OUTP:EXT1:STAT OFF"
```

Since the last command is the end of the command string, the EOL terminator is not suppressed.

Note In the C language programs contained in this manual, there is no end-of-line terminator to suppress as the commands are executed as elements of an array.

C Language Programs

The C language versions of the example programs (disk P/N E1429-10301) were written for the HP 82335 HP-IB Interface Card using the HP-IB Command Library for C. Unless otherwise noted, the library functions used in the programs are compatible with the ANSI C standard.

The following section identifies the system on which the programs were written, shows how to compile and link the programs, and describes the structure of an example program.

System Configuration

The C programs were developed on the following system:

Controller:	HP Vectra 386/25 personal computer (386 processor operated at 25 MHz)
HP-IB Interface Card:	HP 82335 HP-IB Interface with Command Library
Mainframe:	HP 75000 Series C
Slot0/Resource Manager:	HP E1406 Command Module
HP E1429A/B Logical Address:	40
Instrument Language:	SCPI

C Compilers Used

The C Language programs were compiled (and tested) using the following compilers:

- Microsoft® QuickC© Version 2.0
- Borland Turbo C++© Version 1.0

Compiling and Linking the Programs

To run a C program, you must compile and link the program to make an executable file. To compile and link a program:

- Be sure the necessary paths have been added to the AUTOEXEC.BAT file for the compilers to find the library and header files (see the appropriate C Language manual to set the proper paths).
- Link the appropriate HP-IB C library (located on the HP-IB Command Library disk that came with the HP-IB Interface Card). Use the following libraries:
  - **Microsoft® QuickC©:** clhpib.lib
  - **Turbo C++©:** tchhpib.lib
- If NOT compiling in the Large/Huge memory model, include the "cfunc.h" header file (located on the HP-IB Command Library disk) in the program.

Command Line Compiling

To compile and link the programs from the DOS command line using the Large memory model, execute the following from the directory containing "qcl" or "tcc".

- **Microsoft® QuickC©:**

  qcl /AL <*path* \program name > <*path* \clhpib.lib >

  e.g. qcl /AL b:\input.c c:\qc2\lib\clhpib.lib

- **Turbo C++©:**

  tcc -ml <*path* \program name > <*path* \tchhpib.lib >

  e.g. tcc -ml b:\input.c c:\tc\lib\tchhpib.lib

Change the "/AL" and "-ml" parameters to the appropriate types when compiling in the smaller memory models (see your C language manual for the parameter type). For some programs executed under the Microsoft® QuickC© environment, if may be necessary to change the stack size using the /F option of the 'qcl' command.

Once compiled and linked, an executable file (.EXE) and object file (.OBJ) are created in the current directory. You execute the program by typing and entering the file name (with the .EXE extension).

Compiling in the Integrated Environment

You can compile, link, and run your C programs from the Microsoft® QuickC© or Turbo C++© integrated environments. To do so, add:

program_name.C
CLHPIB.LIB

to the program list (under the "Make" menu) in the Microsoft® QuickC© environment. Under 'Environment' in the "Options" menu, include paths to the header files and external CLHPIB.LIB library. For example:

Include Files Directory: [c:\qc2\include]

Library Files Directory: [c:\qc2\lib]

In the Turbo C++© environment, add the items:

program_name.C
TCHHPIB.LIB

to the project (under the "Project" menu). Under 'Directories ...' in the "Options" menu, include paths to the header files and external TCHHPIB.LIB library. For example:

Include Directories
C:\TC\INCLUDE

Library Directories
C:\TC\LIB

C Program Format

The general format of the C language programs on the example programs disk is shown in the program listings at the end of this chapter. Generally, the program flow is:

- reset and clear the digitizer
- configure the digitizer
- check for configuration errors
- trigger the digitizer and retrieve the readings

Introduction to Programming

The SCPI commands used to program the digitizer are separated into two groups: common commands and subsystem commands. Common commands begin with an asterisk, and include commands such as *RST, *CLS, *OPC?. Chapter 4 contains a complete listing of the digitizer's common commands.

Subsystem commands are those commands which configure the digitizer. Each subsystem is a set of commands that roughly corresponds to a functional block inside the digitizer. Figure 1-4 identifies the SCPI subsystems used with the digitizer.

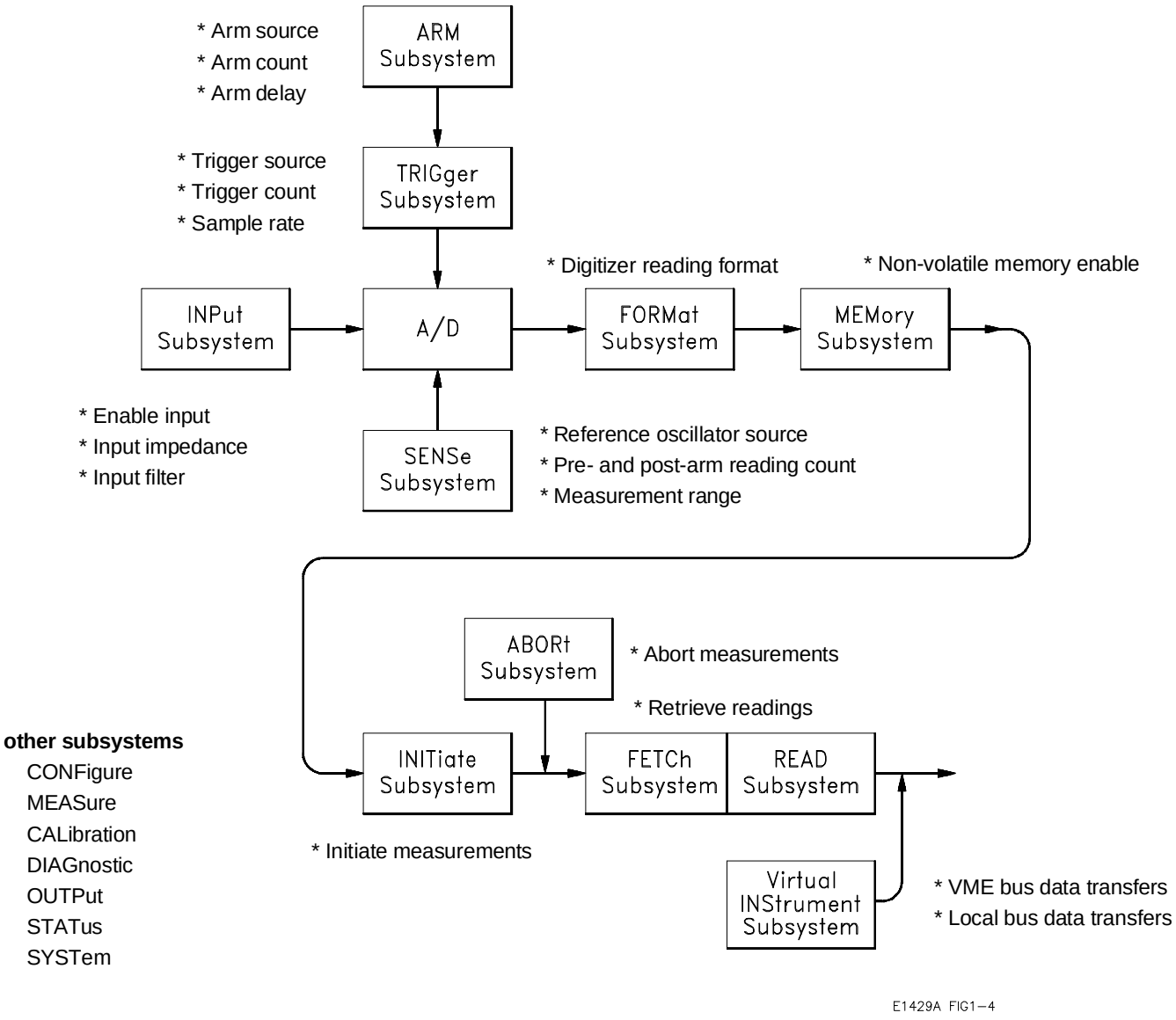


Figure 1-4. HP E1429A/B Digitizer Command Subsystems

Using the MEASure and CONFigure Commands

Each time the digitizer takes a reading, it does so from a configuration based on parameters set by the digitizer subsystems (Figure 1-4). The easiest way to set these parameters is with the MEASure or CONFigure command.

MEASure[<*chan* >]:ARRay[:VOLTage][:DC]? (<*size* >)
[,<*expected value* >[,<*resolution* >]] [,(@<*input port* >)]

CONFigure[<*chan* >]:ARRay[:VOLTage][:DC] (<*size* >)
[,<*expected value* >[,<*resolution* >]] [,(@<*input port* >)]

< ***chan*** > is the digitizer channel (1 or 2) configured. This parameter is optional. If a channel is not specified, channel 1 is assumed.

(< ***size*** >) is the total number of pre-arm and post-arm readings (samples) taken each time an arm signal occurs. Note the space between the command header and the <*size* > parameter.

<***expected value*** > is the amplitude (range) of the signal to be measured. This optional parameter is used to set the digitizer measurement range. If an expected value is not specified, the digitizer defaults to the 1V range.

<***resolution*** > is the reading resolution and is determined from the *expected value*. There is a fixed resolution for each measurement range (Table 3-2). If a specified resolution is greater that what is available for that range (expected value), an error occurs.

(***@***<***input port***>) is the channel input port (single ended or differential) to which the input signal is applied. Readings can be taken on only one input port per channel at a time.

Table 1-3 lists some of the commands and their settings that are equivalent to the values set by MEASure and CONFigure.

When MEASure? or CONFigure is executed, many of the digitizer parameters are set to their reset values (see Appendix B, Table B-2 for a complete listing of reset values). The parameters specified within the MEASure or CONFigure command are then set accordingly. This prevents "Settings conflict" errors from occurring due to previous digitizer configurations.

Table 1-3. Digitizer Configuration using MEASure? and CONFigure

Parameter	Command	Setting
Reference Oscillator Source	SENSe<*chan*>:ROSCillator:SOURce <*source*>	INTernal (the digitizer's internal 20 MHz oscillator)
Input Port	SENSe<*chan*>:FUNCtion "<*function*> <*port*>"	"VOLT <port>" (where <*port*> is set by the (@<*input port*>) parameter of MEASure? or CONFigure)
Measurement range	SENSe<*chan*>:VOLTage:DC:RANGe <*range*>	set according to the <*expected value*> parameter of MEASure? or CONFigure
Input Impedance	INPut<*port*>:IMPedance <*impedance*>	50Ω (when <*port*> is 1 or 2)
10 MHz Input Filter	INPut<*port*>:FILTer:LPASs:STATe <*mode*>	ON
Input State	INPut<*port*>:STATe <*mode*>	ON (for all ports)
Arm Source	ARM:STARt:SOURce<*n*> <*source*>	IMMediate (for *n* = 1) HOLD (for *n* = 2)
Arm Count	ARM:STARt:COUNt	1
Arm Delay	ARM:STARt:DELay	0
Trigger Source	TRIGger:STARt:SOURce <*source*>	TIMer1
Pre-arm Readings	SENSe<*chan*>:SWEep:OFFSet:POINts <*count*>	0
Trigger Count	TRIGger:STARt:COUNt <*number*> SENSe<*chan*>:SWEep:POINts <*number*>	<*number*> is set to the (<*size*>) parameter of MEASure? or CONFigure
Sample Rate (single)	TRIGger:STARt:TIMer1 <*period*>	50 ns
Sample Rate (dual)	TRIGger:STARt:TIMer2 <*period*>	100 ns
Output State	OUTPut:ECLTrg<*n*>:STATe <*mode*> OUTPut:TTLTrg<*n*>:STATe <*mode*> OUTPut:EXTernal1:STATe <*mode*>	OFF OFF OFF
Output Feed	OUTPut:ECLTrg<*n*>:FEED <*source*> OUTPut:TTLTrg<*n*>:FEED <*source*> OUTPut:EXTernal1:FEED <*source*>	"TRIGger:STARt" (ECLTrg0) "EXTernal1" (ECLTrg1) "ARM:STARt" "TRIGger:STARt"
VME Bus Mode	VINStrument:CONFigure:VME:MODE <*mode*>	OFF
VME Bus Feed	VINStrument:CONFigure:VME:FEED <source >	"MEMory:BOTH32"
Local Bus Mode (HP E1429B only)	VINStrument:CONFigure:LBUS:MODE <*mode*>	OFF
Local Bus Feed (HP E1429B only)	VINStrument:CONFigure:LBUS:FEED <*source*>	"MEMory:BOTH"
Reading Format	FORMat[:DATA] <*type*>[,<*length*>]	ASCii, 9
Data label	DIAGnostic:CHANnel[<chan >]:LABel <*label* >	0 (channel 1) 0 (channel 2)

Programming Sequence

The recommended sequence for programming the digitizer is shown in Figure 1-5. Note that CONFigure sets many digitizer parameter values that usually do not have to be changed with "lower-level" subsystem commands. The lower-level commands are used when you want to set a value different from the value set by CONFigure.

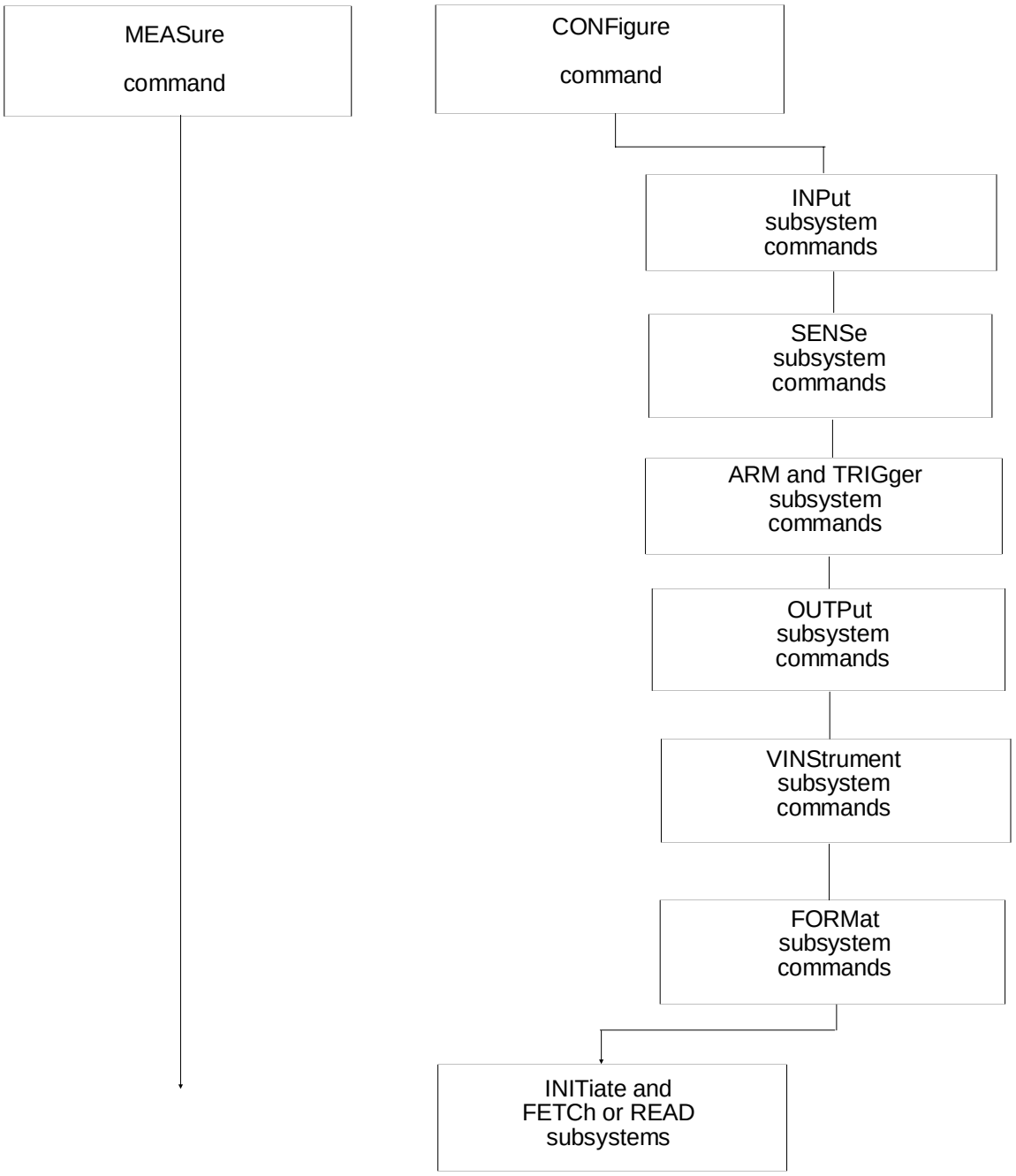


Figure 1-5. HP E1429A Programming Sequence

Configuring the Channels

Digitizer channels 1 and 2 share the digitizer's arming and triggering circuitry and memory. Thus, the arming and triggering configuration and the number of readings set for one channel applies to the other channel as well. When the digitizer is armed and triggered, both channels sample and store their readings in memory simultaneously.

How to Make Measurements

This section explains when to use MEASure or CONFigure to configure the digitizer. It also shows you how to make measurements when the configuration has been modified with lower-level commands.

Using MEASure

MEASure is used in applications where the digitizer parameters set by the command are acceptable, and the data is to be retrieved immediately after the readings are taken. MEASure is equivalent to executing the command sequence:

```
ABORt;:CONFigure;:INITiate:IMMediate;:FETCh?
```

The following programs execute the MEASure command as shown below:

```
MEAS1:ARR:VOLT? (10),5,(@3)
```

MEASure configures the digitizer as follows:

channel:	1
number of readings:	10
expected value:	5V
input port:	3

The remainder of the digitizer parameters are set as indicated in Table 1-3. Because the readings are taken immediately, variations to the digitizer configuration are limited to the parameters within the MEASure command (number of readings, expected value, resolution, input port).

HP BASIC

```
10   DIM Readings(1:10)
20   OUTPUT 70905;"*RST;*CLS;*OPC?"
30   ENTER 70905;Ready
40   OUTPUT 70905;"MEAS1:ARR:VOLT? (10),5,(@3)"
50   ENTER 70905;Readings(*)
60   PRINT Readings(*)
70   END
```

MEAS.C

```
/* MEAS.C - This program demonstrates how to take readings using the */
/* digitizer's MEASure command. In this program, MEASure configures the */
/* digitizer to take 10 readings on the 5V range, using the differential */
/* input port. */

        /* Include the following header files */

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <malloc.h>
#include    <cfunc.h>           /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L          /* I/O path from PC to the digitizer, via the E1406 */

/* Function Prototypes */

void rst_clr(void);
void ad_meas(void);
void check_error(char *func_tion);

/*****************************************************************************/
void main(void)             /* run the program */
{
        rst_clr();          /* reset the digitizer */
        ad_meas();          /* function which configures the digitizer and makes */
                            /* the measurement */

}

/*****************************************************************************/
void ad_meas(void)
{
        int    i = 0, readings = 10;
        float *rdgs;

        /* dynamically allocate memory for readings */

        rdgs = malloc(10 * sizeof(float));
```

Continued on Next Page

```
        /* Use the MEASure command to configure the digitizer and to take */
        /* the readings */

        IOOUTPUTS(ADDR, "MEAS1:ARR:VOLT? (10),5,(@3)", 27);
                                /* Send (27) characters */

        IOENTERA(ADDR, rdgs, &readings);
           /* Read readings from MEASure command */

        for (i = 0; i < readings; i++)
           {
                printf("\nReading %d = %f", i, *rdgs++);
           }
        free (rdgs - readings);
}

/*****************************************************************************/
void rst_clr(void)
{
           /* Reset and clear the A/D */

        IOOUTPUTS(ADDR, "*RST;*CLS", 9);        /* Send (9) characters */
}
```

Using CONFigure

When an application requires a configuration different from that available with MEASure, CONFigure is used. CONFigure *does not* take readings after setting the configuration. Thus, any of the low-level commands (Table 1-3) can be used to change selected parameters before a measurement is made.

Assume an application requires the following configuration:

- 10 pre-arm and 10 post arm readings
- 1V range
- single ended input

MEASure cannot be used since it sets the pre-arm reading count to 0. By using CONFigure, the low-level command:

SENSe[<*chan* >]:SWEep:OFFSet:POINts <*count* >

can be used to set the desired number of pre-arm readings:

CONF1:ARR:VOLT (20),1,(@1)

SENS1:SWE:OFFS:POIN -10

Taking Readings After Using CONFigure

To take readings, the digitizer must be triggered. The MEASure command automatically triggers the digitizer after setting the configuration.

When CONFigure is used, the digitizer must be triggered using the READ? command or INITiate[:IMMediate] and
FETCh[<chan >]? commands as shown.

CONF1:ARR:VOLT (20),1,(@1)

SENS1:SWE:OFFS:POIN -10

READ? (readings are sent to the digitizer output buffer from memory)

or

CONF1:ARR:VOLT (20),1,(@1)

SENS1:SWE:OFFS:POIN -10

INIT:IMM (readings are stored in memory)

FETC?(readings are retrieved from memory)

READ? is equivalent to executing ABORt +INITiate:IMMediate +FETCh?. ABORt stops any measurement or VME or Local bus transfer before proceeding with INITiate. With READ?, the readings pass directly through digitizer memory to the device's output buffer.

INITiate places the digitizer in the wait-for-arm state. FETC? waits for the readings to complete and then retrieves (fetches) the readings from memory and places them into the output buffer.

HP BASIC

```
10   DIM Readings(1:20)
20   OUTPUT 70905;"*RST;*CLS;*OPC?"
30   ENTER 70905;Ready
40   OUTPUT 70905;"CONF1:ARR:VOLT (20),1,(@1)"
50   OUTPUT 70905;"  SENS1:SWE:OFFS:POIN -10"
60   OUTPUT 70905;"READ?"
70   ENTER 70905;Readings(*)
80   PRINT Readings(*)
90   END
```

CONF.C

```
/* CONF.C - This program demonstrates how to use the CONFigure command and */
/* low-level digitizer commands to configure the digitizer. INITialize and */
/* FETCh? are used to trigger the digitizer and retrieve the readings from */
/* digitizer memory. */

        /* Include the following header files */

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <malloc.h>
#include    <cfunc.h>              /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L             /* I/O path from the PC to the digitizer */

/* Function Prototypes */

void rst_clr(void);
void ad_conf(void);
void ad_fetch(void);
void check_error(char *func_tion);

/*****************************************************************************/
void main(void)              /* run the program */
{
        rst_clr();           /* reset the digitizer */
        ad_conf();           /* send commands which configure the digitizer */
        ad_fetch();          /* send command which retrieves the digitizer readings */
}

/*****************************************************************************/
void ad_conf(void)
{
        int    length = 0, loop = 0;

        /* use the "set_commands" array to configure digitizer channel 1 */

        char static *set_commands[] =
        {"CONF1:ARR:VOLT (20),1,(@1)",     /* set 20 readings, 1V range, */
                                           /* S/E input port 1 */
           "SENS1:SWE:OFFS:POIN -10",          /* set 10 pre-arm readings */
         "INIT"};                          /* place digitizer in */
                                           /* wait-for-arm state */
```

Continued on Next Page

```
        length = (sizeof(set_commands) / sizeof(char*));

       /* Execute each command using a loop */

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR, set_commands[loop], strlen(set_commands[loop]));
           }

        /* function call to check for digitizer configuration errors */

        check_error("ad_conf");
}

/*****************************************************************************/
void ad_fetch(void)
{
        char go;
        int    i = 0, readings = 20;
        float *rdgs;

        /* dynamically allocate memory for readings */

        rdgs = malloc(20 * sizeof(float));

        /* fetch (retrieve) and print    readings */

        IOOUTPUTS(ADDR, "FETC1?", 6);

        IOENTERA(ADDR, rdgs, &readings);

           for (i = 0; i < readings; i++)
           {
                   printf("\nReading %d = %f", i, *rdgs++);
           }

        free(rdgs - readings);
 }

/*****************************************************************************/
void rst_clr(void)
{
           /* Reset and clear the digitizer */
```

Continued on Next Page

```
        IOOUTPUTS(ADDR, "*RST;*CLS", 9);
}

/*****************************************************************************/
void check_error(char *func_tion)
{
        char      into[161];
        int length = 160;

        IOOUTPUTS(ADDR, "SYST:ERR?", 9);          /* Query error register */
        IOENTERS(ADDR, into, &length);            /* Enter error message */

        if (atoi(into) != 0)                      /* Determine if error is present */
                                                  /* If errors present, print    and exit */
        {
           while (atoi(into) != 0)
           {
                printf("Error %s in function %s\n\n", into, func_tion);
                IOOUTPUTS(ADDR, "SYST:ERR?", 9);
                IOENTERS(ADDR, into, &length);
           }

           exit(1);
        }
}
```

Querying Command Settings

As you configure the digitizer it is often useful to determine command settings programmatically. This can be done by adding a question mark (?) at the end of any SCPI command header (except the MEASure command), and then sending the command without parameters. The following programs query the parameters of the CONFigure command. Assuming the CONFigure command was executed as indicated in program CONF.C, the following query response is returned:

"ARR (20),1.000000,0.000500,(@1)"

where:

(20)	= reading count (*size* parameter)
1.000000	= expected value
0.000500	= reading resolution
(@1)	= input port

HP BASIC

```
10    DIM Setting$[1000]
20    OUTPUT 70905;"CONF?"      !query CONFigure command
30    ENTER 70905;Setting$
40    PRINT Setting$
50    END
```

QUERY.C

```
/* This program queries HP E1429 settings */

/* Include the necessary header files */

#include   <stdio.h>
#include   <string.h>
#include   <cfunc.h>              /* from HP-IB command library */

/* Define E1429 HP-IB address macro */

#define    ADDR 70905L            /* I/O path between the PC and the digitizer */

/* Function prototypes */

void query(void);

void main(void)
{
       query();                   /* function call to query HP E1429 parameters */
}

/*****************************************************************************/
void query(void)
{
       char *gets();
       char qry_cmd[80];          /* query command array */
       char qry_resp[80];         /* query response array */
       int    length = 80;

       /* Query user for digitizer query command */

       printf("\nEnter query command: ");

       gets(qry_cmd);       /* get input string (query command) */
```

Continued on Next Page

```
        IOOUTPUTS(ADDR, qry_cmd, strlen(qry_cmd));

        IOENTERS(ADDR, qry_resp, &length);

        printf("\n%s = %s", qry_cmd, qry_resp);
}
```

Checking for Errors

The following HP BASIC program shows the lines and subprogram which can be added to HP BASIC programs to check for errors. Line 140 clears the digitizer standard event status register. Lines 150 and 160 unmask the appropriate bits in the digitizer's status byte register and standard event status register.

When an error occurs, the subprogram "Errmsg" reads the digitizer error queue and displays the code and message. Note that line 310 is used as an "end of statement" should a syntax error occur among coupled commands. Otherwise, line 320 would serve as the end of statement and the ABORT command would be ignored by the digitizer parser.

Note An alternative HP BASIC error checking program can be found in the "C-Size VXIbus Systems Installation and Getting Started Guide".

HP BASIC

```
1    !This program represents one method that can be used to
2    !check for programming errors in HP BASIC programs.
3    !
10   !Assign I/O path between the computer and HP E1429A/B.
20   ASSIGN @A_d TO 70905
30   COM @A_d
40   !Define branch to be taken when an HP E1429A/B error occurs.
50   !Enable HP-IB interface to generate an interrupt when an error
60   !occurs.
70   ON INTR 7 CALL Errmsg
80   ENABLE INTR 7;2
90   !Clear all bits in the standard event status register, unmask the
100  !standard event status group summary bit in the HP E1429A/B status byte
110  !register (decimal weight 32), unmask the query error, device
120  !dependent error, execution error, and command error bits
130  !(decimal sum 60) in the HP E1429A/B standard event status register.
```

Continued on Next Page

```
140  OUTPUT @A_d;"*CLS"
150  OUTPUT @A_d;"*SRE 32"
160  OUTPUT @A_d;"*ESE 60"
170  !
180  !Subprogram calls would be here
190  !
200  WAIT .1  !allow error branch to occur before turning intr off
210  OFF INTR 7
220  END
230  !
240  SUB Errmsg
250 Errmsg:  !Subprogram which displays HP E1429 programming errors
260      COM @A_d
270      DIM Message$[256]
280      !Read digitizer status byte register and clear service request bit
290      B =SPOLL(@A_d)
300      !End of statement if error occurs among coupled commands
310      OUTPUT @A_d;""
320      OUTPUT @A_d;"ABORT"  !abort digitizer activity
330      REPEAT
340              OUTPUT @A_d;"SYST:ERR?"  !read digitizer error queue
350              ENTER @A_d;Code,Message$
360              PRINT Code,Message$
370      UNTIL Code=0
380      STOP
390  SUBEND
```

ERRORCHK.C

```
/* ERRORCHK.C - This program contains the C function used by the example */
/* programs to check for digitizer configuration errors. When an error */
/* occurs, the function reads the digitizer's error buffer and prints the */
/* error messages until all of the errors have been read. */

/* Include the following header files */

#include    <stdio.h>
#include    <cfunc.h>      /* This file is from the HP-IB Command Library Disk */

#define ADDR 70905L      /* I/O path from PC to the digitizer, via the HP E1406 */

/* Function Prototype */

void check_error(void);
```

Continued on Next Page

```
/*****************************************************************************/
void main(void)
{
check_error();        /* call error check function */
}

/*****************************************************************************/
void check_error(void)
{
char    into[161];
int       length = 160;

IOOUTPUTS(ADDR, "SYST:ERR?", 9);       /* Query error register */
IOENTERS(ADDR, into, &length);              /* Enter error message */

if (atoi(into) != 0)           /* Determine if error is present */
                                /* If errors present, print    and exit */
{
        while (atoi(into) != 0)
        {
           printf("%s\n\n", into);
           IOOUTPUTS(ADDR, "SYST:ERR?", 9);
           IOENTERS(ADDR, into, &length);
        }

        exit(1);
}
}
```

Digitizer/ Command Module Deadlock

If the digitizer's access light remains on while programming the digitizer over HP-IB, it may be due to a deadlock between the digitizer and the HP E1406 Command Module. If such a deadlock occurs and you cannot access the Command Module, send the following command sequence:

```
ABORT 7
ABORT 7
CLEAR 70905  (selected device clear)
```

Where to go Next

- For additional progamming examples:

  Chapter 2: Using the Digitizer

- For the digitizer description of operation:

  Chapter 3: Understanding the Digitizer

- For information on the digitizer command set:

  Chapter 4: Command Reference

- For the digitizer operating specifications:

  Appendix A: Specifications

- For a listing of the digitizer error messages:

  Appendix B: Useful Tables

- For register programming information:

  Appendix C: Register Programming

Chapter 2
Using the Digitizer

Chapter Contents

This chapter contains example programs that show you how to use the digitizer. The programs, which demonstrate the various features of the digitizer, are presented in the same sequence as the features are covered in Chapter 3 - "Understanding the Digitizer". The examples in this chapter include:

Using the Programs

Each example program listed in this chapter (and on the example programs disk) begins with the CONFigure command, and then uses lower-level digitizer commands (Chapter 1, Table 1-3) to customize the configuration. Using this format, the programs can easily be modified to match your application.

The Programming Language

Each example in this chapter lists only the digitizer's SCPI commands, with the exception of the VME and Local bus data transfer programs. The I/O (input/output) constructs of the programming language you use need to be added to the programs. Note that the example programs disk (HP P/N E1429-10302) contains copies of the programs compiled and tested using Borland Turbo C++ © Version 1.0 and Microsoft® QuickC© Version 2.0. The program name is shown prior to the program listing in the chapter.

Configuring the Digitizer Input

This program demonstrates the commands used to configure the digitizer's input section. The program sets up the digitizer to take 10 readings on the 1V range of the digitizer's single ended input port. This includes:

- enabling/disabling the input ports
- setting the input impedance
- switching the 10 MHz filter into the signal path.
- setting the signal range

INPUT.C

```
*RST;*CLS                       /* reset and clear the digitizer */
CONF1:ARR:VOLT (10),(@1)        /* set 10 readings on channel 1, input port 1 */
    INP3:STAT OFF                   /* disable differential port 3 */
    INP1:IMP 50                 /* set input impedance to 50 ohms */
    INP1:FILT ON                /* switch 10 MHz filter into signal path */
    SENS1:VOLT:RANG 1           /* set 1V range */
READ?                           /* initialize digitizer, fetch readings */
/* retrieve the readings from the digitizer */
```

Comments

1. **Digitizer INPut Commands**. The CONFigure command sets most of the (INPut) subsystem parameters to the same values as set by the INP commands listed in the program. The INPut commands were executed to show the context in which they are used.

2. **Disabling an Input Port**. When taking readings, it is **not** necessary to disable the channel's input port that is not being used (INP3:STAT OFF). It was done in this program to show the versatility of the digitizer.

3. **Digitizer Measurement Range**. In this program, the measurement range is specified with the SENSe:VOLTage:RANGe command, rather than with the *expected value* parameter of the CONFigure command. For most applications, however, it is easier to specify an **expected value**. SENSe:VOLTage:RANGe can be used to change the signal range without changing the entire digitizer configuration with CONFigure.

Taking a Burst of Readings

This program demonstrates:

- how to set the arm count for multiple bursts of (post-arm) readings
- how to set the arm source to VXI backplane trigger line ECLTRG0 and use the HP E1406 Command Module to apply arming pulses to the trigger line.

ARMCNT.C

```
/* digitizer commands */
*RST;*CLS                          /* reset and clear the digitizer */
CONF1:ARR:VOLT (10), 1,(@1)        /* set 10 readings, 1V range, S/E input  */
                                   /* port 1*/
    ARM:SOUR1 ECLT0                    /* arm source is ECLTRG0 trigger line */
    ARM:COUN 3                     /* set arm count for 3 bursts */
OUTP:ECLT0:STAT ON                 /* enable line ECLTRG0 - Command Module */
INIT                               /* place digitizer in wait-for-arm state */
/* send arming pulse - Command Module commands*/
OUTP:ECLT0:IMM                     /* apply a pulse to ECLTRG0 */
/* ask for next pulse */
OUTP:ECLT0:IMM                     /* apply a 2nd pulse to ECLTRG0 */
/* ask for next pulse */
OUTP:ECLT0:IMM                     /* apply a 3rd pulse to ECLTRG0 */
/* digitizer command */
FETC1?                                 /* retrieve digitizer readings */
```

Comments

1. **Arm and Trigger Counts**. The arm count is the number of reading bursts or arm signals the digitizer is to accept before the digitizer returns to the idle state. When the digitizer receives an arm, it takes a reading each time a trigger is received, and continues to take readings until the trigger count is reached. Notice the CONFigure command specifies (10) readings (trigger count = 10). Therefore, with an arm count of three, 30 readings are taken before the digitizer returns to the idle state.

2. **Additional Information**. Digitizer arming and triggering is covered in Chapter 3 - "Understanding the Digitizer".

Level Arming

This program demonstrates:

- how to set the arm source such that the digitizer is armed when the input signal enters a specified voltage range (window) from a level outside the range.
- how to set the arm slope such that an arm occurs when the input signal enters the arm window from either a positive-going or negative-going direction.
- how to set the voltage levels which define the arm window.

For this example, the arm signal should be applied to the HI input of port 3, with the LO input of port 3 grounded.

ARMLEVEL.C

```
*RST;*CLS                       /* reset and clear the digitizer */
CONF1:ARR:VOLT (10),10,(@3)         /* set 10 readings on Diff port 3 */
    ARM:SOUR1 INT1              /* set arm source 1 to level arming */
    ARM:SOUR2 HOLD              /* disable arm source 2 */
    ARM:SLOP1 EITH              /* arm when signal enters window from */
                                /* either direction */
    ARM:LEV1:POS 4              /* set arm window lower boundary */
    ARM:LEV1:NEG 6              /* set arm window upper boundary */
    ARM:COUN 2                  /* set two measurement bursts */
INIT                            /* put digitizer in wait-for-arm state */
FETCH?                          /* retrieve readings after arms occur */
```

Comments

1. **The Arm Window**. The arming window set by this program is +4V to +6V on the digitizer's 10V range. The upper boundary of the window (6V) is set with the ARM:LEVel1:NEGative command. This means when a NEGative-going input signal reaches 6V, the digitizer is armed. Similarly, the lower boundary (4V) of the window is set with the ARM:LEVel1:POSitive command. Thus, when a POSitive-going input signal reaches 4V, the digitizer is armed. Because the NEGative-going level is greater than the POSitive-going level, the digitizer is armed each time (up to the arm count) the signal **enters** the window.

2. **Additional Information**. Level arming is covered in detail in Chapter 3, in the section "Arming and Triggering".

Pre- and Post-Arm Readings

This program demonstrates:

- how to program the digitizer to take a minimum of 100 pre-arm readings and 100 post-arm readings.
- how to set the arm source to an external signal applied to the "Ext 1" BNC.

PREPOST.C

```
*RST;*CLS                          /* reset and clear the digitizer */
CONF1:ARR:VOLT (200),2,(@3)            /* set 200 readings total, 2V range, */
                                   /* Diff input port 3*/
    ARM:SOUR1 EXT                  /* arm source is front panel "Ext 1" BNC */
    SENS1:SWE:OFFS:POIN -100       /* set 100 pre-arm readings */
INIT                               /* put digitizer in wait-for-arm state */
FETCH?                             /* retrieve pre- and post-arm readings */
```

Comments

1. **Pre-arm and Post-arm Reading Count**. When measurements consist of pre- and post-arm readings, there must be at least three pre-arm readings and seven post-arm readings specified. Note that pre-arm readings are preceded by a minus (-) sign.

2. **Total Reading Count**. The *size* parameter of the CONFigure command specifies the total number of readings (pre- and post-arm). SENS1:SWE:OFFS:POIN specifies the number of pre-arm readings. The number of post-arm readings in this program is then (200) - 100 = 100. The total reading count can be changed without re-sending the CONFigure command by using the TRIGger:STARt:COUNt command or SENSe:SWEep:POINts command.

3. **Pre-arm Readings**. Pre-arm readings start when the digitizer receives the INITiate[:IMMediate] command. Pre-arm readings continue until an arm is received. Arms are ignored until the pre-arm reading count is reached. If the pre-arm count is exceeded before the arm occurs, the last SENSe:SWEep:OFFSet:POINts number of readings taken are stored in memory.

Specifying a Sample Rate

This program demonstrates:

- how to set the digitizer trigger (sample) source
- how to set the sample rate

In this example, a 1 kHz square wave is sampled at a rate which includes the 11th harmonic. All samples are post-arm.

SAMPLE.C

```
*RST;*CLS                       /* reset and clear the digitizer */
CONF1:ARR:VOLT (50),10,(@3)         /* set 50 readings, 10V range */
    ARM:SOUR1 IMM               /* arm when put in wait-for-arm state */
    TRIG:SOUR TIM                   /* set trigger source to timer 1 */
    TRIG:TIM1 20E-6             /* set sample period to 20 us */
READ?                           /* put digitizer in wait-for-arm state */
                                /* and retrieve the readings */
```

Comments

1. **Sample Period and Sample Count**. The period at which to sample a 1 kHz signal and include the 11th harmonic is determined by:

period = 1 / 4(f_C) = 1 / 4(11 kHz) = 22.7 μs

where f_C is the frequency of the 11th harmonic (11 kHz). Because of the trigger source used in this program (TIMer), the sample periods available are 1, 2, 4, through 1E8, 2E8, 4E8 multiples of the reference clock (the internal 20 MHz oscillator set by the CONFigure command). Therefore, given the 22.7 μs sample period calculated, the actual sample period used is 20 μs.

The number of samples to take is computed by:

sample count = signal period (fundamental)/actual sample period
= .001/.000020 = 50

2. **Specifying the Sample Count**. The sample count (i.e. trigger count) is specified by the *size* parameter (50) of the CONFigure command. This is the most convenient way to specify the sample count since size is a required parameter of CONFigure. TRIGger:STARt:COUNt can be used to set/change the sample count without also changing the entire configuration with CONFigure.

3. **Additional Information**. Additional information on trigger sources and sample rates is found in Chapter 3 - "Understanding the Digitizer", and in Chapter 4 - "Command Reference".

Dual Rate Sampling

This program demonstrates:

- how to set up the digitizer's dual rate sampling function whereby pre-arm and post-arm readings are taken at different sample rates.
- how level arming can be used with dual rate sampling

The digitizer pre-arm samples at 50 ns until the level of the input signal on channel 1 reaches 5V. At 5V, the digitizer is armed and post-arm samples at 10 ms.

DUALSAMP.C

```
*RST;*CLS                       /* clear and reset the digitizer */
CONF1:ARR:VOLT (20),10,(@3)         /* set 20 readings, 10V range */
    ARM:SOUR1 INT1              /* set arm source 1 for level arming */
    ARM:SOUR2 HOLD              /* disable arm source 2 */
    ARM:SLOP1 POS               /* arm on increasing input signal */
    ARM:LEV1:POS 5              /* arm at 5V (10V range) */
    SENS:SWE:OFFS:POIN -10          /* set 10 pre-arm readings */
    TRIG:SOUR DTIM              /* set dual rate sampling */
    TRIG:TIM1 50E-9             /* set sample rate for pre-arm readings */
    TRIG:TIM2 10E-3             /* set sample rate for post-arm readings */
INIT                            /* put digitizer in wait-for-arm state */
FETCH?                          /* retrieve readings */
```

Comments

1. **Dual Rate Sampling Periods**. The dual rate sampling trigger source DTIMer uses the internal timer sources. Thus, one sample period must equal the reference period. In this program, the digitizer's internal 20 MHz oscillator is the reference source. Therefore, one sample rate must be 50 ns (1 / 20 MHz).

2. **Pre- and Post-Arm Sample Rates.** With dual rate sampling, pre-arm and post-arm readings occur at different sample rates. It is recommended that pre-arm readings use the faster of the two sample rates. The reason is once the arm is received, one addtional sample at the pre-arm rate must occur before the post-arm rate is used.

3. **Dual Rate Sampling Reference Sources**. The reference sources available with trigger source DTIMer are:

INTernal - the digitizer's internal 20 MHz oscillator (default source)
CLK10 - the VXIbus system's 10 MHz clock
ECLTrg0 - the VXI backplane ECLTrg0 trigger line
ECLTrg1 - the VXI backplane ECLTrg1 trigger line
EXTernal2 - the digitizer's front panel "Ext 2" BNC port

When using any of these references, one of the two sample rates must be set to the reference period. Also, the reference used must be specified with the SENSe:ROSCillator:SOURce command, and if the source is ECLTrg0/1 or EXTernal2, the frequency of the reference must be specified with the SENSe:ROSCillator:EXTernal:FREQuency command.

4. **Other Dual Rate Sampling Sources.** When the dual rate sampling source is DECLtrg, ECLTrg0 paces the pre-arm readings and ECLTrg1 paces the post-arm readings. The sample rates are determined entirely by the source(s) driving the trigger lines. When the dual rate sampling source is DEXTernal, front panel port "Ext 1" paces the pre-arm readings and port "Ext 2" paces the post-arm readings. The sample rates are determined entirely by the source(s) driving the ports.

5. **Additional Information.** More information on dual rate sampling is found in Chapter 3 - "Understanding the Digitizer".

Using Multiple Digitizers

This program demonstrates:

- how to route the internal reference clock from one digitizer to a second digitizer, and how to configure the digitizers such that they are armed simultaneously.

MULT_AD.C

```
/* digitizer 1 */
CONF1:ARR:VOLT (10),1,(@1)          /* set 10 readings, 1V range, on */
                                    /* channel 1, S/E input port 1 */
    SENS:ROSC:SOUR INT                  /* reference source is internal */
                                    /* 20 MHz oscillator */
    OUTP:ECLT0:FEED 'SENS:ROSC'     /* route reference oscillator */
                                    /* clock signal to ECLT0 */
    OUTP:ECLT0:STAT ON              /* enable routing of the signal */
    ARM:SOUR1 TTLT0                     /* arm source is TTLT0 trigger line */
    TRIG:SOUR TIM                   /* trigger source is period derived */
                                    /* from the TRIG:TIM command */
    TRIG:TIM1 20E-6                 /* set sample period to 20 us */
    SENS2:FUNC 'VOLT4'              /* set up input port 4 on channel 2 */
    SENS2:VOLT:RANG 5               /* set voltage range on channel 2 */
                                    /* the arm and trigger sources are */
                                    /* the same as for channel 1 */

/* digitizer 2 */


CONF1:ARR:VOLT (10),1,(@1)          /* set 10 readings, 1V range, on */
                                    /* channel 1, S/E input port 1 */
    SENS:ROSC:SOUR ECLT0            /* reference oscillator source is */
                                    /* ECLT0 trigger line */
    SENS:ROSC:EXT:FREQ 20E6         /* specify frequency of clock signal */
    ARM:SOUR1 TTLT0                     /* arm source is TTLT0 trigger line */
    TRIG:SOUR TIM                       /* trigger source is period derived */
                                    /* from the TRIG:TIM command */
    TRIG:TIM1 20E-6                 /* set sample period to 20 us */
    SENS2:FUNC 'VOLT4'              /* set up input port 4 on channel 2 */
    SENS2:VOLT:RANG 5               /* set voltage range on channel 2 */
                                    /* the arm and trigger sources are */
                                    /* the same as for channel 1 */
```

Continued on Next Page

```
/* digitizer commands */
INIT          /* put digitizer 1 in wait-for-arm state */
INIT          /* put digitizer2 in wait-for-arm state */

/* Command Module commands */
OUTP:TTLT0:STAT ON       /* enable line TTLT0 */
 "Press Enter (return) to arm the digitizers"
OUTP:TTLT0:IMM           /* apply a pulse to TTLT0 */

/* digitizer commands */
FETC1?                   /* retrieve readings from digitizer1, channel 1 */
FETC2?                   /* retrieve readings from digitizer1, channel 2 */
FETC1?                   /* retrieve readings from digitizer2, channel 1 */
FETC2?                   /* retrieve readings from digitizer2, channel 2 */
```

Comments

1. **Synchronizing Digitizers**. By routing (OUTPut) the reference clock from one digitizer to all digitizers in the system, the sample rate is derived from the same reference. Digitizer samples can then be taken at precise intervals of each other.

2. **Input Channels**. In this program, samples are taken on the single ended and differential inputs of two digitizers, for a total of four channels. When sampling, either the channel's single ended input **or** the differential inputs can be used. You cannot use both sets of inputs simultaneously on a single channel.

3. **CONFiguring Channels Independently**. In this program, the digitizers' single ended input ports (channel 1) and differential input ports (channel 2) are set to two different signal ranges. This was done using SENSe:FUNCtion to select the port and SENSe:VOLTage[:DC]:RANGe to set the range. CONFigure was not used since executing CONF2:ARR:VOLT (10),5,(@3) would not only set the range on channel 2, but would also change the settings made following the first CONFigure (CONF1) command.

4. **Routing/Sourcing Signals**. If a digitizer trigger port (e.g. "Ext 1", "Ext 2") or a VXI backplane trigger line (ECLTrg<n >, TTLTrg<n >) is the source of a reference signal, arm signal or trigger signal, then that port or line cannot be used to route (OUTPut) a synchronization signal.

5. **Retrieving Readings**. When readings are retrieved from digitizer memory using the FETCh? command, each channel's readings must be retrieved separately. That is, channel 1's readings must be fetched and then channel 2's must be fetched, or vice versa. However, VME bus data transfers (retrievals) allow both channels' readings to be retrieved simultaneously. The "VME Bus Data Transfers" examples in this chapter show how this is done.

6. **Additional Information**. Additional information on using arming, triggering, and reference signals to synchronize other digitizers can be found in Chapter 3 - "Understanding the Digitizer". Additional information on the CONFigure command can be found in Chapter 1 - "Getting Started".

Using the Packed Data Format

This program demonstrates:

- how to specify the digitizer's packed data format
- how to remove the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header which preceeds the data
- how to assign a label to identify a set of readings
- how to convert the readings to voltages

The program takes 20 post-arm samples which are returned in the digitizer's packed data format. For completeness, the entire C language version of the program is listed. (The program is also contained on the example programs disk - HP E1429-10302).

PACKED.C

```
/* PACKED.C - This program takes 20 post-arm samples and returns the readings */
/* in the digitizer's packed data format. A label identifying the readings */
/* is assigned to the four least significant bits. The arbitrary block header */
/* preceding the readings is removed and the packed data is coverted to the */
/* measured voltages. The program features the FORMat and READ? commands. */

/* Include the following header files */

#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>
#include   <malloc.h>
#include   <cfunc.h>          /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L        /* I/O path from PC to the digitizer, via the HP E1406 */
```

Continued on Next Page

```
/* Function Prototypes */

void rst_clr(void);
void ad_confread(void);
void check_error(char *func_tion);

/*************************************************************************/
void main(void)       /* run the program */
{
rst_clr();            /* reset the digitizer */
ad_confread();        /* function which configures the digitizer */
                      /* and takes the readings */
}

/*************************************************************************/
void ad_confread(void)
{
int   length = 0, loop = 0, i = 0, swap = 0, bytes = 0, label = 0;
int   *rdgs;
char lf_remove[1];

/* use the "set_commands" array to configure single ended input port */
/* 1 on digitizer channel 1 */

char static *set_commands[] =
{"CONF1:ARR:VOLT (20),1,(@1)",   /* set 20 readings, 1V range  */
       "FORM PACK",              /* set packed reading format */
       "DIAG:CHAN:LAB 1"};       /* add label to each reading */

length = (sizeof(set_commands) / sizeof(char*));

       /* Execute each command using a loop */

for (loop = 0; loop < length; loop++)
 {
   IOOUTPUTS(ADDR, set_commands[loop], strlen(set_commands[loop]));
 }

      /* function call to check for digitizer configuration errors */

check_error("ad_confread");

/* dynamically allocate memory for readings */

rdgs =  malloc(20 * sizeof(int));
```

Continued on Next Page

```
/* set number of bytes placed in memory, and number of bytes read */

swap = sizeof(int);          /* place 2 bytes/reading in memory */
bytes = 20 * swap;  /* read 40 bytes */

IOOUTPUTS(ADDR, "READ?", 5);    /* retrieve the readings */

IOENTERAB(ADDR, rdgs, &bytes, swap);          /* enter the readings and */
                                              /* remove the block header */

/* Remove line feed which trails the last data byte */

length = 1;
IOENTERS(ADDR, lf_remove, &length);

/* print    label */

label = (rdgs[0] & 0x000F);
printf("\nLabel #: %d", label);

/* convert and print each reading as a voltage */

for (i = 0; i < 20; i++)
  {
       rdgs[i] /= 16;          /* remove label from each reading */
       if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
          printf("\nReading overrange");
       else
          printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0005));
  }

free(rdgs);
}

/*****************************************************************************/
void rst_clr(void)
{
       /* Reset and clear the digitizer */

IOOUTPUTS(ADDR, "*RST;*CLS", 9);      /* Send (9) characters */
}
```

Continued on Next Page

```
/*****************************************************************************/
void check_error(char *func_tion)
{
char  into[161];
intlength = 160;

IOOUTPUTS(ADDR, "SYST:ERR?", 9);    /* Query error register */
IOENTERS(ADDR, into, &length);      /* Enter error message */

if (atoi(into) != 0)          /* Determine if error is present */
                              /* If errors present, print and exit */
{
      while (atoi(into) != 0)
      {
         printf("Error %s in function %s\n\n", into, func_tion);
         IOOUTPUTS(ADDR, "SYST:ERR?", 9);
         IOENTERS(ADDR, into, &length);
      }

      exit(1);
}
}
```

Comments

1. **Packed Reading Format**. Packed digitizer readings are signed, 16-bit numbers preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header. Packed readings are always a number between -1.0230 (-2046) and +1.0235 (2047), and must be converted to voltages by the user.

2. **Line Feed Following Packed Readings**. Packed readings preceded by the arbitrary block header are also followed by a line feed (LF) character. When readings are retrieved from the digitizer, the LF remains in the output buffer. If the line feed is not removed with an additional "IOENTERS" statement, error -410 "Query INTERRUPTED" occurs the next time data is read from the digitizer.

3. **Channel Labels**. A numeric label identifying a set of readings can be specified using the four least significant bits of each reading. The label, which is any number from 0 to 15, is assigned using the DIAGnostic:CHANnel<chan >:LABel command. The label is included with the reading bits when data is returned in the PACKed,16 format. The assigned label is ignored when the data format is ASCii,9 or REAL,64. If no label is assigned, the four least significant bits of the reading are ’0’s. See "How Readings are Stored" in Chapter 3 for more information.

4. **Packed Reading Conversion Formula**. The equation for converting packed readings to voltages is:

$reading_{voltage} = (reading_{packed} / 16) * reading\ resolution$

The reading resolutions, which are a function of the signal range, are listed in Chapter 3 in the section "Converting Packed Readings".

VME Bus Data Transfers

The following programs demonstrate:

- how post-arm readings are transferred from the digitizer's A/D converter directly to the VME bus (VME_REAL.C)
- how segmented readings (pre- and post-arm) are transferred from the digitizer's A/D converter directly to the VME bus (VME_SEG1.C)
- how segmented, 32-bit readings (channel 2 and channel 1 combined) are transferred from the A/D converter to the VME bus

The system configuration on which programs VME_REAL.C and VME_SEG1.C were developed is listed on page 1-10.

VME_REAL.C

```
/* VME_REAL.C - This program reads data directly from the digitizer's A/D */

/* converter and places it on the VME (VXI data transfer) bus. Each time the */
/* digitizer's data register is accessed, a measurement is triggered and the */
/* reading is transferred to the VME bus during the same reading cycle. */

/* Include the following header files */

#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>
#include   <malloc.h>
#include   <cfunc.h>          /* This file is from the HP-IB Command Library */

#define ADDR 70905L           /* I/O path between the digitizer and PC */
#define CMD_MOD 70900L        /* I/O path between the digitizer and the Command Module */
```

Continued on Next Page

```
/* Function prototypes */

long get_base_addr(void);
void rst_clr(void);
void ad_conf(void);
void mem_read(long base_addr);
void check_error(char *function);
/*****************************************************************************/
void main(void)
{
long base_addr;                    /* variable for digitizer A24 base address */

rst_clr();                         /* reset and clear the digitizer */
base_addr = get_base_addr();       /* function call to calculate and */
                                   /* return digitizer A24 base address */
ad_conf();                         /* function call to configure the digitizer */
mem_read(base_addr);               /* function call which reads the data register */
}

/*****************************************************************************/
void ad_conf(void)
{
int    length = 0, loop = 0;

/* use the "set_commands" array to configure digitizer channel 1 */

char static *set_commands[] =
{"CONF1:ARR:VOLT (100),1,(@1)",          /* set 100 readings, on S/E port 1 */
"TRIG:STAR:SOUR VME",                    /* trigger source is reads of data register */
"VINS:CONF:VME:MODE GEN",                /* enable data transfer over the VME bus */
"VINS:CONF:VME:FEED 'CONV:CHAN1'",             /* set real time data transfer */
"INIT"};                                 /* place the digitizer in the wait-for-arm state */

length = (sizeof(set_commands) / sizeof(char*));

/* Execute each command using a loop */

for (loop = 0; loop < length; loop++)
  {
IOOUTPUTS(ADDR, set_commands[loop], strlen(set_commands[loop]));
  }

/* function call to check for digitizer configuration errors */

check_error("ad_conf");
}
```

Continued on Next Page

```
/*****************************************************************************/
void mem_read(long base_addr)
{
int    readings = 100, i = 0;
float *rdgs;

char rd_mem[80];            /* command string variable */

/* dynamically allocate memory for readings */

rdgs = malloc(100 * sizeof(float));

        /* Create the (HP E1406 Command Module) command string which reads the data register */

sprintf(rd_mem, "DIAG:PEEK? %ld, %d", base_addr+0x0C,16);

        /* Send DIAG:PEEK? command which accesses the data register */
        /* and triggers measurements, and then retrieve measurements */

for (i = 0;i < readings; i++)
{
        IOOUTPUTS(CMD_MOD, rd_mem, strlen(rd_mem));

        IOENTER(CMD_MOD, &rdgs[i]);
}

        /* Print    a subset of the readings */
for (i = 0;i < 20;i++)
{
        rdgs[i] /= 16;          /* remove label from reading */

        if (rdgs[i] >= 2047 ||rdgs[i] <= -2046)
           printf("\nReading overrange");
          else
           printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0005));
}

free(rdgs);
}

/*****************************************************************************/
long get_base_addr(void)
{
/* digitizer logical address */
long logical_addr = (ADDR - 70900L) * 8;
```

Continued on Next Page

```
        /* base address of (A24) offset register in A16 address space */
long base_addr = (0x1FC000 + (logical_addr * 64)) + 6;

float a24offst;                 /* A24 offset from A16 offset register */
char rd_addr[80];               /* command string variable */

        /* Create the command string which reads the A24 base address */
sprintf(rd_addr, "DIAG:PEEK? %ld, %d", base_addr,16);

        /* Send DIAG:PEEK? command */
IOOUTPUTS(CMD_MOD, rd_addr, strlen(rd_addr));

        /* Read value from offset register */
IOENTER(CMD_MOD, &a24offst);

        /* Multiply offset value by 256 for 24-bit address value */
a24offst *= 256.;

return (long)a24offst;
}

/*****************************************************************************/
void rst_clr(void)
/*****************************************************************************/
void check_error(char *func_tion)
```

Comments

1. **VME Data Transfer Modes**. There are two modes of VME data transfers: real time and post measurement. In a real time data transfer (shown in this program) accessing the digitizer's data register triggers a measurement and returns the A/D reading directly to the VME bus in the same measurement cycle. The reading(s) is also stored in digitizer memory. In a post measurement transfer, each data register access transfers a A/D reading from digitizer memory to the VME bus.

2. **Locating the Data Register**. Access to the data register is through its address which is mapped by the HP E1406 Command Module into A24 address space. The data register has an offset of 12 ($0C_{16}$) which is added to the A24 base address to form the complete register address. In the program, the C function long get_base_addr(void) determines the A24 base address by reading the digitizer's offset register in A16 address space. Detailed information on locating the data register can be found in Chapter 3 under the section "VME Bus Data Transfers".

3. **VME Bus Data Format**. Data is transferred over the VME bus in the digitizer's packed data format. Readings are 16-bits or 32-bits depending on the source specified by the VINStrument[:CONFigure]:VME:FEED <*source* > command. The *source* parameters are listed in Chapter 3 under the section "Setting the VME bus Transfer Mode".

4. **System Configuration**. The system configuration on which the program VME_REAL.C was developed is listed on page 1-10.

VME_SEG1.C

```
/* VME_SEG1.C - This program demonstrates how to transfer segmented readings */
/* over the VME bus. The program sets up 2 bursts (segments) of 10 pre-arm */
/* and 10 post-arm readings. A reading is taken each time the digitizer's */
/* data register is accessed, and is transferred real time, over the VME bus. */
/* Before the next burst is taken, bit 1 of (offset) register 43h is */
/* monitored to determine when the next segment of readings can be taken. */

      /* Include the following header files */

#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>
#include   <malloc.h>
#include   <cfunc.h>          /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L        /* I/O path from the PC to the digitizer */
#define CMD_MOD 70900L        /* Path from the PC to the Command Module */

/* Function Prototypes */

long get_base_addr(void);
void rst_clr(void);
void ad_conf(void);
void ad_read(long base_addr);
void check_error(char *func_tion);

/*****************************************************************************/
void main(void)                              /* run the program */
{
      long base_addr;                 /* variable for digitizer A24 base address */
      clrscr();
      rst_clr();                             /* reset the digitizer */
      base_addr = get_base_addr();           /* function call to get digitizer */
                                             /* A24 base address */
      ad_conf();                             /* function call which configures the digitizer */
      ad_read(base_addr);                    /* function call which reads the digitizer */
                                             /* data register */
}
```

```
/*****************************************************************************/
void ad_conf(void)
{
        int    length = 0, loop = 0;

        /* use the "set_commands" array to configure digitizer channel 1 */

        char static *set_commands[] =
        {"CONF1:ARR:VOLT (20),5,(@3)",          /* set 20 readings per burst, 5V range */
           "ARM:STAR:SOUR IMM",                 /* set arm source immediate */
           "ARM:STAR:COUN 2",                           /* set 2 bursts (arms) */
           "TRIG:STAR:SOUR VME",                /* Data register access triggers readings */
           "SENS:SWE:OFFS:POIN -10",            /* set 10 pre-arm readings */
           "VINS:CONF:VME:MODE GEN",            /* enable VME bus data transfers */
           "VINS:CONF:VME:FEED 'CONV:CHAN1'",   /* real time data transfer */
         "INIT"};                                       /* put digitizer in wait-for-arm state */

        length = (sizeof(set_commands) / sizeof(char*));

       /* Execute each command using a loop */

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR, set_commands[loop], strlen(set_commands[loop]));
           }

        /* function call to check for digitizer configuration errors */

        check_error("ad_conf");
}

/*****************************************************************************/
void ad_read(long base_addr)
{
        int    i, readings = 20, index = 0, loop = 0;
        float *rdgs, bit;
        char read_str[80], bit_str[80];          /* command string variables */

        /* dynamically allocate memory for readings */

        rdgs = malloc(40 * sizeof(float));

           /* Create the command string which reads the data register */

        sprintf(read_str, "DIAG:PEEK? %ld, %d", base_addr+0x0C,16);
```

Continued on Next Page

```
    /* Create the command string which reads bit 1 */

sprintf(bit_str, "DIAG:PEEK? %ld, %d", base_addr+0x43,8);

    /* Send DIAG:PEEK? to access the data register 20 times. */

while (loop < 2)       /* two bursts (segments) */
{
    for (i = index;i < readings; i++)
    {
            IOOUTPUTS(CMD_MOD, read_str, strlen(read_str));
            IOENTER(CMD_MOD, &rdgs[i]);
    }

      /* Check bit 1 of offset register 43h before proceeding with */
     /* the next segment. */

    do     /* decimal value of bit 1 */
    {
            IOOUTPUTS(CMD_MOD, bit_str, strlen(bit_str));
            IOENTER(CMD_MOD, &bit);
     } while ((int)bit & 2 == 0);

    index +=20;                /* increment index for next segment */
    readings +=20;   /* increment readings for next segment */
    loop++;                    /* increment loop */

}

    /* Convert to voltages and print the readings */
for (i = 0;i < 20; i++)
{
    rdgs[i] /= 16;             /* remove label from reading */

    if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
            printf("Reading overrange");
    else
            printf ("%.6E", (rdgs[i] * 0.0025));

    rdgs[i+20] /= 16;

    if (rdgs[i+20] >= 2047 || rdgs[i+20] <= -2046)
            printf("\t\tReading overrange\n");
    else
            printf("\t\t%.6E\n", (rdgs[i+20] * 0.0025));
}
```

Continued on Next Page

```
        free(rdgs);
}

/*****************************************************************************/
long get_base_addr(void)
{
        /* digitizer logical address */
        long logical_addr = (ADDR - 70900L) * 8;

       /* base address of (A24) offset register in A16 address space */
        long base_addr = (0x1FC000 + (logical_addr * 64)) + 6;

         float a24offst;         /* A24 offset from A16 offset register */

        char rd_addr[80];    /* command string variable */

           /* Create the command string which reads the A24 base address */
        sprintf(rd_addr, "DIAG:PEEK? %ld, %d", base_addr,16);

           /* Send DIAG:PEEK? command */
        IOOUTPUTS(CMD_MOD, rd_addr, strlen(rd_addr));

           /* Read value from offset register */
        IOENTER(CMD_MOD, &a24offst);

           /* Multiply offset value by 256 for 24-bit address value */
        a24offst *= 256.;

        return (long)a24offst;
}

/*****************************************************************************/
void rst_clr(void)
{

           /* Reset and clear the digitizer */

        IOOUTPUTS(ADDR, "*RST;*CLS", 9);
}

/*****************************************************************************/
void check_error(char *func_tion)
{
        char      into[161];
        int length = 160;
```

Continued on Next Page

```
        IOOUTPUTS(ADDR, "SYST:ERR?", 9);      /* Query error register */
        IOENTERS(ADDR, into, &length);        /* Enter error message */

        if (atoi(into) != 0)          /* Determine if error is present */
                                      /* If errors present, print    and exit */
        {
           while (atoi(into) != 0)
           {
                 printf("Error %s in function %s\n\n", into, func_tion);
                 IOOUTPUTS(ADDR, "SYST:ERR?", 9);
                 IOENTERS(ADDR, into, &length);
           }
           exit(1);
        }
}
```

Comments

1. **Segmented Readings**. Multiple bursts of pre-arm and post-arm readings segment memory (see Figure 3-13). When transferring segmented readings over the VME bus real time or post measurement, a partition window must be accounted for. A partition window is the period during which the digitizer configures each segment for data transfer. The partition window is monitored by bit 1 of digitizer offset register 43_{16}, and by bit 9 of the condition register in the Operation Status Group (Figure 3-13). A low-to-high transition of the bit indicates the next segment can be transferred.

2. **Monitoring the Partition Window**. It is only necessary to monitor the partition window bits when the digitizer readings are segmented and the data register is accessed at speeds available through an embedded controller. Monitoring bit 1 of offset register 43_{16} is faster than using SCPI commands to monitor bit 9 of the condition register.

3. **Locating the Registers**. Access to the data register and offset register 43_{16} is through their addresses which are mapped by the HP E1406 Command Module or the system resource manager into A24 address space. The data register offset (12_{10} or $0C_{16}$) and Offset register 43_{16} (67_{10}) is added to the A24 base address to form the complete register addresses. In the program, the C function long get_base_addr(void) determines the A24 base address by reading the digitizer's offset register in A16 address space. Detailed information on locating the data and offset 43_{16} registers can be found in Chapter 3 under the section "VME Bus Data Transfers".

VME Bus Data Transfers Using an Embedded Controller

The following programs transfer data over the VME bus using the following system configuration:

- Controller: RadiSys® EPC®-7 Embedded Controller
- Runtime library: Standard Instrument Control Library (SICL) for DOS
- Compiler: Borland© C++ (.CPP)

These programs are also contained on the C language example programs disk (HP E1429-10302).

SEGTST16.CPP

This program transfers 16-bit readings (real time) from the channel 1 A/D converter to the VME bus.

The include files and structure definitions used in this this program are listed following the SEGTST16.CPP and SEGTST32.CPP program listings.

```
// Options|Compiler|CodeGeneration|Model:    Set to Large
// Options|Directories|Include Directories:  Add C:\EPCONNEC\INCLUDE
// Options|Directories|Library Directories:  Add C:\EPCONNEC\LIB
// Project Items:  INST.CPP, E1429.CPP, SEGTST16.CPP, BSICL.LIB, EPCMSC.LIB

#include    <stdlib.h>
#include    <stdio.h>
#include "e1429.h"

#define BUFLEN     200

extern int    ierrno;

int    Measure(E1429 *Dig);
int    ReadData(E1429 *Dig);

/* ********************** Main ********************** */

void main(void) {
  int    Errors;
  E1429 *Dig;

  Errors = 0;
  ierrno = 0;
```

Continued on Next Page

```
  Dig = new E1429;
  if (!Dig->IsValid()) {
    printf("Digitizer could not be opened (%s).\n",
                        igeterrstr(igeterrno()));
    if (Dig != NULL)
      delete Dig;
    Dig = NULL;
    exit(1);
  }

  Errors += Measure(Dig);
  if (!Errors)
    Errors += ReadData(Dig);

  delete Dig;
  exit(0);
}

/* ********************** Measure ********************** */

int    Measure(E1429 *Dig) {
  int    Errors;
  char Buf[BUFLEN+1], **Com;
  static char *Commands[] =  {
    "*RST",
    "CONF1:ARR:VOLT (20), 5, (@3)",
    "ARM:STAR:SOUR IMM",
    "ARM:STAR:COUN 2",
    "TRIG:STAR:SOUR VME",
    "SENS:SWE:OFFS:POIN -10",
    "VINS:CONF:VME:MODE GEN",
    "VINS:CONF:VME:FEED 'CONV:CHAN1'",
    "INIT",
     NULL
  };

  for (Com = Commands; *Com != NULL; Com++)
    Dig->SendMessage(*Com);
  Errors = 0;
  while (Dig->GetErrorMessage(Buf,BUFLEN) != NULL) {
    printf("%s\n",Buf);
    Errors++;
  }
  return Errors;
}
```

Continued on Next Page

```
/* ********************** ReadData ********************** */

int    ReadData(E1429 *Dig) {
  int    SegCnt, ReadCnt;
  WORD *Readings, *Reading, *Reading2;
  const static NReadings = 20, NSegments = 2;
  const static float Scale = 0.0025 / 16.0;

  Reading = Readings = (WORD *)
          malloc(NSegments * NReadings * sizeof(WORD));
  for (SegCnt = 0; SegCnt < NSegments; SegCnt++) {
    while ((Dig->bGet(67) & 2) == 0);
    Dig->MGetDataReg((UWORD *) Reading,NReadings);
    Reading += NReadings;
  }

  printf("    **** Channel 1 ****\n");
  printf("      Seg 1     Seg 2\n");
  printf("   ---------- ----------\n");
  Reading  = Readings;
  Reading2 = Readings + NReadings;
  for (ReadCnt = 1; ReadCnt <= NReadings; ReadCnt++)
    printf("%2d%+10.4f%+10.4f\n",ReadCnt,Scale * *Reading++,Scale * *Reading2++);
  printf("\n");

  free(Readings);
  return 0;
}
```

SEGTST32.CPP

This program transfers 32-bit readings (real time) from the channel 2 and channel 1 A/D converters to the VME bus. In a 32-bit transfer, the upper 16-bits are the reading from channel 2 and the lower 16-bits are the reading from channel 1 (Figure 3-11).

The include files and structure definitions used in this this program are listed following the program listing.

```
// Options|Compiler|CodeGeneration|Model:    Set to Large
// Options|Directories|Include Directories:  Add C:\EPCONNEC\INCLUDE
// Options|Directories|Library Directories:  Add C:\EPCONNEC\LIB
// Project Items:  INST.CPP, E1429.CPP, SEGTST32.CPP, BSICL.LIB, EPCMSC.LIB

#include    <stdlib.h>
#include    <stdio.h>
#include "e1429.h"

#define BUFLEN     200

extern int    ierrno;

int    Measure(E1429 *Dig);
int    ReadData(E1429 *Dig);

/* ********************** Main ********************** */

void main(void) {
  int    Errors;
  E1429 *Dig;

  Errors = 0;
  ierrno = 0;

  Dig = new E1429;
  if (!Dig->IsValid()) {
    printf("Digitizer could not be opened (%s).\n",
                        igeterrstr(igeterrno()));
    if (Dig != NULL)
      delete Dig;
    Dig = NULL;
    exit(1);
  }

  Errors += Measure(Dig);
  if (!Errors)
    Errors += ReadData(Dig);

  delete Dig;
  exit(0);
}
```

Continued on Next Page

```
/* ********************** Measure ********************** */

int    Measure(E1429 *Dig) {
  int    Errors;
  char Buf[BUFLEN+1], **Com;
  static char *Commands[] =  {
    "*RST",
    "CONF1:ARR:VOLT (20), 5, (@3)",
    "CONF2:ARR:VOLT (20), 5, (@4)",
    "ARM:STAR:SOUR IMM",
    "ARM:STAR:COUN 2",
    "TRIG:STAR:SOUR VME",
    "SENS:SWE:OFFS:POIN -10",
    "VINS:CONF:VME:MODE GEN",
    "VINS:CONF:VME:FEED 'CONV:BOTH32'",
    "INIT",
     NULL
  };

  for (Com = Commands; *Com != NULL; Com++)
    Dig->SendMessage(*Com);
  Errors = 0;
  while (Dig->GetErrorMessage(Buf,BUFLEN) != NULL) {
    printf("%s\n",Buf);
    Errors++;
  }
  return Errors;
}

/* ********************** ReadData ********************** */

int    ReadData(E1429 *Dig) {
  int    SegCnt, ReadCnt;
  LONG *Readings, *Reading, *Reading2;
  const static NReadings = 20, NSegments = 2;
  const static float Scale = 0.0025 / 16.0;

  Reading = Readings = (LONG *)
          malloc(NSegments * NReadings * sizeof(LONG));
  for (SegCnt = 0; SegCnt < NSegments; SegCnt++) {
    while ((Dig->bGet(67) & 2) == 0);
    Dig->MGetLongDataReg((ULONG *) Reading,NReadings);
    Reading += NReadings;
  }
```

Continued on Next Page

```
  printf("    **** Channel 1 ****     **** Channel 2 ****\n");
  printf("      Seg 1     Seg 2         Seg 1     Seg 2\n");
  printf("   ---------- ----------   ---------- ----------\n");
  Reading  = Readings;
  Reading2 = Readings + NReadings;
  for (ReadCnt = 1; ReadCnt <= NReadings; ReadCnt++) {
    printf("%2d%+10.4f%+10.4f    %+10.4f%+10.4f\n",ReadCnt,
      Scale*LOWORD(*Reading), Scale*LOWORD(*Reading2),
      Scale*HIWORD(*Reading), Scale*HIWORD(*Reading2)
    );
    Reading++;
    Reading2++;
  }
  printf("\n");

  free(Readings);
  return 0;
}
```

SEGTST16.CPP and SEGTST32.CPP #include Files

The following files are used with programs SEGTST16.CPP and SEGTST32.CPP:

- INST.H
- INST.CPP
- E1429.H
- E1429.CPP

INST.H

```
#ifndef INST_DEFD

#define INST_DEFD

#include   <sicl.h>

#define BYTE  char
#define WORD  short int
#define LONG  long
#define UBYTE unsigned char
#define UWORD unsigned short int
#define ULONG unsigned long
```

Continued on Next Page

```
#define LOWORD(lWord) ((WORD)(lWord))
#define HIWORD(lWord) ((WORD)((LONG)(lWord) >> 16))

class RegInst {
  static RegInst *MappedInst;
  int    Valid;
  int    Mapped;
  int    MapSpace;
  unsigned int    PageStart, PageCount;
  UBYTE *SuggestedAddr;
protected:
  INST   Inst;
  UBYTE *BaseAddr;
  void   Unmap(void);
public:
  RegInst(UWORD lAddr = 0);
 ~ RegInst(void);

  int      IsValid(void) { return (this != NULL && Valid); }
  INST  GetInstID(void) { return Inst; }
  void  SetMapping(int    mapSpace, unsigned int    pageStart, unsigned int    pageCount,
UBYTE *suggestedAddr);
  void  Map(void);

  UBYTE bGet(UWORD offset) { if (!Mapped) Map(); return ibpeek((UBYTE *)
(BaseAddr+offset)); }
  void  bSet(UWORD offset, UBYTE value = 0) { if (!Mapped) Map(); ibpoke((UBYTE *)
(BaseAddr+offset),value); }
  UWORD wGet(UWORD offset) { if (!Mapped) Map(); return iwpeek((UWORD *)
(BaseAddr+offset)); }
  void  wSet(UWORD offset, UWORD value = 0) { if (!Mapped) Map(); iwpoke((UWORD *)
(BaseAddr+offset),value); }
  void  wMGet(UWORD offset, UWORD *dest, ULONG count) {
    if (!Mapped) Map(); iwpopfifo(Inst,(UWORD *) (BaseAddr+offset),dest,count,1);
  }
  void  wMSet(UWORD offset, UWORD *src,  ULONG count) {
    if (!Mapped) Map(); iwpushfifo(Inst,(UWORD *) (BaseAddr+offset),src,count,1);
  }
  void  lMGet(UWORD offset, ULONG *dest, ULONG count) {
    if (!Mapped) Map(); ilpopfifo(Inst,(ULONG *) (BaseAddr+offset),dest,count,1);
  }
  void  lMSet(UWORD offset, ULONG *src,  ULONG count) {
    if (!Mapped) Map(); ilpushfifo(Inst,(ULONG *) (BaseAddr+offset),src,count,1);
  }
};
```

Continued on Next Page

```
class MessInst : public RegInst {
public:
  MessInst(WORD lAddr) : RegInst(lAddr) { }
  void  Clear(void) { iclear(Inst); }
  void  SendMessage(const char *message);
  char *ReceiveMessage(char *message, int    maxLen = 80);
  char *GetErrorMessage(char *message, int    maxLen = 80);
};

#endif
```

INST.CPP

```
#include    <stdlib.h>

#include    <stdio.h>
#include    <string.h>
#include "inst.h"

#define BUFLEN 200

/* ************************** RegInst ************************** */

RegInst *RegInst::MappedInst = NULL;

RegInst::RegInst(UWORD lAddr) {
  char Buf[32];

  BaseAddr = NULL;
  MapSpace = 0;
  PageStart = 0;
  PageCount = 0;
  sprintf(Buf,"vxi,%u",lAddr);
  Valid = ((Inst = iopen(Buf)) != NULL) ? 1 : 0;
  Mapped = 0;
}

RegInst::~ = RegInst(void) {
  if (Inst != NULL) {
    Unmap();
    iclose(Inst);
  }
}
```

Continued on Next Page

```
void RegInst::SetMapping(int    mapSpace, unsigned int    pageStart, unsigned int
pageCount, UBYTE *suggestedAddr) {
  int    WasMapped;

  if (this != NULL) {
    WasMapped = Mapped;
    if (Mapped)
      Unmap();
    MapSpace      = mapSpace;
    PageStart     = pageStart;
    PageCount     = pageCount;
    SuggestedAddr = suggestedAddr;
    if (WasMapped)
      Map();
  }
}

void RegInst::Map(void) {
  if (this != NULL) {
    Valid = 0;
    if (Inst != NULL) {
      MappedInst->Unmap();
      BaseAddr  = imap(Inst,MapSpace,PageStart,PageCount,SuggestedAddr);
      if (BaseAddr != NULL) {
        MappedInst = this;
        Valid = 1;
        Mapped = 1;
      }
    }
  }
}

void RegInst::Unmap(void) {
  if (this != NULL) {
    Valid = 0;
    if (Inst != NULL) {
      if (this == MappedInst) {
        iunmap(Inst,BaseAddr,MapSpace,PageStart,PageCount);
        MappedInst = NULL;
      }
      Valid = 1;
    }
    BaseAddr = NULL;
    Mapped = 0;
  }
}
```

Continued on Next Page

```
/* ************************* MessInst ************************** */

void MessInst::SendMessage(const char *message) {
  unsigned long ActualLen;
  char Buf[BUFLEN+1];

  strcpy(Buf,message);
  strcat(Buf,"\n");
  iwrite(Inst,Buf,strlen(Buf),0,&ActualLen);
}

char *MessInst::ReceiveMessage(char *message, int    maxLen) {
  int    Reason;
  unsigned long ActualLen;
  char *SPtr, Buf[BUFLEN+1];

  iread(Inst,message,maxLen,&Reason,&ActualLen);
  message[ActualLen] = '\0';
  SPtr = message + strlen(message) - 1;
  while (SPtr >= message && *SPtr == '\n')
    *SPtr-- = '\0';
  while (Reason == I_TERM_MAXCNT)
    iread(Inst,Buf,BUFLEN,&Reason,&ActualLen);
  return message;
}

char *MessInst::GetErrorMessage(char *message, int    maxLen) {
  char *MPtr;

  SendMessage("SYST:ERR?");
  ReceiveMessage(message,maxLen);
  MPtr = (atoi(message)) ? message : NULL;
  return MPtr;
}
```

E1429.H

```
#ifndef E1429_DEFD

#define E1429_DEFD

#include "inst.h"

class E1429 : public MessInst {
  int    MemoryMode;
public:
  E1429(WORD lAddr = 40);
  void  SetDataReg(UWORD value) { wSet(0x0c,value); }
  UWORD GetDataReg(void) { return wGet(0x0c); }
  void  MGetDataReg(UWORD *dest, ULONG count) { wMGet(0x0c,dest,count); }
  ULONG GetLongDataReg(void); // Cannot use lGet here because of ilpeek problem
  void  MGetLongDataReg(ULONG *dest, ULONG count) { lMGet(0x0c,dest,count); }
};

#endif
```

E1429.CPP

```
#include    <stdlib.h>


#include "e1429.h"

E1429::E1429(WORD lAddr) : MessInst(lAddr) {
  SetMapping(I_MAP_EXTEND,0,1,NULL);
  Map();
}

ULONG E1429::GetLongDataReg(void) {
  ULONG Result;

  lMGet(0x0c,&Result,1);
  return Result;
}
```

Local Bus Data Transfers

The following programs demonstrate:

- how readings are transferred over the Local bus from a single digitizer to the HP E1488 memory card (LOCAL_AD.C)
- how readings in digitizer memory are transferred over the Local bus from two digitizers to the HP E1488 memory card (LBUS2PST.C)
- how readings from two digitizer A/Ds are transferred over the Local bus to the HP E1488 memory card (LBUSAUTO.C)

For completeness, the entire C language versions of these programs are listed. The programs are also contained on the example programs disk (HP p/n E1429-10302).

LOCAL_AD.C

This program transfers readings from both digitizer channels to the HP E1488 memory card. The readings are tranferred directly from the digitizer A/Ds.

```
/* LOCAL_AD.C - This program demonstrates a Local bus data transfer */
/* directly (real time) from the HP E1429B digitizer A/Ds to the E1488A memory */
/* card. */
        /* Include the following header files */

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <malloc.h>
#include    <cfunc.h>              /* This file is from the HP-IB Command Library Disk */

#define    ADDR 70905L             /* I/O path from the PC to the digitizer */
#define ADDR_MEM 70903L            /* I/O path from the PC to the memory card */

/* Function Prototypes */

void rst_clr(long address);
void configure(void);
void initiate(void);
void check_error(char *array, long address);
```

Continued on Next Page

```
/*****************************************************************************/
void main(void)                 /* run the program */
{
        rst_clr(ADDR);          /* reset the digitizer */
        rst_clr(ADDR_MEM);      /* reset the memory card */
        configure();            /* configure the digitizer and the memory card */
        initiate();             /* initiate the digitizer and memory card; */
                                /* retrieve the readings from the memory card */
}

/*****************************************************************************/
void configure(void)
{
        int    length = 0, loop = 0;

        /* use the "digitizer" array to configure the digitizer for readings on */
        /* each channel's (HI) differential input */

        char static *digitizer[] =
        {"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                                /* channel 1 input port 3 */
         "CONF2:ARR:VOLT (10),5,(@4)",          /* configure channel 2, port 4 */
           "VINS:LBUS:RES",                     /* reset the Local bus chip */
           "VINS:LBUS:MODE GEN",                /* set Local bus mode to GENerate */
           "VINS:LBUS:FEED 'CONV:BOTH'"};       /* set Local bus feed (direct from A/D)

        /* use the "memory" array to configure the memory card */

        char static *memory[] =
        {"FORM:DATA PACK",          /* set packed data format */
         "TRAC:DEL:ALL",            /* delete all readings on memory card */
         "TRAC:DEF SET2, 40",       /* store readings (40 bytes) in "SET2" */
         "VINS:LBUS:RES",                   /* reset the Local bus chip */
         "VINS:LBUS:MODE CONS",     /* set Local bus mode to consume */
         "STAT:OPC:INIT OFF"};      /* execute *OPC? after INIT is parsed */

        /* Execute each command in "digitizer" */

        length = (sizeof(digitizer) / sizeof(char*));

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR, digitizer[loop], strlen(digitizer[loop]));
           }

        /* Execute each command in "memory" */
```

Continued on Next Page

```
        length = (sizeof(memory) / sizeof(char*));

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR_MEM, memory[loop], strlen(memory[loop]));
           }

        /* check for configuration errors */

        check_error("digitizer", ADDR);

        check_error("memory", ADDR_MEM);

}

/*****************************************************************************/
void initiate(void)
{
        int    i = 0, readings = 20, swap = 0, bytes = 0, length = 1, *rdgs;
        float rdy;
        char lf_remove[1];

        /* dynamically allocate memory for readings */

        rdgs = malloc(20 * sizeof(float));      /* allocate computer memory for reading storage */
        swap = sizeof(int);                     /* each reading in memory is two bytes */
        bytes = 20 * swap;                      /* read 40 bytes (2 channels, 10 readings per channel) */

        IOOUTPUTS(ADDR_MEM, "INIT", 4);             /* initiate the memory card */
        IOOUTPUTS(ADDR_MEM, "*OPC?", 5);   /* wait for INIT to parse before continuing */
        IOENTER(ADDR_MEM, &rdy);                /* enter *OPC? response from memory card */

        IOOUTPUTS(ADDR, "INIT", 4);             /* initiate the digitizer */
        IOOUTPUTS(ADDR, "*OPC?", 5);    /* allow readings to complete before */
                                        /* retrieving them from the memory card */
        IOENTER(ADDR, &rdy);                    /* enter *OPC? response from digitizer */

        IOOUTPUTS(ADDR_MEM, "TRAC:DATA? SET2", 15); /* retrieve readings from memory card
*/
        IOENTERAB(ADDR_MEM, rdgs, &bytes, swap); /* enter readings and remove block header */

        /* remove line feed which trails the last data byte */

        IOENTERS(ADDR_MEM, lf_remove, &length);
```

Continued on Next Page

```
        /* convert and display the readings; readings are in the sequence */
        /* channel 2 reading 1, channel 1 reading 1, channel 2 reading 2, */
        /* channel 1 reading 2, and so on */

        for (i = 0; i < readings; i++)
        {
            rdgs[i] /= 16;
            if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
                    printf("\nReading overrange");
            else
                    printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0025));

        }

        free(rdgs);
}

/*****************************************************************************/
void rst_clr(long address)
{
            /* Reset and clear the digitizer and memory card */

        IOOUTPUTS(address, "*RST;*CLS", 9);
}

/*****************************************************************************/
void check_error(char *array, long address)
{
        char      into[161];
        int length = 160;

        IOOUTPUTS(address, "SYST:ERR?", 9);          /* Query error register */
        IOENTERS(address, into, &length);            /* Enter error message */

        if (atoi(into) != 0)                         /* Determine if error is present */
                                                     /* If errors present, print    and exit */
        {
            while (atoi(into) != 0)
            {
                    printf("Error %s in array %s\n\n", into, array);
                    length = 160;
                    IOOUTPUTS(address, "SYST:ERR?", 9);
                    IOENTERS(address, into, &length);
            }
            exit(1);
        }
}
```

Comments

1. **Digitizer Configuration.** Both channel's HI differential inputs (ports 3 and 4) are CONFigured for 10 readings on the 5V range. Two readings (channel 1 and channel 2) are taken on each sample trigger. The sample rate, as set by the CONFigure command, is 50 ns (20 MHz). Thus, data is transferred at a rate of 40 MSamples (80 MBytes)/second.

Before setting the digitizer's Local bus configuration the Local bus chip is reset. Next, the Local bus mode is set to GENerate and the feed (data source) is set to CONVerter:BOTH.

2. **Post-Arm Readings.** When transferring readings over the Local bus from the digitizer A/D, all readings must be post-arm.

3. **Initiating the Digitizer.** After the memory card is configured, it is INITiated first so that it is ready to receive the digitizer readings. When the digitizer is INITiated, *OPC? is used to allow the readings to complete and be transferred before they are retrieved from the memory card.

4. **Reading Sequence and Format.** When this program executes, the readings are transferred to the memory card and later displayed in the following sequence:

channel 2 reading 1
channel 1 reading 1
channel 2 reading 2
channel 1 reading 2
.
.
.

The memory card was set up to store the readings in the digitizer's packed data format. The packed readings are signed, 16-bit numbers preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header. Packed readings are always numbers between -1.0230 (-2046) and +1.0235 (2047). To convert the readings to voltages, each reading is divided by 16 to remove the data label bits (0 - 3), and is multipled by 0.0025 which is the reading resolution for the 5V range.

5. **Additional Information.** Additional information on Local bus operation and on the Local bus commands can be found in Chapter 3 - "Understanding the Digitizer", and in Chapter 4 - "Command Reference."

LBUS2PST.C This program transfers readings in digitizer memory from two digitizers to the HP E1488 memory card. The program shows how the digitizers are used in the Local bus GENerate and APPend modes.

```
/* LBUS2PST.C - This program demonstrates how to transfer readings in */
/* digitizer memory from multiple digitizers to the E1488 memory card. The */
/* leftmost digitizer is set to GENerate mode and the inner digitizer is set */
/* to the APPend mode. */

        /* Include the following header files */

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <malloc.h>
#include    <cfunc.h>                   /* This file is from the HP-IB Command Library Disk */

#define    ADDR_G 70905L                /* I/O path from the PC to the generator digitizer */
#define ADDR_A 70906L                       /* I/O path from the PC to the appender digitizer */
#define ADDR_MEM 70903L                 /* I/O path from the PC to the memory card */

/* Function Prototypes */

void rst_clr(long address);
void configure(void);
void initiate(void);
void check_error(char *func_tion, long address);

/*****************************************************************************/
void main(void)                     /* run the program */
{
        rst_clr(ADDR_G);            /* reset generator digitizer */
        rst_clr(ADDR_A);            /* reset appender digitizer */
        rst_clr(ADDR_MEM);          /* reset memory card */
        configure();                /* configure the digitizers and the memory card */
        initiate();                 /* initiate the digitizers and the memory card; */
                                    /* retrieve the readings from the memory card */
}

/*****************************************************************************/
void configure(void)
{
        int    length = 0, loop = 0;

        /* use the "digitizer1" array to configure the generator digitizer */
```

Continued on Next Page

```
char static *digitizer1[] =
{"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                        /* channel, 1 input port 3 */
   "VINS:LBUS:RES",                     /* reset the Local bus chip */
   "VINS:LBUS:MODE GEN",                /* set Local bus mode to GENerate */
   "VINS:LBUS:FEED 'MEM:CHAN1'"};       /* set Local bus feed */

/* use the "digitizer2" array to configure the appender digitizer */

char static *digitizer2[] =
{"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                        /* channel, 1 input port 3 */
   "VINS:LBUS:RES",                     /* reset the Local bus chip */
   "VINS:LBUS:MODE APP",                /* set Local bus mode to APPend */
   "VINS:LBUS:FEED 'MEM:CHAN1'"};       /* set Local bus feed */

/* use the "memory" array to configure the memory card */

char static *memory[] =
{"FORM:DATA PACK",              /* set packed data format */
 "TRAC:DEL:ALL",                /* delete all readings on memory card */
 "TRAC:DEF SET1, 40",           /* store readings (40 bytes) in "SET1" */
 "VINS:LBUS:RES",                       /* reset the Local bus chip */
 "VINS:LBUS:MODE CONS",         /* set Local bus mode to consume */
 "STAT:OPC:INIT OFF"};          /* execute *OPC? after INIT is parsed */

/* Execute each command in "digitizer1" using a loop */

length = (sizeof(digitizer1) / sizeof(char*));

for (loop = 0; loop < length; loop++)
   {
   IOOUTPUTS(ADDR_G, digitizer1[loop], strlen(digitizer1[loop]));
   }

/* Execute each command in "digitizer2" using a loop */

length = (sizeof(digitizer2) / sizeof(char*));

for (loop = 0; loop < length; loop++)
   {
   IOOUTPUTS(ADDR_A, digitizer2[loop], strlen(digitizer2[loop]));
   }
```

Continued on Next Page

```
        /* Execute each command in "memory" */

        length = (sizeof(memory) / sizeof(char*));

        for (loop = 0; loop < length; loop++)
            {
            IOOUTPUTS(ADDR_MEM, memory[loop], strlen(memory[loop]));
            }

        /* check for digitizer and memory card configuration errors */

        check_error("digitizer1", ADDR_G);

        check_error("digitizer2", ADDR_A);

        check_error("memory", ADDR_MEM);
}

/*****************************************************************************/
void initiate(void)
{
        int    i = 0, readings = 20, swap = 0, bytes = 0, length = 1, *rdgs;
        float rdy;
        char lf_remove[1];

        /* dynamically allocate memory for readings */

        rdgs = malloc(20 * sizeof(float));      /* allocate computer memory for reading storage */
        swap = sizeof(int);                     /* each reading in memory is two bytes */
        bytes = 20 * swap;                      /* read 40 bytes */

        IOOUTPUTS(ADDR_MEM, "INIT", 4);             /* initiate the memory card */
        IOOUTPUTS(ADDR_MEM, "*OPC?", 5);   /* wait for INIT to parse before continuing */
        IOENTER(ADDR_MEM, &rdy);            /* enter *OPC? response from memory card */

        IOOUTPUTS(ADDR_A, "INIT", 4);       /* initiate the appender digitizer */

        IOOUTPUTS(ADDR_G, "INIT", 4);       /* initiate the generator digitizer */
        IOOUTPUTS(ADDR_G, "*OPC?", 5);          /* allow the readings to complete before */
                                            /* retrieving them from the memory card */
        IOENTER(ADDR_G, &rdy);              /* enter *OPC? response from the digitizer */

        IOOUTPUTS(ADDR_MEM, "TRAC:DATA? SET1", 15);   /* retrieve readings from memory
card */
        IOENTERAB(ADDR_MEM, rdgs, &bytes, swap);   /* enter readings and remove block header
*/
```

Continued on Next Page

```
        /* remove line feed which trails the last data byte */

        IOENTERS(ADDR_MEM, lf_remove, &length);

        /* convert and display the readings; the generator digitizer readings */
        /* occur first, followed by the appender digitizer readings */

        for (i = 0; i < readings; i++)
        {
            rdgs[i] /= 16;
            if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
                    printf("\nReading overrange");
            else
                    printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0025));

        }

        free(rdgs);
}


/*****************************************************************************/
void rst_clr(long address)
{
            /* Reset and clear the instruments */

        IOOUTPUTS(address, "*RST;*CLS", 9);

}

/*****************************************************************************/
void check_error(char *array, long address)
{
        char      into[161];
        int length = 160;

        IOOUTPUTS(address, "SYST:ERR?", 9);          /* Query error register */
        IOENTERS(address, into, &length);            /* Enter error message */

        if (atoi(into) != 0)                         /* Determine if error is present */
                                                     /* If errors present, print    and exit */
        {
```

Continued on Next Page

```
        while (atoi(into) != 0)
        {
                printf("Error %s in %s\n\n", into, array);
                length =160;
                IOOUTPUTS(address, "SYST:ERR?", 9);
                IOENTERS(address, into, &length);
        }
        exit(1);
    }
}
```

Comments

1. **GENerator Digitizer Configuration.** Channel 1 of the GENerator digitizer is CONFigured for 10 readings on the 5V range.

Before setting the digitizer's Local bus configuration, the Local bus chip is reset. Next, the Local bus mode is set to GENerate and the feed (data source) is set to MEMory:CHANnel1.

2. **APPender Digitizer Configuration.** Channel 1 of the APPender digitizer is CONFigured for 10 readings on the 5V range.

For all digitizer's doing Local bus transfers, the Local bus chip must be reset first. Next, the Local bus mode is set to APPend and the feed (data source) is set to MEMory:CHANnel1.

3. **Pre- and Post-Arm Readings.** When the Local bus data source is MEMory: ..., the readings are transferred over the Local bus from digitizer memory rather than directly from the A/D. In this case, pre- and post-arm readings are allowed prior to the Local bus transfer.

4. **Initiating the Digitizers.** After the memory card is configured, the card is INITiated first so that it is ready to receive the digitizer readings. In a configuration with multiple digitizers in the APPend mode, the APPender digitizer(s) is (are) INITiated next. Because CONFigure sets the arm source to IMMediate, INITiating the (APPender) digitizer causes it to take its readings. After the APPender is INITiated the GENerator digitizer is INITiated. *OPC? is used to allow the GENerator readings to complete and be transferred before they are retrieved from the memory card.

Note that the APPender digitizers must have finished taking data before the GENerator is finished. All APPenders must be ready to pipeline the GENerator's data at the time the data is sent.

5. **Reading Sequence and Format.** When this program executes, the readings are transferred to the memory card and later displayed in the following sequence:

GENerator digitizer reading 1
GENerator digitizer reading n
.
.
APPender digitizer reading 1
APPender digitizer reading n

The memory card was set up to store the readings in the digitizer's packed data format. The packed readings are signed, 16-bit numbers preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header. Packed readings are always numbers between -1.0230 (-2046) and +1.0235 (2047). To convert the readings to voltages, each reading is divided by 16 to remove the data label bits (0 - 3), and is multipled by 0.0025 (the reading resolution for the 5V range).

6. **Additional Information.** Additional information on Local bus operation and on the Local bus commands can be found in Chapter 3 - "Understanding the Digitizer", and in Chapter 4 - "Command Reference."

LBUSAUTO.C

This program transfers readings over the Local bus from two digitizer A/Ds to the HP E1488 memory card. The program shows how the digitizers are used in the Local bus GENerate and INSert modes. The program is similar to the previous program; however, rather than INITiate each digitizer individually, the INSerter digitizer INITiates the GENerator digitizer once the INSerter digitizer readings are complete.

```
/* LBUSAUTO.C - This program demonstrates how to transfer readings over the */
/* Local bus from two digitizer A/Ds to the HP E1488 memory card. The leftmost */
/* digitizer is set to the GENerate mode and the inner digitizer is set to the */
/* INSert mode. The generator digitizer is armed from the TTLT0 trigger line */
/* when the inserter digitizer has completed its measurements. */

        /* Include the following header files */

#include    <stdio.h>
#include    <stdlib.h>
#include    <string.h>
#include    <malloc.h>
#include    <cfunc.h>       /* This file is from the HP-IB Command Library Disk */
```

Continued on Next Page

```
#define    ADDR_G 70905L          /* I/O path from the PC to the generator digitizer */
#define ADDR_I 70906L             /* I/O path from the PC to the inserter digitizer */
#define ADDR_MEM 70903L           /* I/O path from the PC to the memory card */
#define CMD_MOD 70900L            /* I/O path from the PC to the Command Module */

/* Function Prototypes */

void rst_clr(long address);
long get_base_addr(void);
void configure(void);
void initiate(long base_addr);
void check_error(char *func_tion, long address);

/*****************************************************************************/
void main(void)                   /* run the program */
{
      long base_addr;             /* variable for A24 base address */
      clrscr();

      rst_clr(ADDR_G);            /* reset the generator digitizer */
      rst_clr(ADDR_I);            /* reset the inserter digitizer */
      rst_clr(ADDR_MEM);          /* reset memory card */

      base_addr = get_base_addr(); /* get digitizer A24 base address */

      configure();                          /* configure the digitizers and memory card*/

      initiate(base_addr);                        /* initiate the digitizers and memory card; */
                                            /* retrieve the readings from the memory card */
}

/*****************************************************************************/
void configure(void)
{
      int    length = 0, loop = 0;

      /* use the "digitizer1" array to configure the generator digitizer */

      char static *digitizer1[] =
      {"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                              /* channel, 1 input port 3 */
         "ARM:STAR:SOUR TTLT0",               /* set arm source */
         "ARM:STAR:DEL 50E-6",                /* set arm delay */
         "VINS:LBUS:RES",                     /* reset the Local bus chip */
         "VINS:LBUS:MODE GEN",                /* set Local bus mode to GENerate */
         "VINS:LBUS:FEED 'CONV:CHAN1'"};      /* set Local bus feed */
```

Continued on Next Page

```
/* use the "digitizer2" array to configure the inserter digitizer */

char static *digitizer2[] =
{"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                        /* channel, 1 input port 3 */
   "OUTP:TTLT0:FEED 'READY'",           /* feed ready signal to next digitizer */
   "OUTP:TTLT0:STAT ON",                /* enable ready signal feed */
   "VINS:LBUS:RES",                     /* reset the Local bus chip */
   "VINS:LBUS:MODE INS",                /* set Local bus mode to INSert */
   "VINS:LBUS:FEED 'CONV:CHAN1'"};      /* set Local bus feed */

/* use the "memory" array to configure the memory card */

char static *memory[] =
{"FORM:DATA PACK",              /* set packed data format */
 "TRAC:DEL:ALL",                /* delete all readings on memory card */
 "TRAC:DEF SET1, 40",           /* store readings (40 bytes) in "SET1" */
 "VINS:LBUS:RES",                       /* reset the Local bus chip */
 "VINS:LBUS:MODE CONS",         /* set Local bus mode to consume */
 "STAT:OPC:INIT OFF"};          /* execute *OPC? after INIT is parsed */

/* Execute each command in "digitizer1" using a loop */

length = (sizeof(digitizer1) / sizeof(char*));

for (loop = 0; loop < length; loop++)
   {
   IOOUTPUTS(ADDR_G, digitizer1[loop], strlen(digitizer1[loop]));
   }

/* Execute each command in digitizer2 using a loop */

length = (sizeof(digitizer2) / sizeof(char*));

for (loop = 0; loop < length; loop++)
   {
   IOOUTPUTS(ADDR_I, digitizer2[loop], strlen(digitizer2[loop]));
   }

/* Execute each command in "memory" */

length = (sizeof(memory) / sizeof(char*));
```

Continued on Next Page

```
        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR_MEM, memory[loop], strlen(memory[loop]));
           }

        /* check for digitizer configuration errors */

        check_error("digitizer1", ADDR_G);

        check_error("digitizer2", ADDR_I);

        check_error("memory", ADDR_MEM);
}

/*****************************************************************************/
void initiate(long base_addr)
{
        int    i = 0, readings = 20, swap = 0, bytes = 0, length = 1, *rdgs;
        int    bit_reg = 0;
        float rdy, bit_pat = 0;
        char lf_remove[1], command[80];

        /* dynamically allocate memory for readings */

        rdgs = malloc(20 * sizeof(float));      /* allocate computer memory for reading storage */
        swap = sizeof(int);                     /* each reading in memory is two bytes */
        bytes = 20 * swap;                      /* read 40 bytes */

        /* create DIAG:PEEK? command which reads the generator digitizer's */
        /* arm source register */

        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x49,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);

        /* retain register settings, set arm source 1 slope to positive */
        bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xF7);
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x49,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        IOOUTPUTS(ADDR_MEM, "INIT", 4);             /* initiate the memory card */
        IOOUTPUTS(ADDR_MEM, "*OPC?", 5);    /* wait for INIT to parse before continuing */
        IOENTER(ADDR_MEM, &rdy);            /* enter *OPC? response from memory card */

        IOOUTPUTS(ADDR_G, "INIT", 4);       /* initiate the generator digitizer */
```

Continued on Next Page

```
        IOOUTPUTS(ADDR_I, "INIT", 4);              /* initiate the inserter digitizer */
        IOOUTPUTS(ADDR_G, "*OPC?",5);                  /* wait for generator digitizer to finish */
        IOENTER(ADDR_G, &rdy);

        IOOUTPUTS(ADDR_MEM, "TRAC:DATA? SET1", 15); /* retrieve readings from memory card
*/
        IOENTERAB(ADDR_MEM, rdgs, &bytes, swap);    /* enter readings and remove block header
*/

        /* remove line feed which trails the last data byte */

        IOENTERS(ADDR_MEM, lf_remove, &length);

        /* convert and display readings; the inserter digitizer readings */
        /* occur first, followed by the generator digitizer readings */

        for (i = 0; i < readings; i++)
        {
            rdgs[i] /= 16;
            if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
                    printf("\nReading overrange");
            else
                    printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0025));

        }

        free(rdgs);
}


/*****************************************************************************/
long get_base_addr(void)
{
        /* base address of generator digitizer's (A24) offset register in A16 */
        /* address space */

        long base_addr = (0x1FC000 + (40 * 64)) + 6; /* generator digitizer logical address is 48 */

         float a24offst;        /* A24 offset from A16 offset register */

        char rd_addr[80];    /* command string variable */

        /* Create the command string which reads the A24 base address from the offset register */
        sprintf(rd_addr, "DIAG:PEEK? %ld, %d", base_addr,16);

            /* Send DIAG:PEEK? command */
        IOOUTPUTS(CMD_MOD, rd_addr, strlen(rd_addr));
```

Continued on Next Page

```
        /* Read value from offset register */
    IOENTER(CMD_MOD, &a24offst);

        /* Multiply offset value by 256 for 24-bit address value */
    a24offst *= 256.;

    return (long)a24offst;
}
/*****************************************************************************/
void rst_clr(long address)
{
        /* Reset and clear the instruments */

    IOOUTPUTS(address, "*RST;*CLS", 9);

}

/*****************************************************************************/
void check_error(char *array, long address)
{
    char    into[161];
    int length = 160;

    IOOUTPUTS(address, "SYST:ERR?", 9);        /* Query error register */
    IOENTERS(address, into, &length);          /* Enter error message */

    if (atoi(into) != 0)                       /* Determine if error is present */
                                               /* If errors present, print   and exit */
    {
        while (atoi(into) != 0)
        {
            printf("Error %s in %s\n\n", into, array);
            length = 160;
            IOOUTPUTS(address, "SYST:ERR?", 9);
            IOENTERS(address, into, &length);
        }
        exit(1);
    }
}
```

Comments

1. **GENerator Digitizer Configuration.** Channel 1 of the GENerator digitizer is CONFigured for 10 readings on the 5V range. The arm source is set to TTLT0. This VXI backplane trigger line is controlled by the INSerter digitizer which feeds its READy signal to arm the GENerator digitizer **after** its readings are complete. An arm delay of 50 µs is specified to allow for the INSerter digitizer to switch to the pipeline mode after its readings are complete (see the "Insert" mode description under "Local Bus Modes" in Chapter 3).

The INSerter digitizer's READy signal goes high when the readings are complete and the digitizer enters the idle state. In order for the GENerator digitizer to arm on the low-to-high transition, the GENerator digitizer must be set to accept a positive slope. This is done by writing to the digitizer's Arm Source register and setting the slope bit from negative (1) to positive (0).

The digitizer's Local bus configuration begins by resetting the Local bus chip. The Local bus mode is set to GENerate and the feed (data source) is set to CONVerter:CHANnel1.

2. **INSerter Digitizer Configuration.** Like the GENerator digitizer, the INSerter digitizer is CONFigured for 10 readings on the 5V range. The INSerter digitizer's arm source and trigger source are INTernal. The INSerter digitizer transfers (feeds) its READy signal to the GENerator digitizer over the VXI backplane TTLT0 trigger line.

Again, the digitizer's Local bus configuration begins by resetting the Local bus chip. The Local bus mode is set to INSert and the feed (data source) is set to CONVerter:CHANnel1.

3. **Digitizer Sample Rates.** The maximum Local bus transfer rate is 80 MBytes/second which is equivalent to 40 MSamples/second. The sample rate for both digitizers as set by the CONFigure command is 50 ns (20 MHz). Thus, data is transferred from the two digitizers at a rate of 40 MSamples (80 MBytes)/second.

4. **Initiating the Digitizers.** After the memory card is configured, the card is INITiated first so that it is ready to receive the digitizer readings. In a configuration with multiple digitizers in the INSert mode, the GENerator digitizer is INITiated next so that it is waiting for an arm signal to begin taking measurements. When the INSerter is INITiated it takes its readings and then arms the GENerator digitizer. *OPC? is used to allow the GENerator readings to complete and be transferred before they are retrieved from the memory card.

5. **Reading Sequence and Format.** When this program executes, the readings are transferred to the memory card and later displayed in the following sequence:

INSerter digitizer reading 1
INSerter digitizer reading n
.
.
GENerator digitizer reading 1
GENerator digitizer reading n

The memory card stores the readings in the digitizer's packed data format. Packed readings are signed, 16-bit numbers preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header. Packed readings are always numbers between -1.0230 (-2046) and +1.0235 (2047). To convert the readings to voltages, each reading is divided by 16 to remove the data label bits (0 - 3), and is multipled by 0.0025 which is the reading resolution for the 5V range.

6. **Additional Information.** Additional information on Local bus operation and on the Local bus commands can be found in Chapter 3 - "Understanding the Digitizer", and in Chapter 4 - "Command Reference." Information on the digitizer's Arm Source register can be found in Appendix C - "Register Programming."

Using the Digitizer Status Registers

This program demonstrates:

- how to use the condition register, transition filter, enable register, and status byte to determine when events monitored by the condition register occur.

One of the conditions monitored by the operation status group's condition register is when the digitizer receives an arm signal. When armed, a high-to-low transition of the wait-for-arm bit (bit 6) in the condition register occurs. In this program, the digitizer is set to arm when the input signal on the differential input reaches 3V. When the level is reached and the arm occurs, the user is notified that the digitizer is armed and is ready to sample.

STATUS.C

```
CONF1:ARR:VOLT (1),5,(@3)                    /* set 1 reading, 5V range */
    ARM:STAR:SOUR INT1                  /* arm on input signal level */
    ARM:STAR:SLOP1 POS                  /* arm on positive-going signal */
    ARM:STAR:LEV1:POS 3                 /* arm at 3V level */
    TRIG:STAR:SOUR HOLD                      /* set trigger source to hold */
    STAT:OPER:PTR 0                          /* prevent any positive transitions */
                                        /* from causing summary bit to set OPER */
    STAT:OPER:NTR 64                    /* set event register bit on */
                                        /* negative transition of bit 6 */
    STAT:OPER:ENAB 64                   /* enable summary bit to set OPER */
                                        /* bit in status register */
    INIT                                /* put digitizer in wait-for-arm state */

        print message "waiting for input signal to reach arm level"

        loop
            *STB?                       /* read status byte */
            enter byte
            is byte < 128. ?            /* check if OPER bit in status byte is cleared (0) */
        end loop

        print message "digitizer armed; press Enter (return) to trigger the reading"

    TRIG:IMM        /* issue a single trigger */
FETC1?                      /* retrieve reading */
```

Comments

1. **Setting the Transition Filter.** When the digitizer is INITiated, a low-to-high transition of bit 6 in the condition register occurs indicating the digitizer is in the wait-for-arm state. When an arm is received, a high-to-low transition of bit 6 occurs. The power-on/reset setting of the transition filter allows the positive transition to be recognized and the negative transition to be ignored. Because the application is interested in the high-to-low transition, the transition filter is set for a negative transition of bit 6 (STATus:OPERation:NTRansition 64), and to ignore the positive transition (STATus:OPERation:PTRansition 0).

2. **Setting the Enable Register.** When the high-to-low transition occurs, the "event" is recorded by setting (to ’1’) a corresponding bit in the event register. (Event register bits are set from low-to-high regardless of the transition recognized by the transition filter.)

In order for the controller/computer to know that the event has occurred, the enable register is set such that the arm event sets a bit in the status byte (STATus:OPERation:ENABle 64).

3. **Reading the Status Byte.** Once the program initiates the digitizer (INIT command), the program continually monitors the digitizer status byte using *STB?. When the arm event occurs, bit 7 in the status byte is set. (Status byte bits are also set from low-to-high regardless of the transition recognized by the transition filter.) This signals the controller/computer that the event occurred. At this point, the program exits the loop and then prompts the user to press ’Enter’ which then triggers the digitizer (TRIGger:IMMediate).

Additional information on the digitizer status groups and on using the registers in the groups can be found in Chapter 3 - "Understanding the Digitizer".

Chapter 3
Understanding the Digitizer

Chapter Contents

This chapter contains the HP E1429 Digitizer description of operation. Where applicable, the chapter relates the digitizer's SCPI commands to the digitizer hardware they control. The main sections of the chapter include:

HP E1429 Digitizer Block Diagram

The HP E1429 Digitizer block diagram is shown in Figure 3-1. Channels 1 and 2 have their own input section and their own 12-bit, 20 MSample/second analog-to-digital (A/D) converter. The message interface, register interface, trigger/timebase section, and memory are common to both channels.

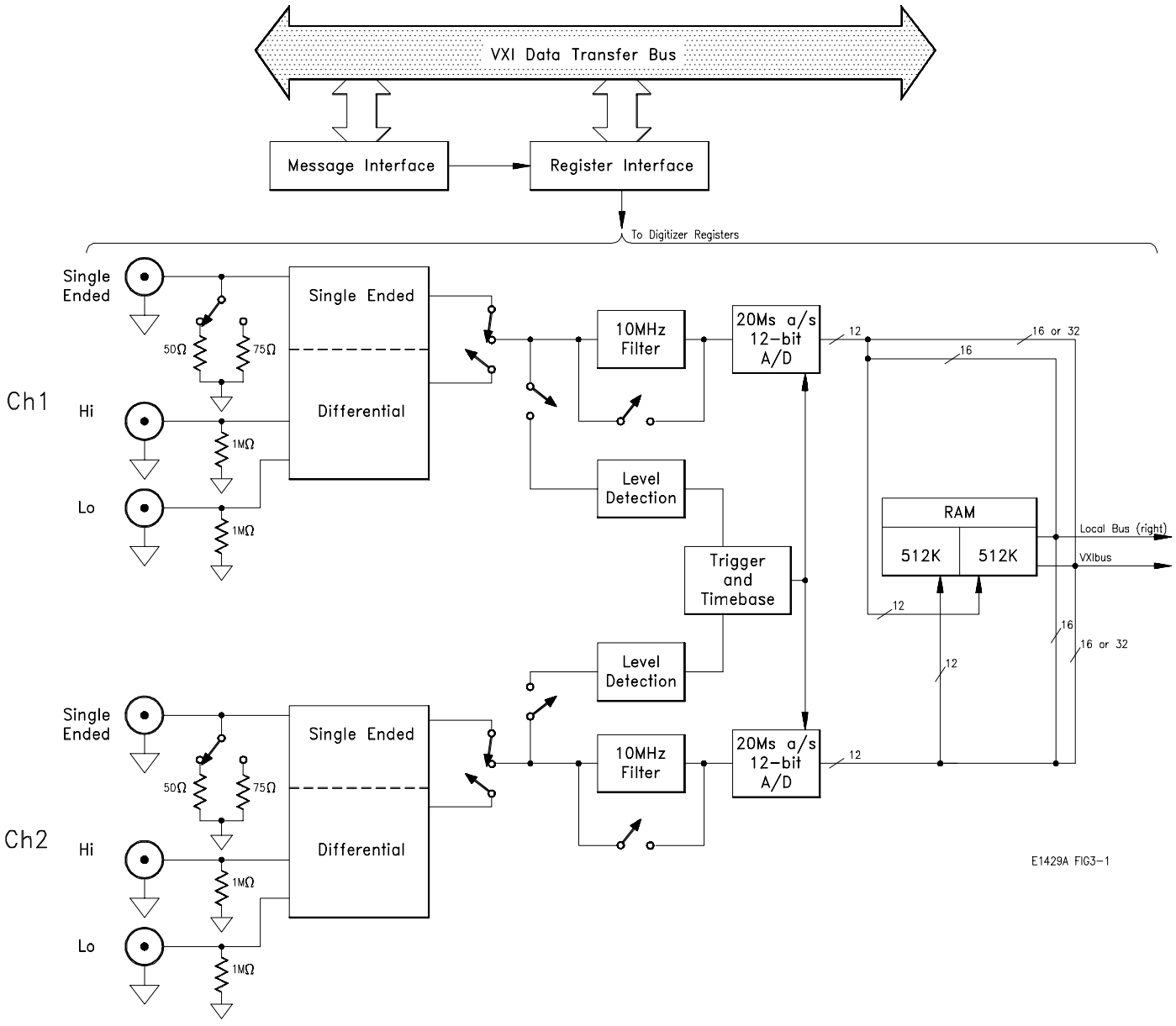


Figure 3-1. HP E1429 Digitizer Block Diagram

The Message and Register Interfaces

The HP E1429 digitizer can be programmed as a message-based device or as a register-based device (Appendix C). When the digitizer functions as a message-based device, the processor within the message interface (Figure 3-1) converts SCPI (ASCII) command strings to register reads and writes through the register interface. When the digitizer functions as a register-based device, command opcodes are written directly to the digitizer registers from the register interface.

Digitizer Command Paths

Figure 3-2 shows the command paths used when programming the digitizer as a message-based device or as a register-based device.

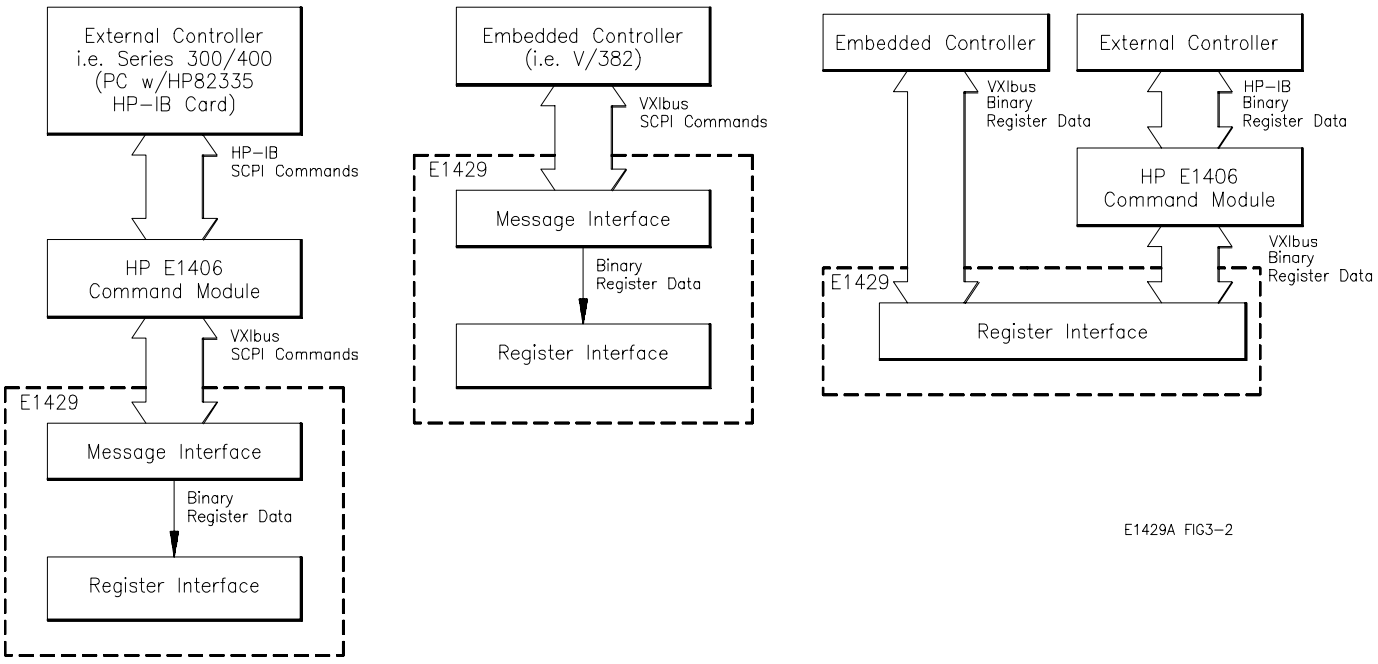


Figure 3-2. HP E1429 Digitizer Command Paths

The Digitizer Input Section

The HP E1429 2-channel digitizer has single-ended and differential input ports on each channel. Each input port has a BNC connector and is DC coupled. A detailed block diagram of the input section is shown in Figure 3-3. Only one channel is shown since both are identical.

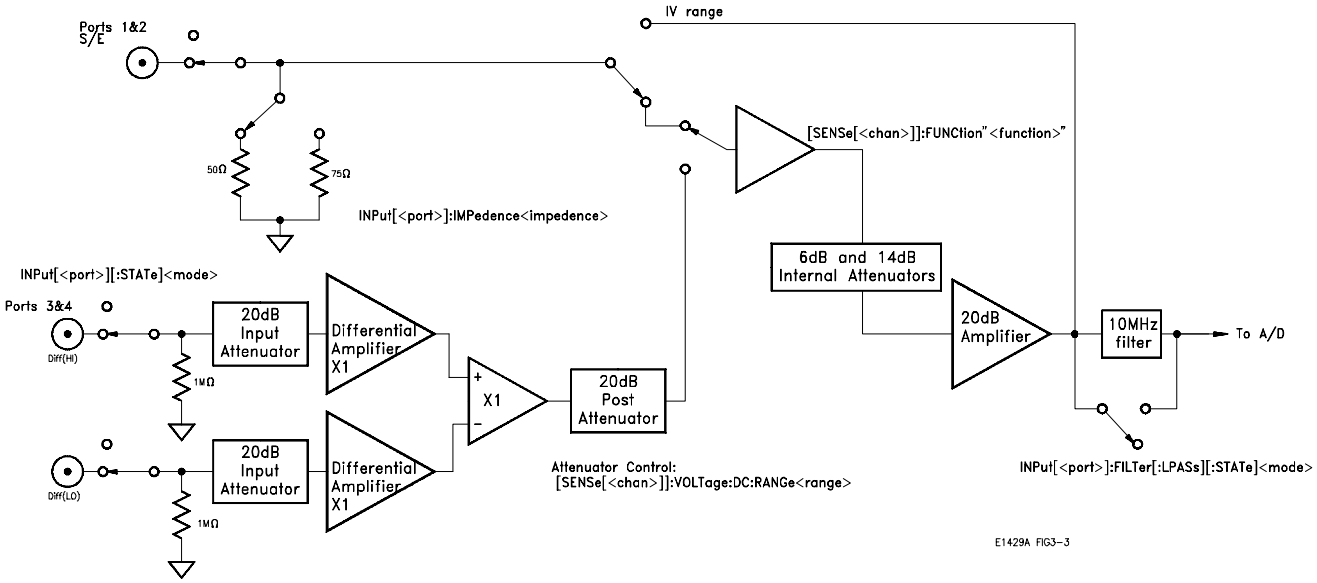


Figure 3-3. HP E1429 Digitizer Input Section

SCPI Command Control

The digitizer's input section including the input state, input impedance, and low-pass filtering is controlled by the SCPI INPut subsystem. The input port is selected by the SENSe[<chan >]:FUNCtion subsystem. The input signal range is controlled by the SENSe[<chan >]:VOLTage subsystem. The commands in these subsystems are listed below.

INPUT[<*port* >]
:FILTer
    [:LPASs]
        [:STATE] <*mode* >
:IMPedance <*impedance* >
[:STATe] <*mode* >

```
[SENSe[<chan >]]
:FUNCtion "<function >"

[SENSe[<chan >]]
:VOLTage
    [:DC]
        :RANGE <range >
```

Note Each command in these subsystems is covered in detail in Chapter 4 - "Command Reference".

Each input parameter setting is unique to each port. The settings are "remembered" so that changes made to one port do not affect the changes made previously to the other port. However, the signal range set (or changed) for one port applies to the other port as well.

Selecting the Input Port

The digitizer input channels are numbered 1 and 2. The input ports on each channel are:

- channel 1
  - port 1 - singled-ended input
  - port 3 - differential input
- channel 2
  - port 2 - single-ended input
  - port 4 - differential input

The active input port is specified by the (@<input port >) parameter of the CONFigure and MEASure commands, and by [<port >] in the other digitizer commands. In those commands, [<port >] is optional as indicated by the brackets []. If a port number is **not** specified, port 1 (channel 1) is assumed.

The active input port is also selected with the command:

[SENSe[<*chan* >]]:FUNCtion "<*function* >"

The "<function >" settings are:

VOLTage[:DC]1port 1 (channel 1)
VOLTage[:DC]2port 2 (channel 2)
VOLTage[:DC]3port 3 (channel 1)
VOLTage[:DC]4port 4 (channel 2)

Readings are taken on only one input port per channel.

Enabling the Input Ports

Each input port has a relay that is used to enable and disable the input. This relay is controlled by the command:

INPut[<*port* >][:STATe] <*mode* >

The <*mode* > settings are:

ON - the input is enabled.
OFF - the input is disabled. For the single ended input, the port is set to a high impedance.

At power-on or following a reset, each input port is enabled (<mode > is ON). However, the port used for the measurement is specified by the (@<*input port* >) parameter of CONFigure/MEASure, or by [SENSe[<chan >]]:FUNCtion.

Setting the Input Impedance

The impedance of the single-ended input port can be set to 50Ω or 75Ω. The impedance is set with the command:

INPut[<*port* >]:IMPedance <*impedance* >

The <*impedance* > settings are:

50 - sets single ended input impedance to 50Ω
75 - sets single ended input impedance to 75Ω

At power-on or following a reset, the impedance is set to 50Ω. The impedance of the differential input ports is 1 MΩ and is not programmable.

The Inverting and Non-inverting Differential Input Ports

The non-inverting (HI) and inverting (LO) differential input ports can be used singly (the non-used port should be grounded), or together such that the difference (algebraic sum) of two signals is supplied to the digitizer. If a single input is applied to the inverting (LO) port, the reading is inverted.

Enabling the 10 MHz Input Filter

Each digitizer channel has a 10 MHz, 2-pole bessel filter that can be switched into the signal path of the single-ended or differential input. The filter is enabled/disabled with the command:

INPut[<*port* >]:FILTer[:LPASs][:STATe] <*mode* >

The <*mode* > settings are:

ON - 10 MHz filter is switched to the signal path.
OFF - 10 MHz filter is removed from the signal path.

Enabling the filter reduces the noise on the input signal. Disabling the filter allows sub-sampling applications over the digitizer's 50 MHz bandwidth.

The filter mode of one input port is independent of the filter mode of the channel's other port. At power-on or following a reset, the filter is disabled on all input ports. CONFigure and MEASure enable the filter on the specified port.

Setting the Signal Range

The digitizer's input signal range is specified in terms of an expected reading value or as an explicit range value. The commands used for each method are as follows:

MEASure[<*chan* >]:ARRay[:VOLTage][:DC]? (<*size* >)
[,**<*expected value* >**[,<*resolution* >]] [,(@<*input port* >)]

CONFigure[<*chan* >]:ARRay[:VOLTage][:DC] (<*size* >)
[,**<*expected value* >**[,<*resolution* >]] [,(@<*input port* >)]

[SENSe[<*chan* >]]:VOLTage[:DC]:RANGe <*range* >

Changing Ranges

With MEASure or CONFigure, if the <*expected value* > specified is within 98% of the maximum signal that can be measured on the nearest measurement range (Table 3-1), that range is selected. If the <*expected value* > is greater than 98%, the next higher range is selected. For example, if the <*expected value* > is 5.1V, the 10V range is selected since 5.1V is greater than 98% of 5.1175V.

Specifying a signal range with the SENSe:VOLTage command that is within a given measurement range (Table 3-1), sets that range. If a signal range is specified that is outside a given measurement range, the next higher range is selected. For example, if a <*range* > of .52 is specified, the 1V range is selected since .52 is outside the -0.5115 to 0.51175 measurement range.

The Digitizer Attenuators

The input section of the HP E1429 digitizer has three sets of attenuators (Figure 3-3):

20 dB input attenuator (differential inputs only)
20 dB post attenuator (differential inputs only)
6 dB and 14 dB internal attenuators

The attenuators used are based on the expected value or range specified. Table 3-1 shows the attenuators used as a function of the expected value and range.

Table 3-1. HP E1429 Digitizer Measurement Ranges

Input Port	<*expected value*> (MEAS/CONF)	Range setting <*range*>	Measurement Range	20 dB Input Attenuator	20 dB Post Attenuator	Internal Attenuators 6 dB	14 dB
1,2,3,4	±0.1	0.10235	-0.10230 to 0.10235	off	off	off	off
1,2,3,4	±0.2	0.2047	-0.2046 to 0.2047	off	off	on	off
1,2,3,4	±0.5	0.51175	-0.5115 to 0.51175	off	off	off	on
1,2,3,4	±1.0	1.0235	-1.0230 to 1.0235	off	on	off	off
3,4	±2.0	2.047	-2.0460 to 2.0470	off	on	on	off
3,4	±5.0	5.1175	-5.115 to 5.1175	off	on	off	on
3,4	±10.0	10.235	-10.230 to 10.235	on	on	off	off
3,4	±20.0	20.470	-20.460 to 20.470	on	on	on	off
3,4	±50.0	51.175	-51.150 to 51.175	on	on	off	on
3,4	±100.0	102.35	-102.30 to 102.35	on	on	on	on

Using the Single-Ended Input 1V Range

When taking readings of low-level signals on the single-ended input, selecting the 1V range ensures the most accurate readings. This is achieved through a path directly to the A/D, which bypasses the attenuators and amplifier which can add noise and distortion to the signal (Figure 3-3).

The single-ended input 1V range can be set by specifying an <*expected value* > of 1V in the MEASure or CONFigure command, or by specifying a range of 1V in the SENSe:VOLTage command:

```
SENS:VOLT:RANG 1
```

Arming and Triggering

The HP E1429 digitizer uses the SCPI Arm-Trigger configuration shown in Figure 3-4. This section describes the Arm-Trigger state sequence and identifies the different ways to arm and trigger the digitizer.

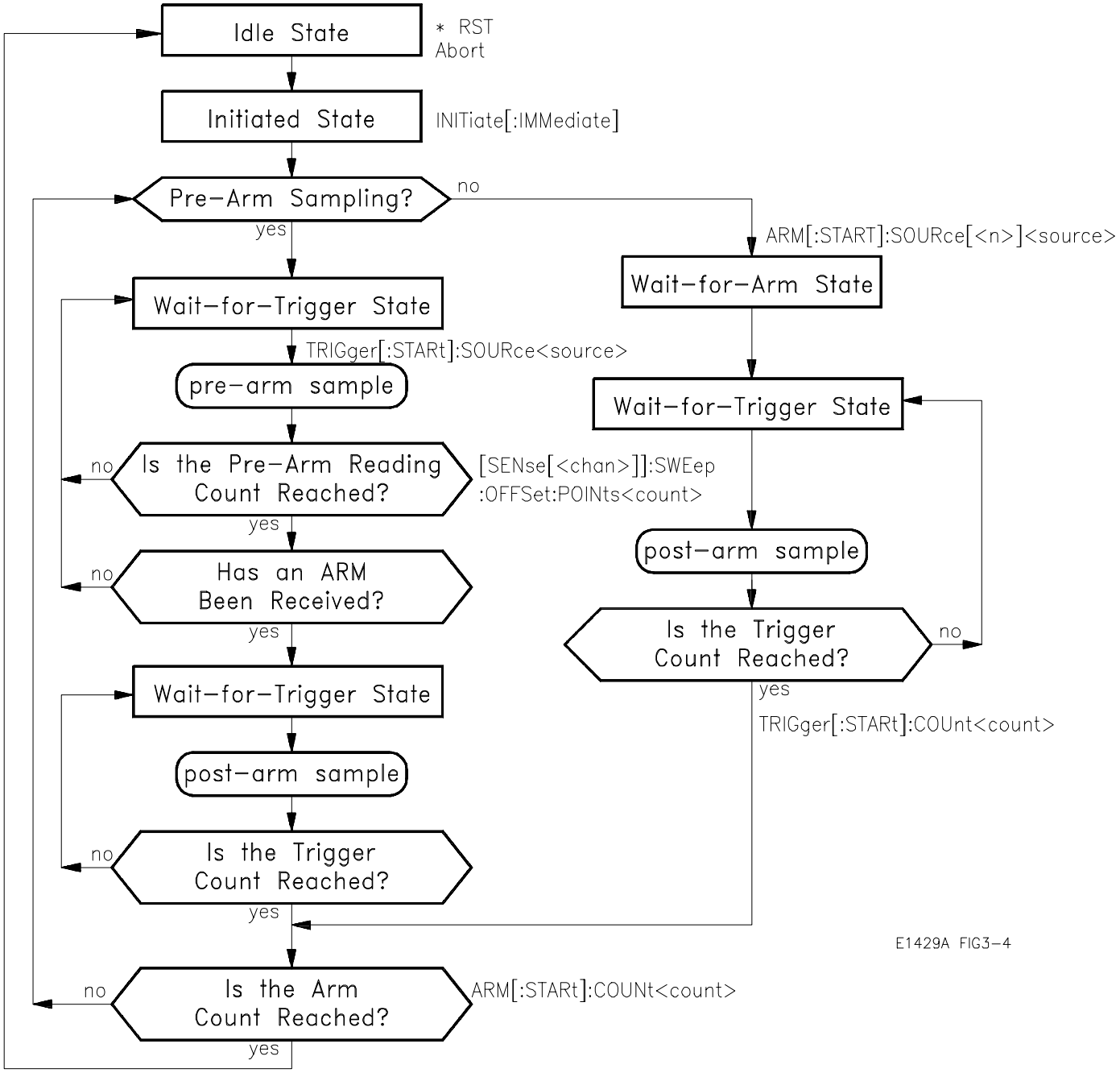


Figure 3-4. HP E1429 Digitizer State Diagram

The ARM-TRIG State Diagram

The state diagram of Figure 3-4 shows that the digitizer operates within four states: idle, initiated, wait-for-arm, and wait-for-trigger. The path through the states depends upon whether pre-arm sampling is part of the measurement configuration.

Both paths begin with the digitizer in the idle state. The digitizer enters the idle state when power is applied, or following a reset or an ABORt. This is the state in which the digitizer is configured using CONFigure and its low-level commands.

Note

When configuring the digitizer's arming and triggering hardware the configuration applies to **both** digitizer channels. Thus, both channels enter the same Arm-Trigger state at the same time. In arming and triggering commands where a channel can be specified such as

SENSe[<*chan* >]:SWEep:OFFSet:POINts <*count* >

the configuration of the last channel specified (or implied) overrides the previous configuration of both channels.

The Pre-Arm Path

The digitizer is capable of pre-arm and post-arm sampling. If pre-arm samples (readings) have been specified (SENSe[<chan >]:SWEep:OFFSet:POINts <count >) in addition to post-arm samples, the digitizer moves to the wait-for-trigger state when INITiate[:IMMediate] is executed and begins sampling. The digitizer continues to (pre-arm) sample until the pre-arm reading count is reached **and** until the digitizer receives an arm signal. When the arm is received, the digitizer begins post-arm sampling as triggers are received. The digitizer continues to (post-arm) sample until the total number of samples (pre- and post-arm) specified by TRIGger[:STARt]:COUNt <count > is reached. Once the trigger count is reached, the digitizer determines if the arm count has been reached. If it has not, the digitzer remains initiated and repeats the path until the arm count is reached.

The Post-Arm Path

When only post-arm samples are specified, the digitizer moves to the wait-for-arm state when INITiate[:IMMediate] is executed. The digitizer moves to the wait-for-trigger state when an arm signal is received. The digitizer remains in the wait-for-trigger state until the number of samples specified by TRIGger[:STARt]:COUNt <count > is reached. Once the trigger count is reached, the digitizer determines if the arm count has been reached. If it has not, the digitizer remains initiated and repeats the path until the arm count is reached.

Arming the Digitizer

Before the digitizer takes a sample it must be "armed", which means it must be in the wait-for-trigger state. Figure 3-5 shows the digitizer arming and triggering sources.

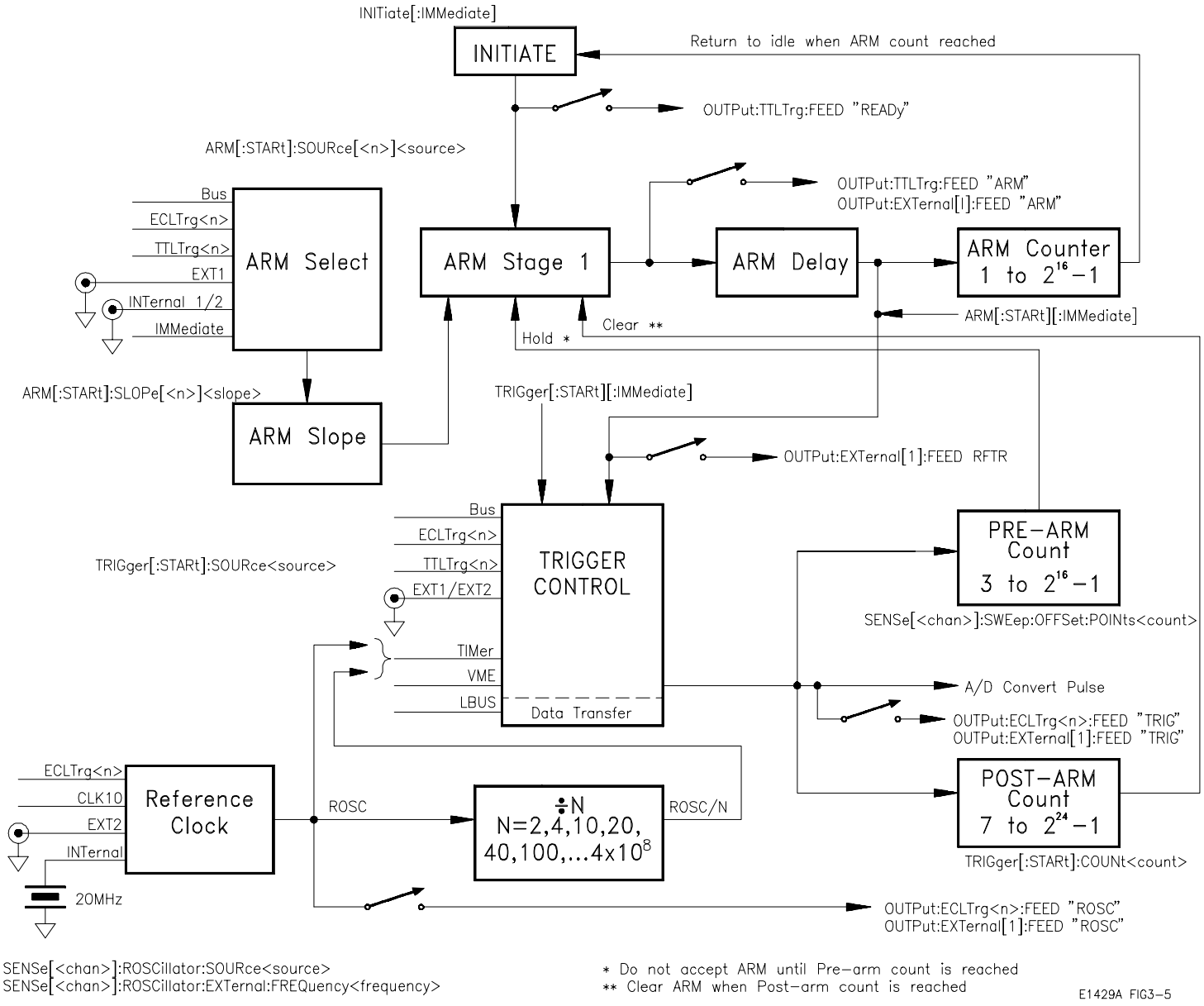


Figure 3-5. HP E1429 Digitizer Arming and Triggering

The digitizer hardware associated with arming the digitizer is controlled by the SCPI ARM subsystem. The commands in this subsystem include:

```
ARM
[:STARt|SEQuence[1]]
    :COUNt <count>
    :DELay <period>
    [:IMMediate]
    :LEVel[<chan>]
        :NEGative <voltage>
        :POSitive <voltage>
    :SLOPe[<n>] <edge>
    :SOURce[<n>] <source>
```

Note Each command in this subsystem is covered in detail in Chapter 4 - "Command Reference".

ARM Subsystem Overview

The following information summarizes each command in the ARM subsystem.

Setting the Arm Source

The arm source specifies the signal(s) which arms the digitizer. The source is specified with the command:

ARM[:STARt]:SOURce[<*n* >] <*source* >

Up to two arm sources ([<*n* >] = 1|2) can be specified. ARM:SOUR1 specifies the first source. ARM:SOUR2 specifies the second source. The first event to occur on either source arms the digitizer. The digitizer arm source(s) can be selected from the following:

- BUS - HP-IB Group Execute Trigger (GET) commmand or the *TRG common command.
- ECLTrg0/ECLTrg1 - the VXIbus ECLTrg0 or ECLTrg1 trigger lines.
- TTLTrg<n > - the VXIbus TTLTrg0 through TTLTrg7 trigger lines.
- EXTernal1 - the "Ext 1" BNC input port.
- INTernal1 - arms when the signal level on channel 1 meets the conditions specified by ARM:LEVel1.
- INTernal2 - arms when the signal level on channel 2 meets the conditions specified by ARM:LEVel2.

- HOLD - disables either ARM:SOURce1 or ARM:SOURce2. This is the reset source for ARM:SOURce2.
- IMMediate - arms the digitizer immediately after INITiate[:IMMediate] is received. This choice is only valid for ARM:SOURce1. If IMMediate is specified, HOLD must be specified for ARM:SOURce2. IMMediate is the reset value for ARM:SOURce1.

Note An active reference oscillator is required for the digitizer to recognize and accept an arm signal from any source.

Setting the Arm Signal Slope

When the arm source is INTernal1, INTernal2, or EXTernal1, the digitizer arms on a specified level of a positive-going or negative-going input signal. For these sources, the digitizer must be informed of the slope on which to arm (positive-going, negative-going, or either (for INTernal<n >)). This is done with the command:

ARM[:STARt]:SLOPe[<*n* >] <*edge* >

The <*edge* > settings are:

POSitive - arm on a positive-going edge of the input signal.
NEGative - arm on a negative-going edge of the input signal.
EITHer - arm on either edge of the input signal (INTernal<n > sources only).

The command is used with the arm source and arm level commands as shown:

```
!set arm source to INT1 or INT2
ARM[:STARt]:SOURce[<n >] <source >

!set positive or negative transition
ARM[:STARt]:SLOPe[<n >] <edge >

!set arm input voltage level
ARM[:STARt]:LEVel[<chan >]:POSitive <voltage >
or
ARM[:STARt]:LEVel[<chan >]:NEGative <voltage >
```

The arm slope command allows you to specify a positive transition, a negative transition, or either transition. When a positive transition is specified, the digitizer is armed when the increasing signal on the channel reaches the specified level. When a negative transition is specified, the digitizer is armed when the decreasing signal on the channel reaches the specified level. When "EITHer" is specified, the digitizer is armed when the input signal enters or exits the defined window. Refer to Figure 3-6.

Note

For arm source TTLTRG<n >, the digitizer arms on the negative-going edge of the signal. For arm source ECLTrg<n >, the digitizer arms on the positive-going edge. These sources are not affected by the ARM:STARt:SLOPe command.

Arm Window Boundaries

The boundaries of the window are set by the voltage levels of ARM[:STARt]:LEVel:POSitive and ARM[:STARt]:LEVel:NEGative. When ARM[:STARt]:LEVel:POSitive specifies a level that is greater than ARM[:STARt]:LEVel:NEGative, the digitizer is armed when the input signal exits the defined window. When ARM[:STARt]:LEVel:POSitive specifies a level that is less than ARM[:STARt]:LEVel:NEGative, the digitizer is armed when the input signal enters the defined window.

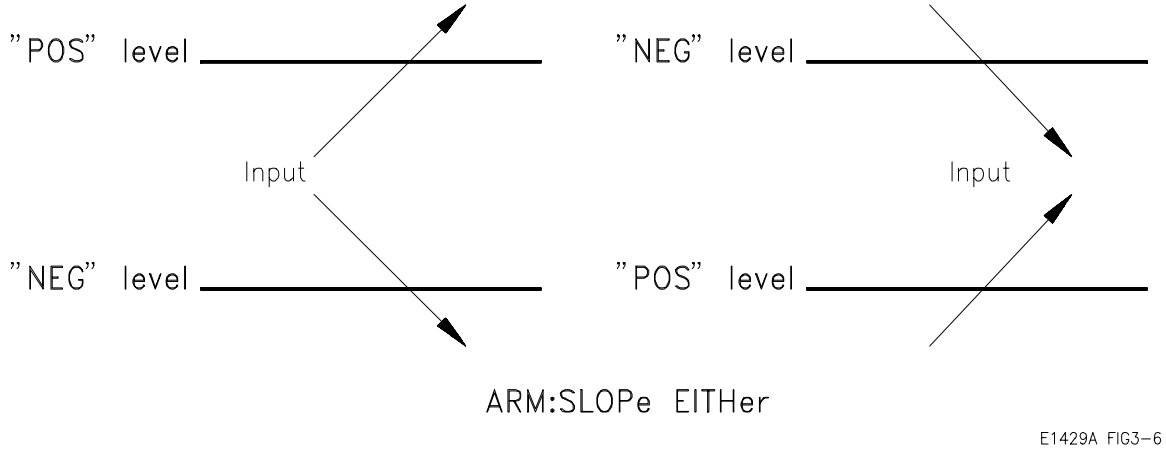


Figure 3-6. Arm Window Boundaries

Setting the Arm Level

When the arm source is INTernal1 or INTernal2, the level of the input signal at which the digitizer becomes armed is set with the commands:

ARM[:STARt]:LEVEL[<*n* >]:NEGative <*voltage* >

This command sets the voltage level on the specified channel that, when reached by a **negative-going** input signal, arms the digitizer.

ARM[:STARt]:LEVEL[<*n* >]:POSitive <*voltage* >

This command sets the voltage level on the specified channel that, when reached by a **positive-going** input signal, arms the digitizer.

The Arm Level Range

When setting the input voltage level which arms the digitizer, the levels are restricted to the current signal range as set by the expected value parameter of the CONFigure or MEASure command, or as set by the SENSE:VOLTage:RANGE command. Table 3-1 lists the digitizer's signal ranges.

Note When level arming, arms that occur because of multiple passes through the arm level are ignored until the trigger count is reached.

Setting the Arm Delay

There is an inherent delay of typically 230 ns plus an amount of sample period uncertainty from when an arm is **received**, to when the arm is **accepted**. When the digitizer is taking only post-arm readings, a delay in addition to the inherent delay can be specified with the command:

ARM[:STARt]:DELay <*period* >

The range for <*period* > is based on the frequency of the reference clock. If T is the reference period, then the arm delay period is expressed as:

0 to 65534T in steps of T
66540T to 655350T in steps of 10T

Using the digitizer's internal 20 MHz reference, this is:

0 to 3.27 ms
3.32ms to 32.76 ms

Note Appendix A - "Specifications" contains additional information on the inherent arm delay.

Setting the Arm Count

The arm count is the number of reading cycles (bursts) the digitizer will take on each channel before it returns to the idle state. The arm count is specified with the command:

ARM[:STARt]:COUNt <*count* >

The range for <*count* > is:

1 through 65,535 or INFinity (post-arm readings only)
1 through 128 (pre-arm and post-arm readings)

When only post-arm readings are specified, an error will occur when the total number of readings (ARM:COUNt * TRIGger:COUNt) is greater than 524,288 readings. However, an error will not occur if the arm count or trigger count is set to INFinity, or if the readings are going the VME bus or Local bus directly from the A/D converter.

When pre-arm readings are included in a measurement sequence with an arm count > 1, digitizer memory is segmented. When memory is segmented, the maximum number of arms is 128 and the maximum number of pre-arm and post-arm readings per arm is 524,288 / (number of arms rounded up to a power of 2).

Note If the arm count is set to INFinity, use the ABORt command to return the digitizer to the idle state.

Arm Rate

When the arm count is greater than one and the sample rate is derived from the digitizer's internal oscillator, the next arm occurs 0.5 µs after the last reading in the burst. This rate will vary with other references.

Immediate Arming

Once the digitizer has been placed in the wait-for-arm state with the INITiate command, the digitizer can be armed immediately with the following command, regardless of the arm source:

```
ARM[:STARt][:IMMediate]
```

ARM:IMM is also the only way to arm the digitizer when the arm source is HOLD and the digitizer is in the wait-for-arm state.

ARM:IMM decrements the arm count by 1, and any programmed delay (other than the inherent delay) is ignored.

ARM Synchronization Signals

As indicated in the block diagram of Figure 3-5, the arm signal can be "routed" to the following locations:

"Ext 1" BNC port
ECLTRG trigger lines
TTLTRG trigger lines

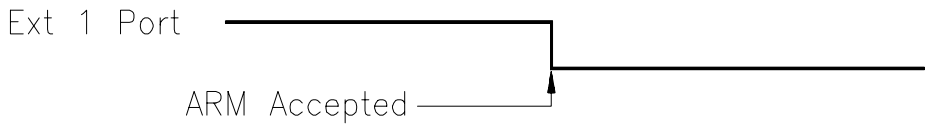
This allows the digitizer to synchronize other digitizers or events. The commands used to output the arm signal are:

OUTPut:EXTernal[1]:FEED <*source* >

- <*source* > = "ARM[:STARt|SEQuence[1]]"

  The "Ext 1" BNC port goes low when the arm occurs.

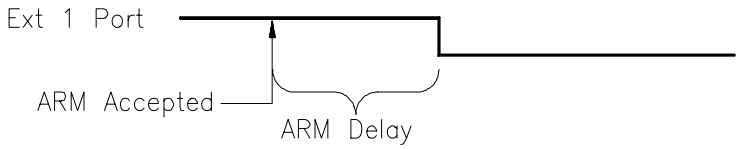
- <*source* > = "RFTRigger"



E1429A FIG2c

Outputs the arm signal to the "Ext 1" BNC port after the specified arm delay has occurred.

OUTPut:ECLTrg<*n* >:FEED <*source* >



E1429A FIG3a

- <*source* > = "EXTernal[1]"

  Outputs the same signal currently specified by the OUTPut:EXTernal[1]:FEED command, however, with the polarity inverted.

OUTPut:TTLTrg<*n* >:FEED <*source* >

- <*source* > = "ARM[:STARt|SEQuence[1]]"

  Outputs the arm signal to the specified TTLTRG trigger line as it is received.

E1429A FIG3b

Routing the Signal to a Source

The arm signal can be routed to the locations described above, provided the port or trigger line is not also used as an input for a reference clock source, arm source, or trigger source. Summarized below are the external sources available to the digitizer:

External reference clock sources: "Ext 2" BNC port, ECLTRG<n > trigger line

External arm sources: "Ext 1" BNC port, ECLTRG<n > trigger line, TTLTRG<n >* trigger line

External trigger sources: "Ext 1" BNC port, "Ext 2" port, ECLTRG<n > trigger line, TTLTRG<n >* trigger line

If one of these ports or trigger lines is an input source, then that **same** port or line cannot be used to route (OUTPut) the signal.

Enabling the Synchronization Signal

In order for the arm synchronization signal to be routed to the "Ext 1" port or to an ECLTRG or TTLTRG trigger line, the routing must be enabled. This is done with the commands:

OUTPut:EXTernal[1][:STATe] <*mode* >

OUTPut:ECLTrg<*n* >[:STATe] <*mode* >

OUTPut:TTLTrg<*n* >[:STATe] <*mode* >

For each command, the <*mode* > settings are:

ON - enables the port or trigger line to route the signal.
OFF - disables the port or trigger line from routing the signal.

Multiple TTLTRG trigger lines are individually enabled or disabled by executing the OUTP:TTLT<n >:STAT command for each line.

Note The TTLTRG trigger lines are not independent with regard to the synchronization signal supplied by FEED. This means that all TTLTRG trigger lines enabled by OUTP:TTLT<n >:STAT will carry the same synchronization pulse.

Triggering the Digitizer

After the digitizer is armed it enters the wait-for-trigger state. Thus, when trigger signals are received, the digitizer samples the inputs on its channels (Figure 3-5). The hardware associated with triggering the digitizer is controlled by the SCPI TRIGger subsystem, and by selected commands in the SENSe subsystem. The commands used in those subsystems include:

```
TRIGger
[:STARt|SEQuence[1]]
    :COUNt <count>
    [:IMMediate]
    :SOURce <source>
    :TIMer1 <period>
    :TIMer2 <period>

[SENSe[<chan>]]
:SWEep
    :OFFSET
        :POINts <count>
    :POINts <count>
```

TRIGger and SENSe Subsystems Overview

The following information summarizes each command in the TRIGger subsystem and the two commands from the SENSe subsystem. The Command Reference (Chapter 4) contains detailed information on the parameter settings and on reset conditions.

Setting the Trigger Source

The trigger source specifies the signal which triggers the digitizer to take a reading. The source is specified with the command:

TRIGger[:STARt]:SOURce <*source*>

The <*source*> parameters are:

- BUS - HP-IB Group Execute Trigger (GET) commmand or the *TRG common command.
- ECLTrg0/ECLTrg1 - the VXIbus ECLTRG0 or ECLTRG1 trigger lines.
- TTLTrg<n> - the VXIbus TTLTRG0 through TTLTRG7 trigger lines.
- EXTernal1 - the "Ext 1" input port.
- EXTernal2 - the "Ext 2" input port.
- HOLD - suspend triggering.
- TIMer - sample rate is specified by the TRIGger:STARt:TIMer1 command. The rate is derived from a reference specified by the SENSe:ROSCillator subsystem. TIMer is the default trigger source.
- VME - sample and/or transfer data from memory to the VME (VXI data transfer) bus when the data register at offset 12 ($0C_{16}$) in A24 address space is accessed.

The Sample Period

The sample period is the interval at which the digitizer takes readings. The sample period is derived from a reference which can be the digitizer's internal 20 MHz oscillator, or a reference from an external source (see "The Digitizer Reference Clock").

Setting the Sample Period

When the trigger source is TIMer, the sample period is set with the command:

TRIGger[:STARt]:TIMer1 <*period* >

The TIMer trigger source derives the sample period from the reference source (see "The Digitizer Reference Clock"). The period is a 1, 2, 4, 10, 20, 40 through 1E8, 2E8, 4E8 multiple of the reference source. Programmed sample periods that are not an exact 1, 2, 4, ... multiple are rounded to the nearest multiple of the source. If the digitizer cannot produce a sample rate that is within 1% of the specified sample rate, bit 2 of the Questionable Signal Status Group condition register is set.

The sample period required can be determined two ways. First, to take a given number of samples of a signal with frequency (f), the sample period is computed as:

<*period* > = signal period / number of samples

The reference is then divided by a value (N) which gives the sample period. For example, assuming the digitizer's 20 MHz oscillator is used and given the following:

input signal frequency = 1 kHz
number of samples = 100

then the <*period* > specified by TRIG:STAR:TIM1 is:

1 ms / 100 = 10 µs

To get the 10 µs sample period (100 kHz rate), the digitizer processor divides the reference by 200 (N):

20 MHz / N = 20 MHz / 200 = 100 kHz
1 / 100 kHz = 10 µs

Oversampling

The Nyquist criteria states that the sample rate must be at least 2 times the maximum frequency component of the input signal. To limit aliasing, it is recommended that a sample rate 4 times or greater the maximum frequency component be used. For example, to oversample at a frequency four times or 10 times the maximum frequency component, the sample period is computed as:

<*period* > = 1 / 4(f_C)

<*period* > = 1 / 10(f_C)

The minimum number of samples (i.e. <*size* > or TRIGger:STARt:COUNt) to take is determined by:

sample count = signal period (fundamental) / [1 / 4(f_C)] or [1 / 10(f_C)]

Dual Rate Sampling

TheHP E1429 digitizer's dual rate sampling feature allows pre-arm readings and post-arm readings to occur at different sample rates. The following trigger sources specify dual rate sampling:

- DECLtrg - VXIbus trigger line ECLTRG0 paces pre-arm readings, ECLTRG1 paces post arm readings.
- DEXTernal - input port "Ext 1" paces pre-arm readings, input port "Ext 2" paces post-arm readings.
- DTIMer - TRIGger[:STARt]:TIMer1 paces pre-arm readings, TRIGger[:STARt]:TIMer2 paces post-arm readings.

When the dual rate sampling trigger source is DTIM, one sample rate must equal the reference period. For example, if the digitizer's 20 MHz oscillator is used, one sample rate must be 50 ns. Similarly, if VXI CLK10 is used, one sample rate must be 100 ns (1/10 MHz). If the reference source for DTIM is ECLT0, ECLT1, or EXT2, one rate must be equal to the reference period. The other rate can be any other valid sample rate given the reference source.

When the dual rate sampling trigger source is DECL or DEXT, both sampling rates can be any periods you choose.

Note

It is recommended that pre-arm readings use the faster of the two sample rates. One additional sample pulse at the pre-arm rate, after the arm occurs, is required for the sample rate to change. The digitizer does not sample on this additional pulse. Post-arm sample pulses may be ignored depending on how fast they occur following the rate change. Refer to Appendix A - "Specifications", for additional information on dual rate sampling characteristics.

The Digitizer Reference Clock

When the digitizer trigger source is TIMer or DTIM (dual rate sampling), the sample (reading) rate is derived from a reference clock (Figure 3-5). The commands used to select the clock source and its frequency are part of the SENSe subsystem:

```
SENSe[<chan >]
:ROSCillator
    :EXTernal
        :FREQuency <frequency >
    :SOURce <source >
```

Setting the Reference Source

The reference source is set with the command:

```
SENSe[<chan >]:ROSCillator:SOURce <source >
```

The reference sources are:

- CLK10 - the VXIbus 10 MHz clock.
- ECLTrg0 - the VXIbus ECLTRG 0 trigger line.
- ECLTrg1 - the VXIbus ECLTRG 1 trigger line.
- EXTernal2 - the digitizer's Ext 2 BNC port.
- INTernal - the digitizer's internal 20 MHz oscillator. This is the default source.

Specifying the External Reference Frequency

When the trigger source is TIMer or DTIM, and the clock source is one of the following:

```
ECLTrg0
ECLTrg1
EXTernal2
```

the digitizer processor must be informed of the reference frequency in order for TRIGger:STARt:TIMer1 or TRIGger:STARt:TIMer2 to generate the correct sample rate. This is done with the command:

```
SENSe[<chan >]:ROSCillator:EXTernal:FREQuency <frequency >
```

Note An active reference clock (oscillator) is required for the digitizer to recognize and accept arm signals.

Setting the Trigger Count

The trigger count is the total number of readings (pre- and post-arm) per arm event. The trigger count is normally set by the <*size* > parameter of the CONFigure and MEASure commands. However, when you want to change the total reading count without entirely re-configuring the digitizer, the reading count (trigger count) can be set with either of the following commands:

TRIGger[:STARt]:COUNt <*count* >

SENSe[<*chan* >]:SWEep:POINts <*count* >

Setting the Pre-Arm Reading Count

Pre-arm readings are samples the digitizer takes before an arm signal is accepted. Pre-arm readings start after the digitizer receives the INITiate[:IMMediate] command. Readings continue until the pre-arm reading count is reached, and until the arm is received. The command which sets the minimum number of pre-arm readings is:

SENSe[<chan >]:SWEep:OFFSet:POINts <count >

<*count* > is specified as a negative number. You must specify at least three pre-arm readings and seven post-arm readings. Refer to the SENSe subsystem in Chapter 4 - "Command Reference" for more details on setting the pre-arm reading count.

If the pre-arm reading count is reached and the arm has not occurred, the readings continue and overwrite previous readings. When the arm occurs, the most recent number of <*count*> readings are stored in memory. If an arm occurs before the pre-arm reading count is reached, the arm is ignored and error -212 "Arm ignored" is generated.

Sending an Immediate Trigger

In some applications the user may want to manually issue a sample trigger. This is accomplished with the command:

TRIGger[:STARt][:IMMediate]

When the digitizer is armed (wait-for-trigger state), executing TRIG:STAR:IMM takes a reading regardless of the specified trigger source. The trigger count is decremented by 1 and the selected trigger source remains unchanged. TRIG:STAR:IMM is often used with TRIG:SOUR:HOLD to suspend triggering until the trigger is issued by TRIG:STAR:IMM.

Trigger Synchronization Signals

The trigger and reference clock signals can be "routed" to the following locations:

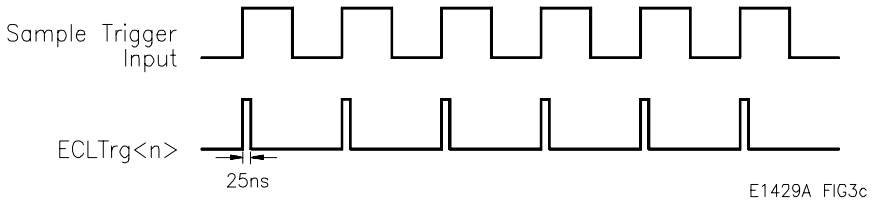
"Ext 1" BNC port
ECLTRG trigger line

This allows the digitizer to synchronize other digitizers or events. The commands used to output the trigger and reference signals are:

OUTPut:ECLTrg<*n* >:FEED <*source* >

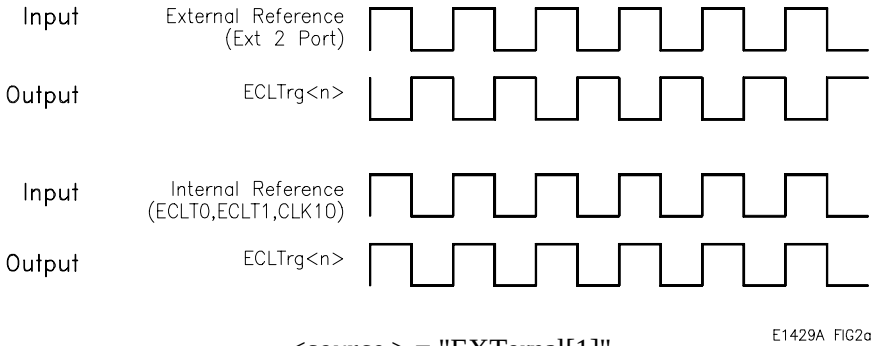
- <*source* > = "TRIGger[:STARt|SEQuence[1]]"

  Outputs a trigger signal to the ECLT<n > trigger line each time a convert pulse is sent to the A/D converter.



- <*source* > = "SENSe[1|2]:ROSCillator"

  The significant edge of an ECL signal is the rising edge. Therefore, the ECLTrg <n > trigger line goes high with the falling edge of an external reference signal, and goes high with the rising edge of reference sources ECLTrg<n > and CLK10.



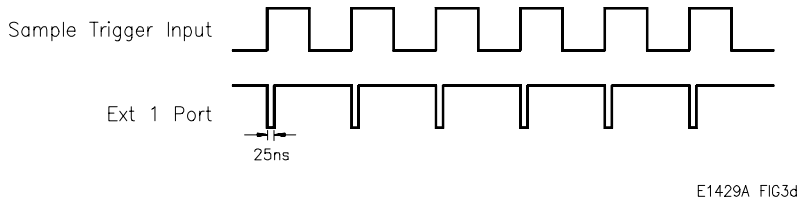
- <*source* > = "EXTernal[1]"

  Outputs the same signal currently specified by the OUTPut:EXTernal[1]:FEED command; however, the polarity is inverted.

OUTPut:EXTernal[1]:FEED <*source* >

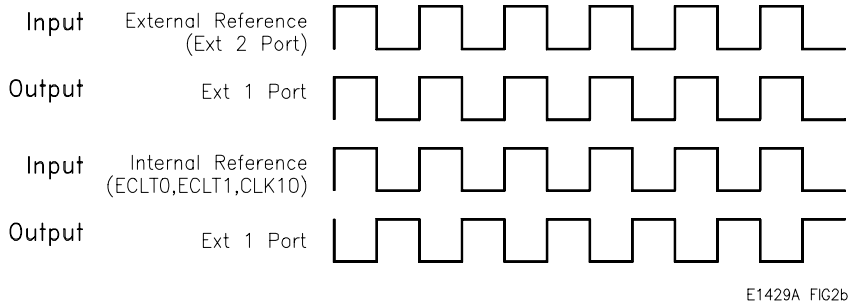
- <*source* > = "TRIGger[:STARt|SEQuence[1]]"

  Outputs a trigger signal to the "Ext 1" BNC port each time a convert pulse is sent to the A/D converter.



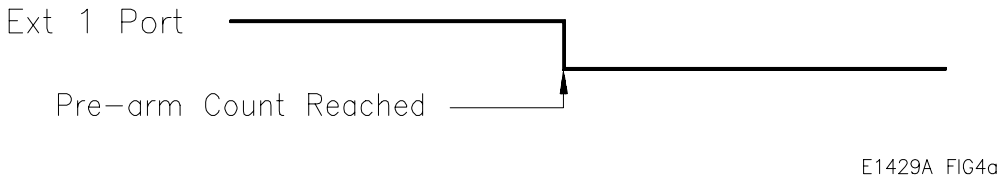
- <*source* > = "SENSe[1|2]:ROSCillator"

  The significant edge of a TTL signal is the falling edge. Therefore, the output goes low with the falling edge of an external reference signal, and goes low with the rising edge of reference sources ECLTrg<n > and CLK10.



- <*source* > = "SENSe:SWEep:OFFSet:POINts"

  The normally high output port goes low after the pre-arm reading count has been reached.

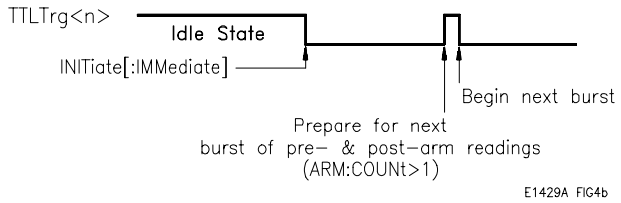


OUTPut:TTLTrg<*n* >:FEED <*source* >

- <*source* > = "READy"

  The level on the selected TTLTRG trigger line goes low while the trigger system is initiated. The line goes high after the readings complete, and then goes low again for the next burst of readings.

When taking post-arm readings only, the signal returns high when the arm count and trigger count are **both** satisfied.



Routing the Signal to a Source

The trigger signal can be routed to the locations described above, provided the port or trigger line is not also used as an input for a reference clock source, arm source, or trigger source. Summarized are the external sources available to the digitizer:

External reference clock sources: "Ext 2" BNC port, ECLTrg<n > trigger line

External arm sources: "Ext 1" BNC port, ECLTrg<n > trigger line, TTLTrg<n > trigger line

External trigger sources: "Ext 1" BNC port, "Ext 2" port, ECLTrg<n > trigger line, TTLTrg<n > trigger line

If one of these ports or trigger lines is a source, then that **same** port or line cannot be used to route (OUTPut) the signal.

Enabling the Synchronization Signal

In order for the trigger or clock synchronization signals to be routed to the "Ext 1" BNC port or to an ECLTRG or TTLTRG trigger line, the routing must be enabled. This is done with the commands:

OUTPut:EXTernal[1][:STATe] <*mode* >

OUTPut:ECLTrg<*n* >[:STATe] <*mode* >

OUTPut:TTLTrg<*n* >[:STATe] <*mode* >

For each command, the <*mode* > settings are:

ON - enables the port or trigger line to route the signal.
OFF - disables the port or trigger line from routing the signal.

Note The ECLTRG trigger lines are independent with regard to the synchronization signal supplied by FEED. This means that trigger lines ECLT0 and ECLT1 (when enabled) can carry different synchronization pulses.

The Analog-to-Digital Converter

Each channel on the HP E1429 digitizer has a 12-bit, 20 MSample/second analog-to-digital (A/D) converter. A paper describing the A/D converter, which was developed by Hewlett-Packard Laboratories, is available from the following:

Jewett, R., et al., "A 12b 20MS/s Ripple-through ADC", ISSCC DIGEST OF TECHNICAL PAPERS, pp. 34-35, Feb. 1992.

Data Flow, Storage, and Conversions

This section of the chapter covers the data flow from the A/D to digitizer memory and to the VME (VXI data transfer) bus. A block diagram of the data flow is shown in Figure 3-7.

Digitizer Data Flow

The HP E1429 digitizer takes readings (samples) on both channels simultaneously, even if an input signal is applied to only one channel. Each 12-bit reading is combined into a single 24-bit number which is sent to the data processor. The processor converts the readings from ECL levels to TTL levels.

Readings are stored and retrieved from memory as single 24-bit numbers (see "How Readings are Stored"). Each 12-bit reading sent to the VME (VXI data transfer) bus directly from the A/D or from memory is expanded

to 16-bits. When both channels readings are selected, the data is expanded to 32-bits; 16-bits per reading.

Similarly, readings sent to the Local bus from either the A/D or memory are expanded to 16- or 32-bits, depending on whether one or two channels is selected. The readings are transferred eight bits at a time, however.

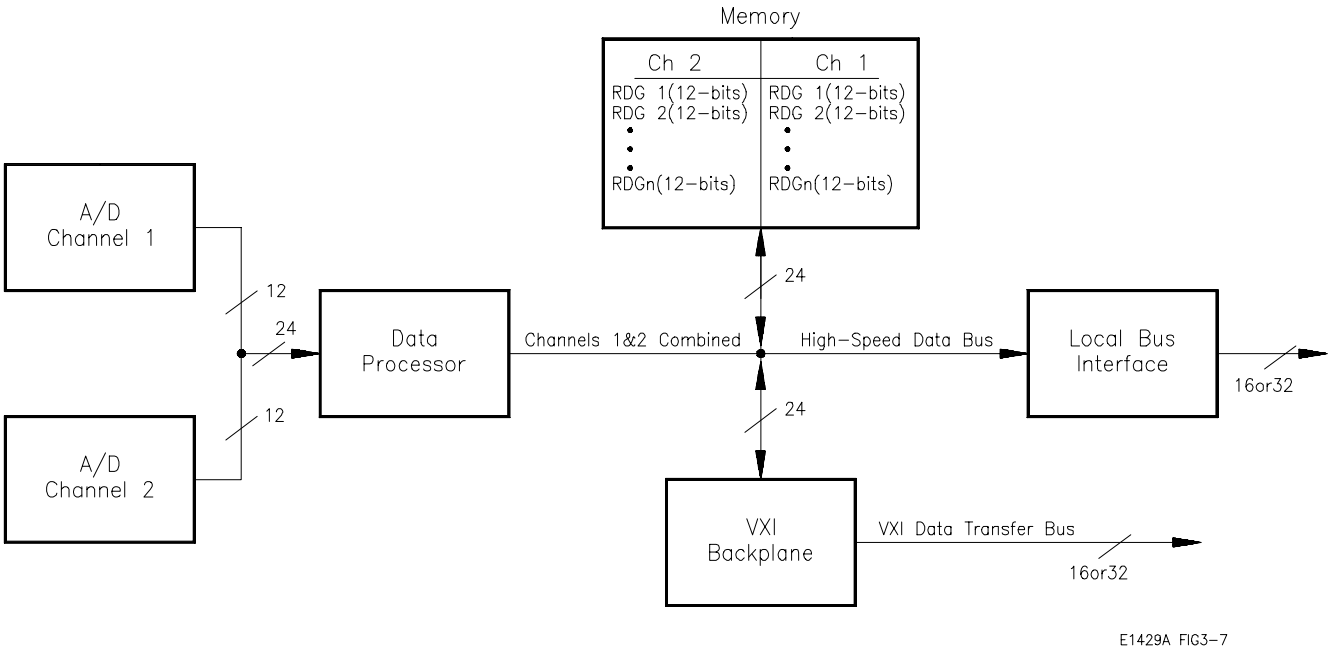


Figure 3-7. HP E1429 Digitizer Data Flow

How Readings are Stored

The digitizer can store 512K (524,288) readings from each channel in memory. The readings, which are taken simultaneously, are stored as a single 24-bit number. A "pair" of readings stored in a single memory location is shown in Figure 3-8.

Data is not available to the user in the internal storage (12-bit) format shown. Instead, readings retrieved from memory are expanded to 16- or 32-bits (see "Retrieving Readings from Memory").

Assigning a Data Label

When a digitizer reading is expanded to 16- or 32-bits as it is retrieved from memory, the four least significant bits are normally set to ’0’s. You can assign the bits a decimal value from 0 to 15 using the command:

DIAGnostic:CHANnel[<*chan* >]:LABel <*label* >

Assigning a data label allows you to identify the channel from which the readings came. The assigned lable appears in the four least significant bits of each reading when the readings are retrieved in the digitizer’s PACKed,16 data format. The label is ignored if the data format is ASCii,9 or REAL,64.

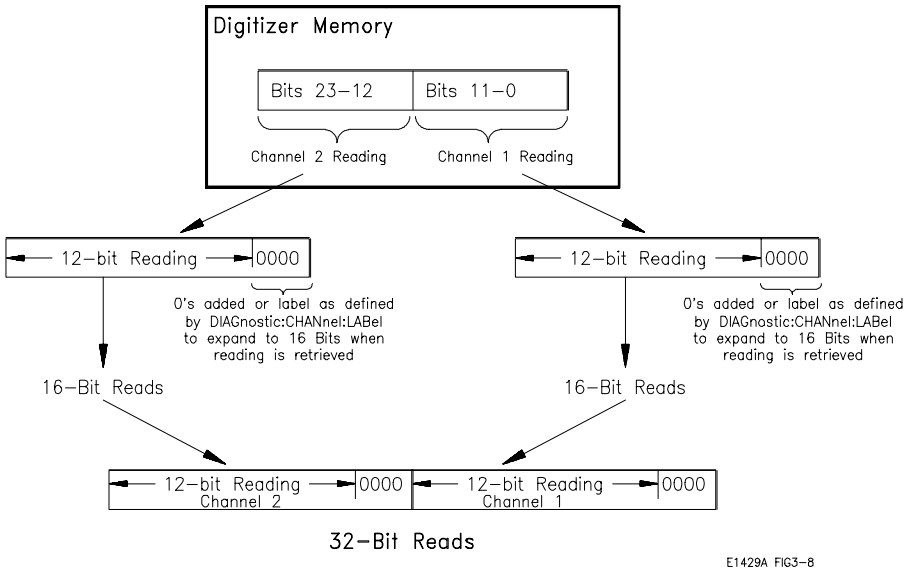


Figure 3-8. HP E1429 Digitizer Reading Storage

Segmented Memory

The HP E1429 digitizer is capable of pre-arm and post-arm readings. Pre-arm readings are taken after the digitizer is INITiated and before an arm is received. Post-arm readings are taken after an arm is received. When multiple reading sequences consist of pre- and post-arm readings, digitizer memory is partioned into segments (Figure 3-9).

When the reading sequence consists of pre-arm and post-arm readings, the number of segments is equal to the arm count. The maximum number of segments (arm count) under these conditions is 128.

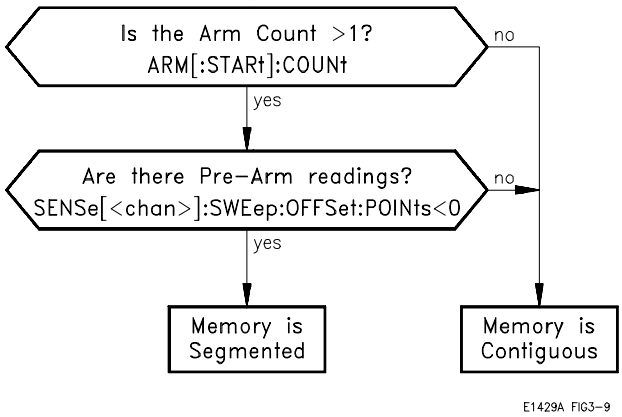


Figure 3-9. When the Digitizer Segments Memory

Figure 3-10 shows the relationship between pre- and post-arm readings and memory segments.

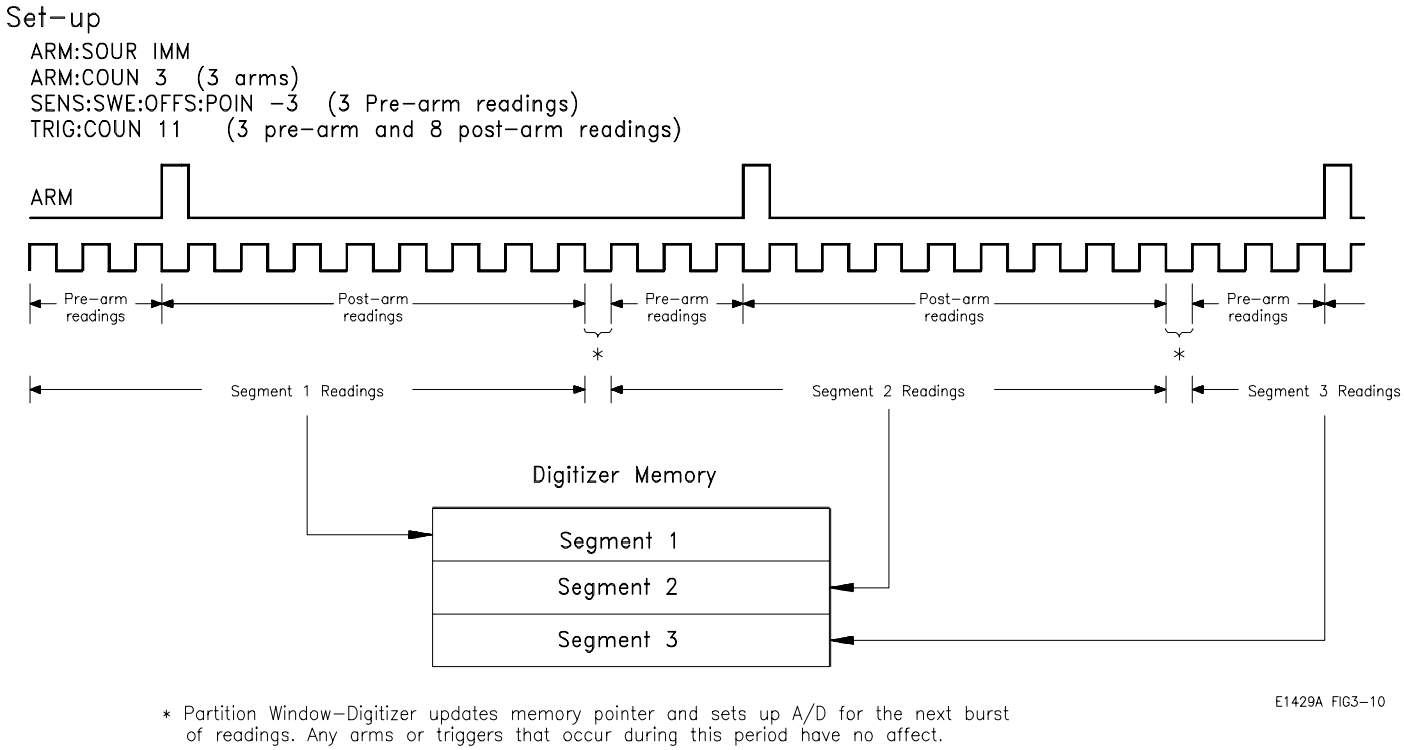


Figure 3-10. Memory Segments with Pre - and Post-arm Readings

The MEMory Subsystem

The digitizer's MEMory subsystem contains commands which enable non-volatile memory and which query the charge on the battery maintaining memory. The commands in the subsystem include:

```
MEMory
:BATTery
    :CHARge?
    [:STATe] <state >
```

Enabling Non-Volatile Memory

Digitizer memory is non-volatile when the battery maintaining memory is enabled with the command:

```
MEMory:BATTery[:STATe] < state >
```

The <*state* > settings are:

ON - enables the battery; memory is non-volatile.
OFF - disables the battery; memory is lost when power is cycled (factory setting).

If memory is to be battery-backed, the battery must be enabled **before** readings are taken.

Cycling power or resetting the digitizer does not affect the battery state set by MEMory:BATTery:STATe. Battery life is normally four years if it remains enabled. Battery life can be extended to approximately seven years if it is disabled when non-volatile memory is not required.

Determining the Battery Charge

To ensure that the battery has sufficient charge to maintain memory, the command:

MEMory:BATTery:CHARge?

can be sent. This command returns a '1' if the battery has sufficient charge to maintain memory. If the battery does not have sufficient charge, a '0' is returned. Leaving the digitizer in the VXI mainframe with the power on **will not** recharge the battery. A battery with insufficient charge must be replaced.

Digitizer Data Formats

There are three digitizer data formats available when readings are taken using MEASure or READ?, or when the readings are FETCh(ed)? from memory. These formats are set using the FORMat subsystem which consists of the command:

FORMat[:DATA] <*type* > [,<*length* >]

The formats available are:

<*type* >	<*length* >
ASCii	9
PACKed	16
REAL	64

ASCii,9 format returns/retrieves data as comma (,) separated ASCII numbers (Figure 3-11). This is the default format.

PACKed,16 format returns signed, 16-bit numbers preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header (Figure 3-11). Packed readings always represent a value between -1.0225 and +1.0230 or an overrange value, and must be converted (by the user) to the actual reading value (see "Packed Reading Conversions").

REAL,64 format returns/retrieves data as 64-bit REAL numbers also preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header (Figure 3-11). REAL,64 readings are converted by the digitizer to the actual reading value. Thus, no further conversions are required.

A line feed (LF) and End-Or-Identify (EOI) follow the last reading in all formats.

The Definite Length Arbitrary Block Header

In the definite length arbitrary block header:

- **#** indicates the data is in an arbitrary block
- **< non-zero digit >** is a single digit number which shows the number of digits contained in "digits". For example, if the "digits" value is 100 or 2000, the "non-zero digit" value is 3 or 4, respectively.
- **< digits >** is the number of 8-bit data bytes which follow the header.
- **<8-bit data byte >**are the digitizer readings. For the PACKed format, each reading is two bytes. For the REAL,64 format, each reading is eight bytes.

Following the last reading in each block is the line feed (LF) character. The line feed must be read from the buffer to prevent error -410 "Query INTERRUPTED" occurs the next time data is read from the digitizer.

Packed Reading Conversions

When the packed reading format is specified with the FORMat command, or when readings are read from memory and transferred to the VME bus or Local bus, the readings must be converted from signed, two’s complement numbers to voltages. Additionally, when FETChing packed readings, the definite length arbitrary block header must be removed. This section explains how to separate the block header from the packed data, and how to convert the data to voltages.

Removing the Arbitrary Block Header

Following are two methods of removing the block data header. The first method uses the HP BASIC programming language. The second method uses a command from the HP 82335 HP-IB Command Library for C.

HP BASIC Example

```
DIM Ndig$[1],Count$[9]!dimension parameters for header
Count$ = ""!set count to zeros
ASSIGN @X TO 70905;FORMAT OFF!return unformatted data
OUTPUT 70905;"FETC1?"!retrieve readings from channel 1
ENTER @X USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                    !remove header preceding the data
ALLOCATE INTEGER Meas_data(1:VAL(Count$)/2)
                                    !allocate an array to hold the data
ENTER @X;Meas_data(*)!read in the measurement data
ENTER 70905 USING "B";Junk!remove the line feed character
```

The parameters of the ENTER ... USING statement function as follows:

- terminate ENTER on last ENTER item (EOI)

X - skip the # character of the header

K - enter the <non-zero digit > part of the header into the Ndig$ variable

K - enter the <digits > part of the header into the Count$ locations specified

B - retrieve one byte (the line feed) from the digitizer

C Language Example (16-bit readings)

```
/* dynamically allocate memory for readings */
rdgs =  malloc(20 * sizeof(int));
/* set number of bytes placed in memory, and number of bytes read */
swap = sizeof(int);/* place 2 bytes/reading in memory */
bytes = 20 * swap;/* read 40 bytes */
IOOUTPUTS(ADDR, "READ?", 5);/* retrieve the readings */
IOENTERAB(ADDR, rdgs, &bytes, swap);      /* enter the readings */
                                          /* and  remove the block header */
/* Remove line feed which trails the last data byte */
length = 1;
IOENTERS(ADDR, lf_remove, &length);
```

The command which removes the block header from the readings is IOENTERAB. The parameters passed to IOENTERAB are:

ADDR - the address of the digitizer

rdgs - the array name which will store the readings

bytes - a variable specifying the maximum number of bytes to be read

swap - a variable specifying how the bytes are placed into memory (2 bytes per reading)

Converting Packed Readings

The equation for converting packed readings is:

$$reading_{voltage} = (reading_{packed} / 16) * reading\ resolution$$

Reading resolution is a function of the signal range. The resolutions for the digitizer signal ranges are given in Table 3-2.

Table 3-2. Reading Resolutions for Packed Data Conversions

Signal Range	Resolution	Signal Range	Resolution
-0.1023 to 0.10235	0.00005	-5.115 to 5.1175	0.0025
-0.2046 to 0.2047	0.00010	-10.230 to 10.235	0.005
-0.5115 to 0.51175	0.00025	-20.460 to 20.470	0.010
-1.0230 to 1.0235	0.0005	-51.15 to 51.175	0.025
-2.0460 to 2.0470	0.0010	-102.30 to 102.35	0.05

Overrange Indications

The digitizer indicates an overrange condition (input greater than the selected range can measure) by returning the values shown in Table 3-3.

An amplifier overrange (single-ended or differential input) sets bit 0 in the

Table 3-3. Digitizer Overload Readings

Condition	Reading	A/D Code
A/D Overrange (positive reading)	+9.900000E+037	+2047
A/D Overrange (negative reading)	-9.900000E+037	-2046
*S/E Input Amplifier Overrange or Differential Input Amplifier Overrange	-9.900000E+037	-2048
*A single-ended input amplifier overrange occurs when the input signal exceeds ± 5V. A differential input amplifier overrange occurs on the 0.1V, 0.2V, 0.5V, 1V, 2V, and 5V ranges when the signal on the HI or LO input exceeds ±11V. An overrange occurs on the 10V, 20V, 50V, and 100V ranges when the signal on the HI or LO input exceeds ±110V.		

digitizer's Questionable Signal Status Register. However, the overrange does not generate an error message.

Retrieving Readings

Each time the digitizer is INITiated it takes a new set of readings. Readings are retrieved directly from the A/D converters or from digitizer memory (Figure 3-11). Each new set of readings overwrites any readings currently in memory. Retrieved readings are transferred over the VME (VXI data transfer) bus or over the Local bus. The methods used to retrieve readings are listed below, relative to their transfer rates.

- READ? (page 139) **slowest**
- FETCh? (page 139)
- DIAGnostic:UPLoad:SADDress? (page 141)
- VME bus Data register access(page 146)
- Local bus transfers (page 156) **fastest**

These methods are described in the following paragraphs. Table 3-4 summarizes the digitizer's data transfer modes and transfer rates.

Table 3-4. HP E1429A/B Data Transfer Methods and Rates

Transfer Method	Transfer Rate (approximate)	Transfer Mode*	Reading Type
Local Bus	80 MBytes/s	Real Time / Post Measurement	Pre-arm / Post-arm
Embedded Controller (VME bus with SICL)	8 MBytes/s	Real Time / Post Measurement	Pre-arm / Post-arm
DIAGnostic:UPLoad:SADDress? (VME bus)	80 KBytes/s	Real Time / Post Measurement	Pre-arm / Post-arm
READ?/FETCh? (VME bus)	11 KBytes/s	Post Measurement	Pre-arm / Post-arm
* Real time readings come directly from the A/D. Post measurement readings come from memory. The transfer mode is set using the digitizer's VINStrument:LBUS:FEED and VINStrument:VME:FEED commands. See "'VME Bus DataTransfers" and "Local Bus Data Transfers" later in this chapter for more information.			

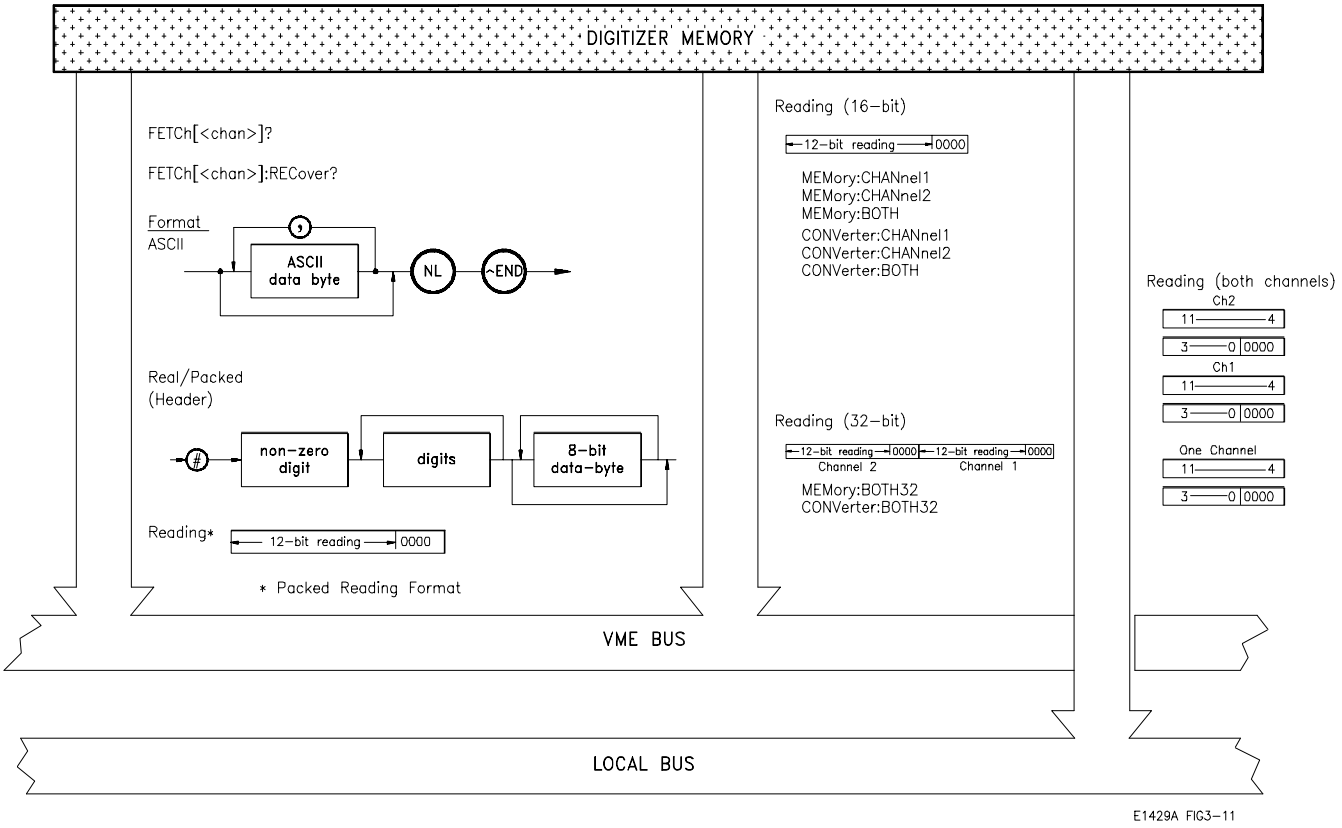


Figure 3-11. Retrieving Readings from Digitizer Memory

Retrieving Readings Using READ?

The most common method of retrieving readings from the digitizer is using the READ? subsystem which consists of the command:

READ[<*chan* >]?

The <*chan* > parameters are:

1 - returns readings from channel 1 (default channel)
2 - returns readings from channel 2

READ? is equivalent to executing the following sequence of commands:

- ABORt - aborts the readings after the arm count and trigger count is reached.
- INITiate - places the digitizer in wait-for-arm state.
- FETCh[<*chan* >]? - retrieves the readings from memory and places them on the VME (VXI data transfer) bus.

As a result, READ? is used for applications that require readings to be immediately available to a computer, rather than remaining in digitizer memory.

Because each READ? initiates the digitizer, executing READ1? followed by READ2? causes the digitizer to take two sets of measurements. If it is necessary to obtain readings from both channels during the same period, use INIT and FETCh? which are described in the next section.

Retrieving Readings Using FETCh?

One method of retrieving readings from memory involves the digitizer's FETCh? command shown below. FETChing readings from memory places them on the VME (VXI data transfer) bus
(Figure 3-11).

FETCh[<*chan* >]?
:COUNt?
:RECover?

FETChing Readings from Memory

Recall that the HP E1429 digitizer samples both channels simultaneously and stores the readings as a single 24-bit number (Figure 3-8). Using the FETCh? command, each channel's readings can be retrieved from memory individually.

The commands used to fetch (retrieve) readings from memory are:

FETCh[<*chan* >]?

If the <*chan* > parameter is ’1’ or is not specified, the readings (pre- and post-arm) from channel 1 and from the most recent INITiate - ARM - TRIGger sequence are retrieved. If the <chan > parameter is ’2’, the readings from channel 2 are retrieved.

FETCh[<*chan* >]:RECover?

This command is used to retrieve readings from non-volatile (battery-backed) memory following a power failure. The command is also used to retrieve readings after a digitizer configuration change or reset. Attempting to fetch (retrieve) readings using FETCh[<*chan* >]? following any of these conditions would cause error 230 "Data corrupt or stale" to occur.

If the <*chan* > parameter is ’1’ or is not specified, the readings from channel 1 are recovered. If the <chan > parameter is ’2’, the readings from channel 2 are recovered.

Determining the Number of Readings FETCh(ed)

The number of readings that FETCh? retrieves can be determined two ways. The first way is with the command:

FETCh[<*chan* >]:COUNt?

This command returns the total number of readings (pre- and post-arm) that the FETCh? command will retrieve. Since both channels always take the same number of readings, either channel can be specified.

The second way to determine the number of readings is to multiply the arm count by the trigger count. These two counts can be obtained from the queries:

ARM:STARt:COUNt? (arm count)
TRIGger:STARt:COUNt? (trigger count)

Separating Pre- and Post-Arm Readings

The number of pre-arm readings and the number of post-arm readings are related to the total reading count as follows:

post-arm readings = total readings - pre-arm readings

The FETCh[< chan >]:COUNt? command and the ARM:STARt:COUNt? and TRIGger:STARt:COUNt? queries described previously return the total number of readings. The pre-arm count is determined by taking the absolute value of the query:

SENSe[< *chan*>]:SWEep:OFFSet:POINts?

This command will return 0 if no pre-arm readings are specified, or a negative number representing the number of pre-arm readings.

Note If the digitizer measurement sequence is aborted (ABORt command), the FETCh[<chan >]:COUNt? command is the only way to determine the number of readings to be retrieved by FETCh?. If pre-arm readings are included in the measurement sequence, there is no way to determine the number of post-arm readings. The digitizer will attempt to return up to TRIGger[:STARt]:COUNt readings.

Using DIAGnostic:UPLoad:SADDress?

A third method of retrieving readings from memory is using the HP E1406 Command Module command:

DIAGnostic:UPLoad:SADDress? <*address* >, <*byte_count* >

<*address* > is the address of the digitizer's data register in A24 address space. The data register's offset in A24 space is 12 ($0C_{16}$). The A24 base address can be found as described in the section "Determining the A24 Base Address" later in this chapter.

<*byte_count* > is the number of (reading) bytes to upload from the digitizer. Because the HP E1405/06 Command Module is only capable of 8- and 16-bit data transfers over the VXI backplane, the *byte_count* parameter will always be 2 (bytes) times the number of readings.

In systems using an HP E1405/06 Command Module and an external controller, this command is the fastest method of transferring readings from memory.

Note Appendix C "Register-Based Programming" contains an example program in which DIAGnostic:UPLoad:SADDress? is used to retrieve readings from digitizer memory.

Memory Management

In certain applications it may be necessary to retrieve a selected set of readings from digitizer memory. This section explains where readings are stored in memory and how to determine the memory addresses of any set of readings.

The DIAGnostic Subsystem

This section introduces two commands used to locate and retrieve readings from memory. These commands, which are part of the DIAGnostic subsystem, are:

```
DIAGnostic
    :FETCh[<chan>]? <start_addr>, <count>
    :MEMory[<chan>]
        :ADDResses?
```

DIAGnostic:FETCh? returns *count* number of readings starting at address *start_addr* from channel *chan*. DIAGnostic:MEMory:ADDResses? returns a list of 32-bit values containing memory address information for each segment. The use of these commands is shown in the following paragraphs.

Locating Unsegmented Readings

Unsegmented readings (SENSe:SWEep:OFFSet:POINts 0) are always contiguous and are stored at:

address location 524287 - (num_readings - 1)

where num_readings = ARM:COUN * TRIG:COUN + pad, and pad are extra counts to make the total reading count divisible by 4. Therefore, the first reading is stored at address location
524287 - (num_readings - 1).

As an example, assume ARM:COUN 3 and TRIG:COUN 53. Then:

num_readings = 3 * 53 + 1 = 160
(1 is the pad count added to the total reading count)

first data point address = 524287 - (160 - 1) = 524128

This address, together with the count of 159 (3∗53) are specified in the DIAGnostic:FETch? command to retrieve the readings (from channel 1):

```
DIAG:FETC1? 524128,159
```

Locating Segmented Readings

For segmented readings (SENSe:SWEep:OFFSet:POINts ≤ -3), the algorithm for locating the readings is similar to that for unsegmented readings, but is slightly more complicated. The number of memory segments is determined by the specified arm count (ARM:STARt:COUNt) as shown in Table 3-5.

Table 3-5. Arm Count Vs. Memory Segments

ARM:STARt:COUNt	Number of Memory Segments	**Maximum Readings (TRIGger:STARt:COUNt)**
1	1	524,288
2	2	262,144
3 - 4	4	131,072
5 - 8	8	65,536
9 - 16	16	32,768
17 - 32	32	16,384
33 - 64	64	8,192
65 - 128	128	4096
NOTE: If the non-volatile mode of memory is enabled (MEMory:BATTery:STATe ON), then all of the maximum reading counts shown above decrease by four. These four memory locations in each segment hold the data necessary to recover all readings after a power failure.		

The algorithm for determining where the first reading of each segment begins is:

first reading in segment = ending segment address - (num_readings - 1)

where num_readings = TRIG:COUN + pad, and pad are extra counts to make the total reading count divisible by 4. The memory partition composed of num_readings is circular, and if large amounts of pre-arm data (readings) are taken, the data keeps overwriting itself until the arm is received and the post-arm count finishes. NOTE that the actual desired starting point for retrieving data will be:

last_data_point_address - (TRIG_COUN - 1)

which in most cases will not be the starting address if the segment has wrapped around at least once with pre-arm data. The digitizer firmware keeps track of the last address used in each segment and automatically reads the data in the proper order when a FETCh?, VME bus transfer, or Local bus data transfer is performed.

The DIAGnostic:MEMory:ADDResses? command returns a list of 32-bit integers for each segment in memory. In each 32-bit list, bit 1 is the aborted flag and bit 0 is the memory wrapped flag. Bit 31 through bit 2 is the value of the address counter for that segment. Thus, to obtain the memory address only, a divide by 4 (right shift of 2) of the 32-bit list must be done. Since the address counter points to the memory location where the *next* reading will be stored, the address should be decremented by 1 before use.

As an example, assume: ARM:COUN 5; SENS:SWE:POIN -20; TRIG:COUN 35

Because of the ARM:COUN specified memory is divided into eight segments (Table 3-5), each of which could contain up to 65536 readings. For ARM:COUN 5, only five of the eight possible segments will be used, starting with the segment that begins at address 0 and ends at address 65535.

The ending address for each of the five segments can be calculated from the equation:

ending segment address = segment_number * 65536 - 1

This yields the following ending addresses for the five segments:

segment 1 = 65535
segment 2 = 131071
segment 3 = 196607
segment 4 = 262143
segment 5 = 327679

Assume you want to read the data from segment number 4. Then, the number of readings = 35 + 1 = 36 (padded to make divisable by 4). The first reading in the segment is at address:

first address = ending segment address - (number of readings - 1)

262143 - (36 - 1) = 262108

If the readings in the memory segment **did not** wrap around, the address 262108 and count 35 would be specified in the DIAGnostic:FETCh? command.

If, for example, readings in the segment **did** wrap around, then the starting address specified in the DIAGnostic:FETCh? command will not be the first address in the segment where a reading is stored. To determine the starting address, use the DIAGnostic:MEMory:ADDRess? command and get the 32 bits representing the 4th segment (this would be the 13th through 16th bytes in the block of data returned).

Assume that this 32-bit value is $FFF81_{16}$. Bit 0 is high ('1') indicating readings have wrapped around the segment. Bit 1 is low ('0'), indicating that this segment completed normally and was not aborted (by the user with the ABORt command).

The address returned is divided by 4 so that the aborted and wrapped bits are discarded, and the address counter value of $3FFE0_{16}$ (262112_{10}) remains. To retrieve the **most recent** 35 readings (with a circular buffer size of 36), the address of the first reading to retrieve is calculated as:

starting address = address counter value - TRIGger:COUNt +buffer size

262112 - 35 + 36

starting address = 262113

Therefore, the DIAGnostic:FETCh? command would be executed as:

```
DIAG:FETC1? 262113,35
```

VME Bus Data Transfers

Another method of transferring readings to the VME (VXI data transfer) bus is with the digitizer's VINStrument (Virtual INStrument) subsystem and accessing the digitizer's data register. This method, which combines message-based (SCPI) programming and reading the data register directly, is faster than the previous methods (READ?, FETCh?, DIAGNostic:UPLoad:ADDRess?) in that readings can be retrieved from the A/D converter or from memory in the A/D's packed data format.

There are two modes of VME data transfers: real-time and post measurement. In a real-time data transfer, reading the digitizer's data register triggers a measurement and returns the A/D reading to the VME bus in the same measurement cycle. In a post measurement data transfer, reading the register transfers a A/D reading from digitizer memory to the VME bus.

How to select the transfer mode is covered in the section "Setting the VME bus Transfer Mode". Examples of VME bus data transfers are listed in Chapter 2 - "Using the Digitizer".

Locating the Data Register

Access to the digitizer's data register is through its address which is mapped into A24 address space. At power-on, the system resource manager reads the digitizer's device type register (in A16 address space) to determine the amount of A24 memory the digitizer needs (which is 4096 bytes). The resource manager allocates a block of A24 memory for the digitizer and writes the A24 base (starting) address into the digitizer's offset register (also in A16 space).

Figure 3-12 is an example of how the digitizer registers are mapped into A16 and A24 address space. Appendix C contains additional register programming information.

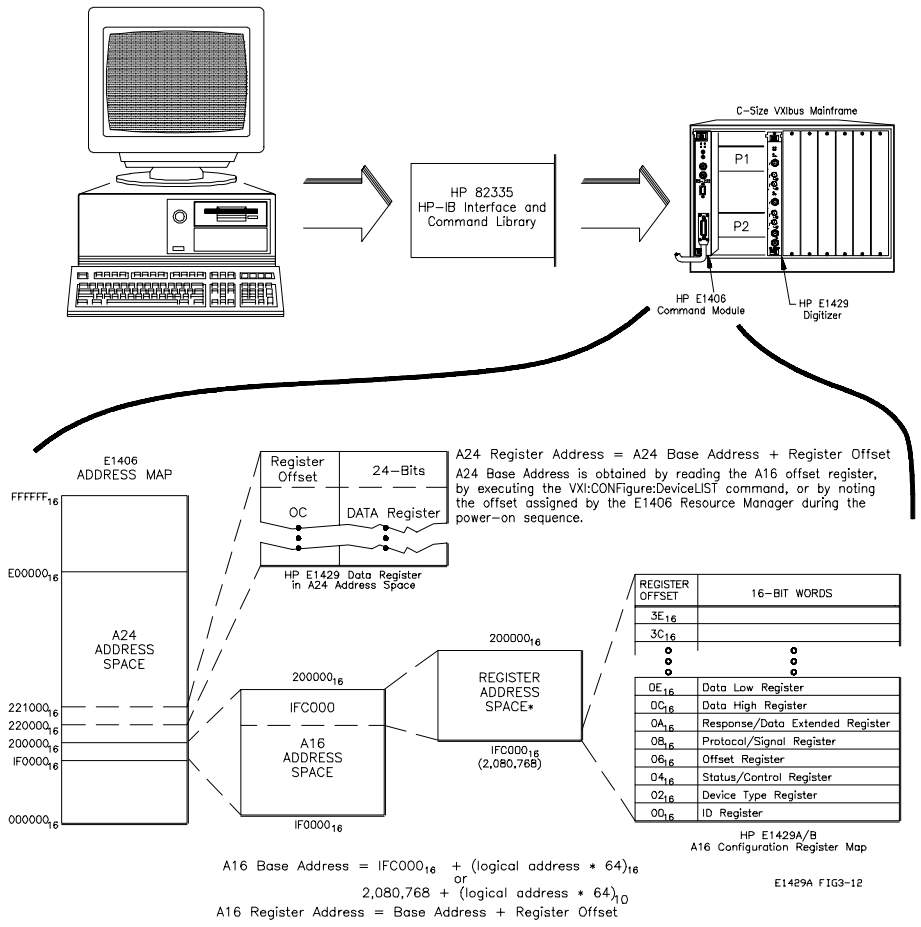


Figure 3-12. Digitizer Registers in A16 and A24 Address Space

---

Note The following information on determining the data register address is based on the computer configuration shown in Figure 3-12, and on address mapping as performed by the HP E1406 Command Module's resource manager. For configurations with embedded controllers or configurations with a resource manager other than the HP E1406 Command Module, refer to those manual(s) containing information on A24 address mapping.

Determining the A24 Base Address

There are three ways to determine the digitizer's A24 base address:

1. Note the base address assigned by the resource manager at power-on. The HP E1406 resource manager configuration sequence can be monitored using an RS-232 terminal or printer. The "C-Size VXIbus Systems Installation and Getting Started Guide" contains information on connecting a terminal.

2. Execute the following HP E1406 Command Module command:

   VXI:CONFigure:DeviceLIST? <logical_address>

   The C language example programs disk contains the program Query.C. By changing the line:

   #define ADDR 70905L (E1429 digitizer address)

   to:

   #define ADDR 70900L (E1406 address)

   and entering the command:

   VXI:CONF:DLIS? 40(or the current E1429 logical address)

   a program string similar to the following is returned when the program executes:

   vxi:conf:dlis? 40 = +40,+0,+4095,+448,+1,+0,MSG,A24, **#H00220000**, #H00001000,Ready,"","","",MBinstr INSTALLED AT SECONDARY ADDR 5"

   The hexadecimal number in **bold** is the digitizer's A24 base address.

3. Read the digitizer's offset register in A16 address space. As shown in Figure 3-12, the Offset register is one of the digitizer's configuration registers.

   In a system where the HP E1406 Command Module allocates address space, the A16 base address of the configuration registers is computed as:

   $1FC000_{16} + (LADDR * 64)_{16}$

   2,080,768 + (LADDR * 64)

   where $1FC000_{16}$ is the starting location of the configuration register addresses, LADDR is the digitizer's logical address, and 64 is the number of address bytes in A16 per VXI device.

   The digitizer's factory set logical address is 40. If this address is not changed, the base address of the digitizer's configuration registers in A16 is:

$1FC000_{16} + (40 * 64)_{16}$

$1FC000_{16} + A00_{16} = 1FCA00_{16}$

or decimal

2,080,768 + (40 * 64)

2,080,768 + 2560 = 2,083,328

Given the A16 base address and the "offset" of the Offset register (06 from Figure 3-12), the digitizer's A24 base address can be determined as shown in the program A24_READ.C.

```
/* A24_READ.C - This program reads the digitizer's A24 base address. */

/* Include the following header files */

#include   <stdio.h>
#include   <cfunc.h>       /* This file is from the HP-IB Command Library */

#define CMD_MOD 70900L     /* I/O path between the digitizer and the Command Module */

/* Function prototypes */

long get_base_addr(void);

/*****************************************************************************/
void main(void)
{
long base_addr;/* variable for digitizer A24 base address */


base_addr = get_base_addr();/* function call to calculate and */
       /* return digitizer A24 base address */

printf("\nA24 base address = %ld", base_addr);
}

/*****************************************************************************/
long get_base_addr(void)
{
/* base address of (A24) offset register in A16 address space */
long base_addr = (0x1FC000 + (40 * 64)) + 6; /* digitizer logical address is 40 */
```

Continued on Next Page

```
float a24offst;/* A24 offset from A16 offset register */

char rd_addr[80];/* command string variable */

/* Create the command string which reads the A24 base address from the offset register*/
sprintf(rd_addr, "DIAG:PEEK? %ld, %d", base_addr,16);

/* Send DIAG:PEEK? command */
IOOUTPUTS(CMD_MOD, rd_addr, strlen(rd_addr));

/* Read value from offset register */
IOENTER(CMD_MOD, &a24offst);

/* Multiply offset value by 256 for 24-bit address value */
a24offst *= 256.;

return (long)a24offst;
}
```

Multiplying the value of the offset register (a24offst) by 256 (100_{16}) converts the 16-bit register value to a 24-bit address.

The Data Register Offset

The offset of the digitizer's data register is 12 ($0C_{16}$). The offset is added to the A24 base address to form the complete register address:

$00220000_{16} + 0C_{16} = 0022000C_{16}$

2,228,224 + 12 = 2,228,236

The offset of the data register can also be read with the SCPI command:

VINStrument[:CONFigure]:VME:SEND:ADDRess:DATA?

This command returns two values: A24,12. A24 indicates that the data register is in A24 address space and 12 is the offset of the data register.

The VINStrument Subsystem

The commands within the VINStrument subsystem used for VME data transfers are shown below:

```
VINStrument
    [:CONFigure]
        :VME
            :FEED <source >
            :MEMory
                :INITiate
            [:MODE] <mode >
            :SEND
                :ADDRess
                    :DATA?
```

VME Bus Transfer Programming Sequence

To configure the digitizer for VME bus data transfers:

- Use the CONFigure command and the low-level digitizer commands to specify the number of readings, the expected value of the readings, the input port, the arm source, etc.
- Use the TRIGger:STARt:SOURce command to set the trigger source to VME (for real time data transfers only).
- **Use the VINStrument subsystem to set the VME bus transfer mode and data source.**
- Use the INITiate:IMMediate command to initiate reading transfers.

Setting the VME Bus Transfer Mode

The VME bus data transfer mode is set with the command:

VINStrument[:CONFigure]:VME[:MODE] <*mode* >

The <*mode* > parameters are:

GENerate - VME bus data transfers are enabled
OFF - VME bus data transfers are disabled. At power-on or following a reset, the <*mode* > is OFF.

Note

When transferring data over the VME bus, the Local bus transfer mode (HP E1429B) must be disabled. This is done with the command:

VINStrument[:CONFigure]:LBUS[:MODE] OFF

Setting the VME bus Data Source

The source of the readings transferred over the VME bus is set with the command:

VINStrument[:CONFigure]:VME:FEED <*source* >

The <*source* > parameters are given below. Sources beginning with "MEMory: " are the post measurement sources (modes), sources beginning with "CONVerter: " are the real time sources (modes).

" MEMory:CHANnel1": Channel 1 memory is the data source for the VME bus. One 16-bit reading is returned.

" MEMory:CHANnel2": Channel 2 memory is the data source for the VME bus. One 16-bit reading is returned.

" MEMory:BOTH": Both channels of memory are the data source for the VME bus. In this mode, channel 1 will be output the first time the data register is accessed, channel 2 is output the second time the data register is accessed. One 16-bit reading is returned with each access.

" MEMory:BOTH32": Both channels of memory are the data source for the VME bus. In this mode, accessing the data register returns a 32-bit number where the high order 16 bits are the channel 2 reading and the low order 16 bits are the channel 1 reading (see Figure 3-11).

" CONVerter:CHANnel1": The channel 1 A/D converter is the data source for the VME bus. One 16-bit reading is returned.

" CONVerter:CHANnel2": The channel 2 A/D converter is the data source for the VME bus. One 16-bit reading is returned.

" CONVerter:BOTH": Accessing the data register triggers both A/D converters at the same time, and one 16-bit reading (channel 1) is returned. Accessing the data register a second time returns the second 16-bit reading (channel 2), but does not trigger the A/Ds.

" CONVerter:BOTH32": Accessing the data register triggers both A/D converters at the same time, and one 32-bit number is returned. The high order 16 bits are the channel 2 reading, and the low order 16 bits are the channel 1 reading (Figure 3-11).

If it is necessary to identify readings from channel 1 and channel 2, a data label can be assigned to the readings as described in the section "Assigning a Data Label" on page 130.

Reading the Data Register

As shown in Figure 3-12, the digitizer's data register is mapped into the HP E1406 Command Module's A24 address space. A command used to read the data register is the Command Module command:

DIAGnostic:PEEK? <*address* >, <*width* >

<*address* >: the address (A24 base + offset) of the data register

<*width* >: the number of bits per digitizer reading. For the VME transfer modes, width is 16 (bits) with the exception of modes "MEMory:BOTH32" and "CONVerter:BOTH32", which are 32 bits. The HP E1406 Command Module cannot do 32-bit transfers to/from the VME (VXI data transfer) bus. However, 32-bit transfers at approximately 8 MByte/ second transfer rates are possible using an embedded controller.

Note

Additional information on the Command Module's DIAGnostic:PEEK? command is located in the HP E1406 User's Manual. Chapter 2 "Using the Digitizer" contains examples of 16-bit and 32-bit data transfers using an embedded PC.

VME Bus Data Format

Data is transferred (real time or post measurement) over the VME bus in the digitizer's packed (A/D) format. Data in this format are left-justified, signed 2's complement numbers . See "Packed Reading Conversions" on page 134 for information on converting the readings to voltages.

Segmented Reading Transfers

Multiple arms (bursts) of pre-arm and post-arm readings segment digitizer memory (Figure 3-13). Real time and post measurement transfers of segmented readings are also achieved by reading the data register. For high-speed configurations such as those which use an embedded controller, the time at which post measurement readings can be retrieved and a delay (partition window) while the digitizer sets up the next memory segment for reading storage and retrieval must be accounted for (Figure 3-13). Post measurement readings (either segmented or non-segmented) can be retrieved when bit 4 of the traffic register (base + 02_{16}) is set high ('1'). The partition window delay is monitored by reading bit 1 of the arm status register (base +43_{16}), or by reading bit 9 (READy) of the digitizer's Condition register in the Operation Status Group (page 169). For real time transfers, monitoring either partition window bit is best accomplished when the arm source is immediate (ARM:STARt:SOURce IMMediate).

To transfer segmented readings, the data register is accessed until each reading (pre- and post-arm) in the segment is transferred. After the last reading is transferred, bit 1 of the arm status register (base $+43_{16}$) or bit 9 of the condition register is monitored for a low to high transition. When the bit is high ('1'), the next segment can be transferred.

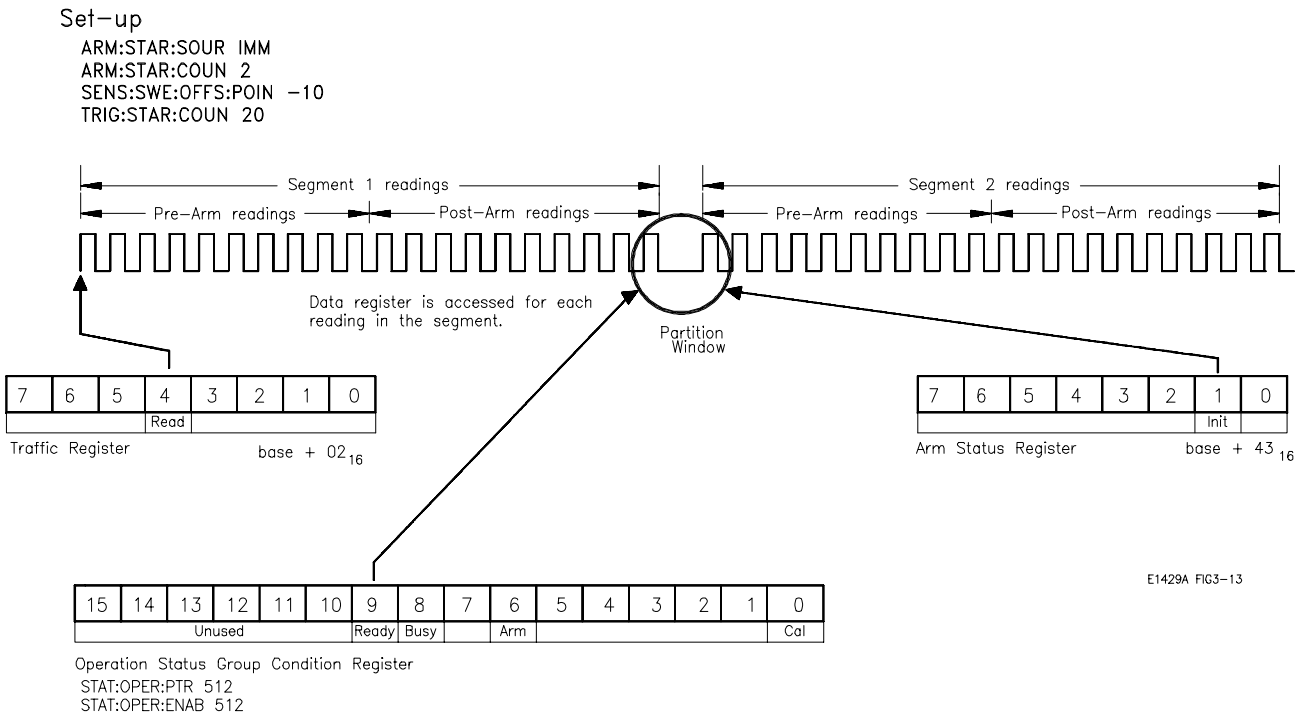


Figure 3-13. Monitoring the Partition Window During Segmented Reading Transfers

---

Note It is only necessary to monitor the partition window bit when the digitizer readings are segmented and the data register is accessed at speeds available through an embedded controller. Also, monitoring bit 1 of the arm status register (base $+43_{16}$) is faster than using SCPI commands to monitor condition register bit 9.

Bit 4 of the traffic register (base $+02_{16}$) is monitored so that post measurement readings, either segmented or non-segmented, can be transferred immediately once all of the readings are available.

Multiple VME Bus Data Transfers

During real time data transfers, readings are taken directly from the A/D converter and sent to the VME bus. These readings are **also** stored in digitizer memory. Transferring (real time or post measurement) readings from memory does not remove the readings from memory. As a result, a set of readings can be transferred to the VME bus multiple times. The digitizer is configured for an additional data transfer with the command:

VINStrument[:CONFigure]:VME:MEMory:INITiate

Reading the data register after executing this command transfers the data to the VME bus. If a subset of the total number of readings is transferred, execute the ABORt command following the transfer to allow for additional transfers. The digitizer should be configured with VINStrument[:CONFigure]:VME:MEMory:INITiate for each subsequent data transfer.

Local Bus Data Transfers

This section describes the use of the VXI backplane Local bus. The HP E1429B digitizer uses the Local bus for high-speed (80 MBytes/second) data transfers between two or more devices installed in adjacent mainframe slots.

Local Bus Description

The Local bus is a set of 12 signal lines on the backplane P2 connector. The Local bus is segmented such that each device with Local bus capability has two interfaces, one to each of the modules in the physically adjacent slots. A device may connect the bus segments together, or connect to each segment independently. Figure 3-14 shows how the lines are used by the HP E1429B digitizer.

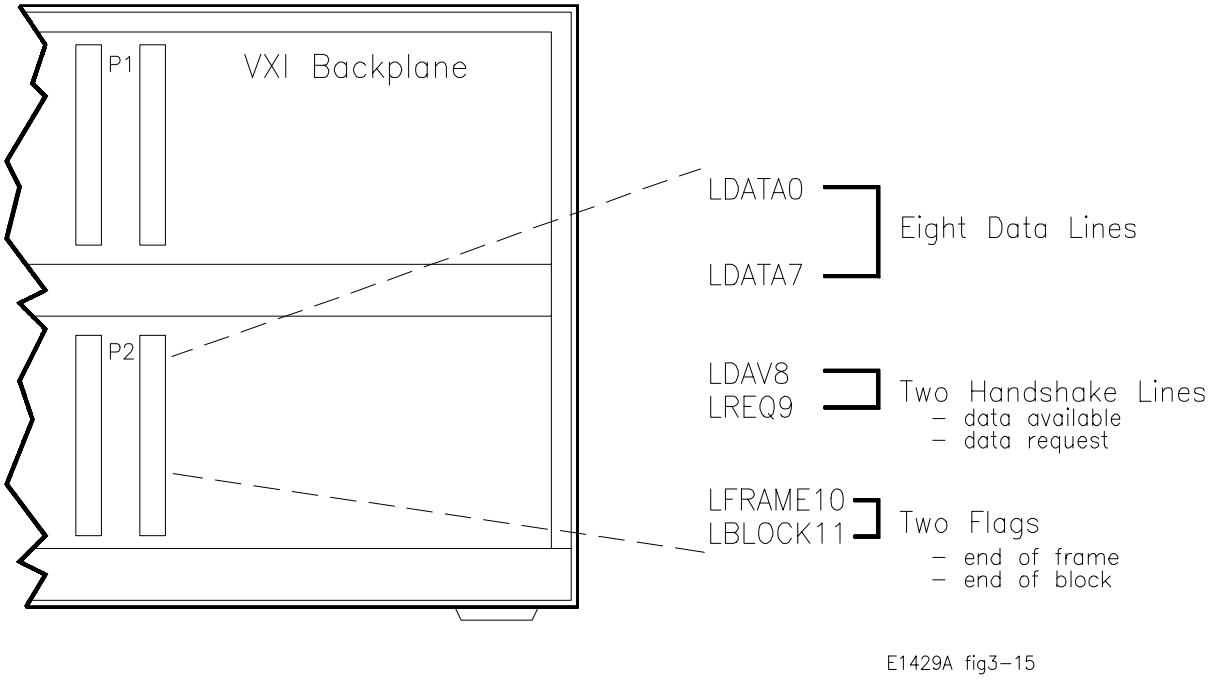


Figure 3-14. Local Bus Signal Line Definitions

Data flows left to right over the Local bus through adjacent slots. The following signal levels are allowed on the bus:

Analog low: ±5.5V, 50Ω
medium: ±16.0V, 500 mA
high: ±42.0V, 500 mA

Digital TTL: -0.5V to +5.5V, 200 mA
ECL: -5.46V to 0.0V, 50 mA

A Local bus key (Figure 1-1) prevents devices with incompatible Local bus signal levels from being installed in adjacent slots.

How Data is Transferred

Data is transferred over the Local bus in units of blocks and frames (Figure 3-15). A frame consists of a series of blocks, and each block is a stream of data bytes generated by a single device. If only one device is generating data, then the block size and frame size are the same.

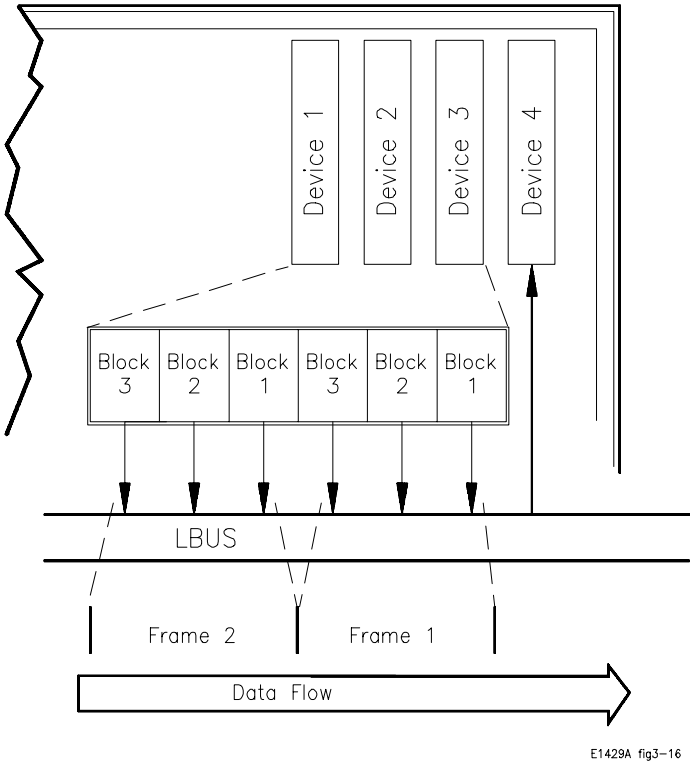


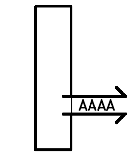
Figure 3-15. HP E1429B Local Bus Data Transfer Protocol

Handshake Protocol

The LDAV8 and LREQ9 lines are used to handshake bytes of data across the Local bus. Each data generating device asserts the LBLOCK11 flag with the last data byte in the block. The device which generates the last block in the frame asserts the LFRAME10 flag along with the LBLOCK11 flag.

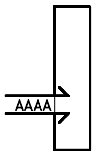
Local Bus Modes

There are several modes of operation used to generate or receive data over the Local bus. The most common modes follow. The **digitizer modes** are shown in **bold** and more information on these modes is contained in the section "Setting the Local Bus Transfer Mode" on page 162.



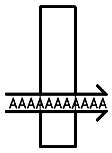
Generate

Generate: In this mode, a device is sending data to the Local bus.



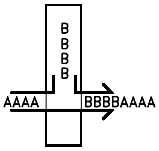
Consume

Consume: In this mode, a device is receiving data from the Local bus.



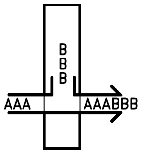
Pipeline

Pipeline: In this mode, a device is neither generating nor consuming data. Data is passed through from the device on the left to the device on the right.



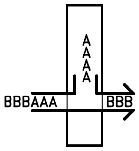
Append

Append: In this mode, a device pipelines the data until the end-of-frame flag is detected. When the flag is detected, the device appends its block of data and sets new end-of-block and end-of-frame flags.



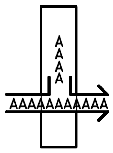
Insert

Insert: In this mode, the device places a block of data and an end-of-block flag on the Local bus, and then pipelines data from the device on the left to the device on the right.



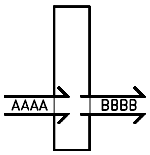
Strip

Strip: In this mode, a device alternates between the pipeline and consume modes. It removes a block of data from the beginning of each frame.



Eavesdrop

Eavesdrop: In this mode, a device is simultaneously consuming and pipelining data. It copies each byte of data as it is passed along.



Transform

Transform: In this mode, a device generates and consumes data.

Digitizer Local Bus Commands

The commands used to configure the HP E1429B digitizer for Local bus data transfers are part of the VINStrument subsystem:

```
VINStrument
    [:CONFigure]
        :LBUS
            :FEED <source>
            :MEMory
                :INITiate
            [:MODE] <mode>
            :RESet
            :SEND
                :POINts <count>
                    :AUTO <mode>
        :TEST
            :DATA <voltage_list>
```

Local Bus Transfer Configurations

The programming sequence used for Local bus transfers depends on the transfer configuration. The transfer configurations covered in this manual include:

- Single digitizer
  - post measurement transfers from digitizer memory
  - real time measurement transfers from the digitizer A/D
- Multiple digitizers and serial transfers
  - post measurement transfers from digitizer memory
  - real time measurement transfers from the digitizer A/D
- Multiple digitizers and interleaved transfers
  - interleaved real time transfers from the digitizer A/D

The programming sequences for the first two configurations follow. Chapter 2 contains example programs using these sequences.

Note Refer to Appendix D "Local Bus Interleaved Data Transfers" for programming information on transferring data from multiple digitizers in an interleaved sequence.

Single Digitizer

In a single digitizer configuration, the digitizer is usually the data generator and a device such as the HP E1488 memory card is the consumer. With this configuration, the programming sequence is:

1. Use the CONFigure command and the low-level digitizer commands to configure the digitizer for the required measurements.
2. **Use the VINStrument subsystem to reset the digitizer's Local bus chip, to set the Local bus transfer mode (generate), and to set the data source (post measurement or real time transfer).**

   VINStrument:CONFigure:LBUS:RESet
   VINStrument:CONFigure:LBUS:MODE <mode>
   VINStrument:CONFigure:LBUS:FEED <source >

3. Reset the consumer's (i.e. memory card's) Local bus chip and configure the consumer to receive data.
4. Activate (initiate) the consumer.
5. Use INITiate:IMMediate to activate the digitizer and start reading transfers.

Multiple Digitizers and Serial Transfers

In a configuration with two or more digitizers transferring data serially, the leftmost digitizer is set to the generate mode and the succeeding digitizer(s) is set to either the append mode or the insert mode. A device such as the HP E1488 memory card is usually the consumer.

In a serial transfer, each digitizer transfers data from its memory or directly from its A/D in sequence. For example, serial transfers from digitizers D_1 - D_3 where D_1 is in **generator** mode and D_2 and D_3 are in **append** mode would appear as:

EOF EOB D_3 D_3 D_3 EOB D_2 D_2 D_2 EOB D_1 D_1 D_1 -----> consumer

When D_1 is in **generator** mode and D_2 and D_3 are in **insert** mode the transfer sequence would appear as:

EOF EOB D_1 D_1 D_1 EOB D_2 D_2 D_2 EOB D_3 D_3 D_3 -----> consumer

In these sequences, EOF is the end-of-frame flag, EOB is the end-of-block flag, and D_n is either a two byte (one channel) or four byte (two channel) reading. The procedure for a serial transfer is:

1. Use the CONFigure command and the low-level digitizer commands to configure the digitizers for the required measurements.

2. **Use the VINStrument subsystem to reset the leftmost digitizer's Local bus chip, to set the Local bus transfer mode to generate, and to set the data source (post measurement or real time transfer).**

   Use the VINStrument subsystem to reset the inner digitizer's Local bus chip, to set the Local bus transfer mode to append or insert, and to set the data source (post measurement or real time transfer).

   VINStrument:CONFigure:LBUS:RESet
   VINStrument:CONFigure:LBUS:MODE <mode>
   VINStrument:CONFigure:LBUS:FEED <source >

   Note that you must reset the Local bus chip of each appender or inserter digitizer each time the generator digitizer's Local bus mode or data source is changed.

3. Reset the consumer's (i.e. memory card's) Local bus chip and configure the consumer to receive data.

4. Activate (initiate) the consumer.

5. If the digitizer(s) is in the append mode, use INITiate:IMMediate to activate the appender digitizer first. Then use INITiate:IMMediate to activate the generator digitizer.

6. If the digitizer(s) is in the insert mode, use INITiate:IMMediate to activate the generator digitizer first. Then use INITiate:IMMediate to activate the inserter digitizer.

Digitizer Configuration Restrictions

The HP E1429B digitizer can be configured for measurements as required with the following exceptions:

- If the Local bus data source is the digitizer A/D (real time transfers), only post-arm readings are allowed. This includes multiple arms (bursts).

- When the Local bus mode is set to a mode **other** than OFF or pipeline, the VME bus transfer mode must be disabled. This is done with the command:

  VINStrument[:CONFigure]:VME[:MODE] OFF

Setting the Local Bus Transfer Mode

The Local bus transfer mode is set with the command:

VINStrument[:CONFigure]:LBUS[:MODE] <*mode* >

The <*mode* > parameters are:

APPend: Local Bus data is received from the left, and passed on to the right until an end of frame is detected. When end of frame is received from the left side, all data from this module is appended, followed by an end of block marker and a new end of frame. After sending the end of frame marker, the module enters the paused state. This mode requires a module to the left that is in GENerate mode. The mode is not active until either an INITiate command or a VINStrument:LBUS:MEMory:INITiate command is sent.

GENerate: Local bus data originates in this module and is passed to the right, followed by an end of frame marker. The mode is not active until either an INITiate command or a VINStrument:LBUS:MEMory:INITiate command is sent.

INSert: Local bus data is inserted onto the bus from this module. The module will place its data out onto the Local bus with an end-of-block flag at the end. The module will then pass through (pipeline) any data it receives from the left, and will enter the paused state when an end-of-frame flag is received from the left. This mode requires at least one module to the left which is in GENerate mode. The mode is not active until either an INITiate command or a VINStrument:CONFigure:LBUS:MEMory:INITiate command is sent.

OFF: The Local bus interface is disabled immediately upon receipt of this command. Local bus data is neither used nor passed through.

PIPeline: Local bus data is passed through and not altered. This mode becomes effective immediately upon receipt of this command. Select this mode when data should be transparently passed through the HP E1429B. The module will remain in the PIPeline mode even after an end-of-frame flag is received; therefore, it is necessary to change modes to take the module out of PIPeline mode.

Setting the Local Bus Data Source

The source of the readings transferred over the Local bus is set with the command:

VINStrument[:CONFigure]:LBUS:FEED <*source* >

The <*source* > parameters follow. Sources beginning with "MEMory: " are the post measurement sources, sources beginning with "CONVerter:" are the real time (A/D) sources. The Local bus data source is independent of any Local bus transfer mode.

" MEMory:CHANnel1" : Channel 1 memory is the data source for the Local bus. Two bytes per reading will be output to the bus.

" MEMory:CHANnel2" : Channel 2 memory is the data source for the Local bus. Two bytes per reading will be output to the bus.

" MEMory:BOTH" : Both channels of memory are the data source for the Local bus. In this mode, the channel 2 reading is output first, followed by the channel 1 reading. With two bytes per reading, four bytes for each set of readings will be output to the bus.

" CONVerter:CHANnel1" : The channel 1 A/D converter is the data source for the Local bus. Two bytes per reading will be output to the bus.

" CONVerter:CHANnel2" : The channel 2 A/D converter is the data source for the Local bus. Two bytes per reading will be output to the bus.

" CONVerter:BOTH" : Both A/D converters are the data source for the Local bus. In this mode, the channel 2 reading is output first, followed by the channel 1 reading. With two bytes per reading, four bytes for each set of readings will be output to the bus.

Local Bus Data Format

Data is transferred (real time or post measurement) over the Local bus in the digitizer's packed (A/D) format. Data is transferred one byte at a time in the following sequence:

Two Channels: channel 2 MSByte
channel 2 LSByte
channel 1 MSByte
channel 1 LSByte

One Channel: channel n MSByte
channel n LSByte

Additionally, if the transfer configuration involves multiple digitizers, appender digitizer readings will follow the generator digitizer readings. Inserter digitizer readings will precede the generator digitizer readings.

Data in the packed format are left-justified, signed 2's complement numbers (Figure 3-8). See "Packed Reading Conversions" on page 134 for information on converting the readings to voltages.

Multiple Local Bus Data Transfers

During real time data transfers, readings are taken directly from the A/D converter and sent to the Local bus. These readings are **also** stored in digitizer memory. Transferring (real time or post measurement) readings from memory does not remove the readings from memory. As a result, a set of readings can be transferred over the Local bus multiple times. The digitizer is configured for each additional data transfer with the command:

```
VINStrument[:CONFigure]:LBUS:MEMory:INITiate
```

Also, if a measurement is aborted with the ABORt command, the digitizer should be configured with VINStrument[:CONFigure]:LBUS:MEMory:INITiate for each subsequent data transfer.

Note Detailed information on the digitizers Local bus commands can be found in Chapter 4 - "Command Reference".

The Digitizer Status Registers

This chapter describes the HP E1429 digitizer status system. Included is information on the STATus subsystem commands, the status groups used by the digitizer, the conditions monitored by each group, and information on how to enable a condition to interrupt a computer.

The Status Subsystem Commands

The commands included in the STATus subsystem are:

```
STATus
    :OPC
        :INITiate <state>
    :OPERation|QUEStionable
        [:CONDition]?
        :ENABle <unmask>
        [:EVENt]?
        :NTRansition <unmask>
        :PTRansition <unmask>
    :PRESet
```

Status System Registers

Operating conditions within the digitizer are monitored by registers in various status groups. The status groups implemented by the digitizer are:

- Questionable Signal Status Group
  - condition register
  - transition filter
  - event register
  - enable register
- Operation Status Group
  - condition register
  - transition filter
  - event register
  - enable register
- Standard Event Status Group
  - standard event status register
  - standard event status enable register
- Status Byte Status Group
  - status byte register
  - service request enable register

The relationship between the registers and filters in these groups is shown in Figure 3-16.

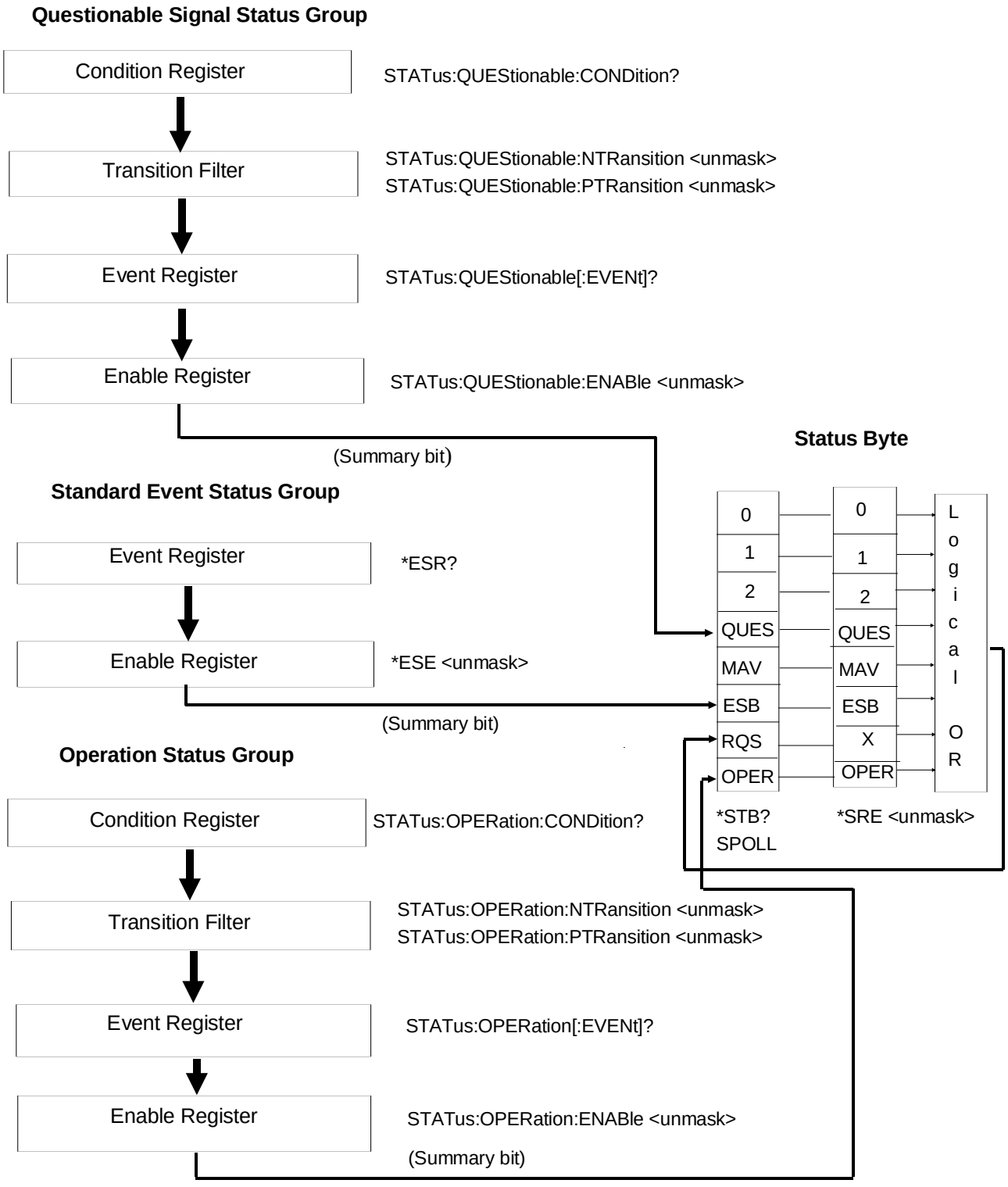


Figure 3-16. HP E1429 Status Groups and Associated Registers.

The Questionable Signal Status Group

The digitizer's Questionable Signal status group monitors overload conditions, the frequency accuracy of the divide-by-n reference source, and error conditions in non-volatile calibration memory.

The Condition Register

Overload conditions, divide-by-n frequency accuracy, and non-volatile calibration memory errors are monitored with the following bits in the Condition register. All other bits are unused.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
unused							CAL	unused					TIME		VOLT

VOLTage: Bit 0 is set (1) if an amplifier overrange (single-ended or differential input) is detected during a measurement sequence. Otherwise, the bit remains cleared (0).

TIME: Bit 2 is set (1) when the divide-by-n reference source cannot generate a sample rate that is within 1% of the rate specified by TRIG:TIMer1 or TRIG:TIMer2. Otherwise, the bit remains cleared (0).

CALibration: Bit 8 is set (1) when an error is detected in non-volatile calibration memory.

Reading the Condition Register

The current state of bits 0, 2, and 8 can be determined by reading the Condition register with the command:

STATus:QUEStionable:CONDition?

Bit 0 has a corresponding decimal value of 1, bit 2 has a decimal value of 4, and bit 8 has a decimal value of 256. Reading the Condition register does not affect the bit settings. The bits are cleared following a reset (*RST). Bit 8 CALibration will remain set, however; if the error condition persists.

The Transition Filter

The Transition Filter specifies which type of bit transition in the Condition register will set corresponding bits in the Event register. Transition filter bits may be set for positive transitions (0 to 1), or negative transitions (1 to 0). The commands used to set the transitions are:

STATus:QUEStionable:NTRansition <*unmask*>

STATus:QUEStionable:PTRansition <*unmask*>

NTRansition sets the negative transition. For each bit unmasked, a 1-to-0 transition of that bit in the Condition register sets the associated bit in the Event register.

PTRansition sets the positive transition. For each bit unmasked, a 0-to-1 transition of that bit in the Condition register sets the associated bit in the Event register.

<*unmask* > is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Condition register bit to be unmasked. (The decimal values of bits 0, 2, and 8 are 1, 4, and 256.)

The Event Register

The Event register latches transition events from the Condition register as specified by the Transition Filter. Bits in the Event register are latched and remain set until the register is cleared by one of the following commands:

STATus:QUEStionable[:EVENt]?

*CLS

The Enable Register

The Enable register specifies which bits in the Event register can generate a summary bit which is subsequently used to generate a service request. The digitizer logically ANDs the bits in the Event register with bits in the Enable register, and ORs the results to obtain a summary bit.

The bits in the Enable register that are to be ANDed with bits in the Event register are specified (unmasked) with the command:

STATus:QUEStionable:ENABle <*unmask* >

<*unmask* > is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Enable register bit to be unmasked. (The decimal values of bits 0, 2, and 8 are 1, 4, and 256.)

The Enable register is cleared at power-on, by specifying an <*unmask* > value of 0, or by executing the STATus:PRESet command.

The Operation Status Group

The Operation status group monitors current operating conditions within the digitizer. The specific conditions include: CALibrating, MEASuring, entering the wait-for-arm state, and execution of the INITiate[:IMMediate] command.

The Condition Register

Calibration, waiting for an arm signal, execution of the INITiate:IMMediate command, and memory partitioning are monitored with the following bits in the Condition register. All other bits are unused.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
unused						READY	BUSY		ARM	unused					CAL

CALibrating: Bit 0 is set (1) during calibration. The bit is cleared (0) otherwise.

Waiting for ARM: Bit 6 is set (1) when the digitizer enters the wait-for-arm state. The bit is cleared (0) when a start arm is received or when the measurements are aborted.

BUSY: Bit 8 is set (1) when the INITiate[:IMMediate] command is executed. The bit is cleared (0) when the measurements are complete or aborted, and the digitizer returns to the idle state.

READY: Bit 9 is set(1) when a digitizer memory segment is ready for data storage. The bit is cleared (0) while the digitizer is partitioning the next memory segment.

Reading the Condition Register

Bit settings in the Condition register can be determined with the command:

STATus:OPERation:CONDition?

Bits 0, 6, 8, and 9 have corresponding decimal values of 1, 64, 256, and 512. Reading the Condition register does not affect the bit settings. The bits are cleared following a reset (*RST).

The Transition Filter

The Transition Filter specifies which type of bit transition in the Condition register will set corresponding bits in the Event register. Transition filter bits may be set for positive transitions (0 to 1), or negative transitions (1 to 0). The commands used to set the transitions are:

STATus:OPERation:NTRansition <*unmask* >

STATus:OPERation:PTRansition <*unmask* >

NTRansition sets the negative transition. For each bit unmasked, a 1-to-0 transition of that bit in the Condition register sets the associated bit in the Event register.

PTRansition sets the positive transition. For each bit unmasked, a 0-to-1 transition of that bit in the Condition register sets the associated bit in the Event register.

<*unmask* > is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Condition register bit to be unmasked. (Bits 0, 6, 8, and 9 have corresponding decimal values of 1, 64, 256, and 512.)

The Event Register

The Event register latches transition events from the Condition register as specified by the Transition Filter. Bits in the Event register are latched and remain set until the register is cleared by one of the following commands:

STATus:OPERation[:EVENt]?

*CLS

The Enable Register

The Enable register specifies which bits in the Event register can generate a summary bit which is subsequently used to generate a service request. The digitizer logically ANDs the bits in the Event register with bits in the Enable register, and ORs the results to obtain a summary bit.

The bits in the Enable register that are to be ANDed with bits in the Event register are specified (unmasked) with the command:

STATus:OPERation:ENABle <*unmask* >

<*unmask* > is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Enable register bit to be unmasked. (Bits 0, 6, 8, and 9 have corresponding decimal values of 1, 64, 256, and 512.)

The Enable register is cleared at power-on, by specifying an <*unmask* > value of 0, or by executing the STATus:PRESet command.

The Standard Event Status Group

The Standard Event status group monitors command execution errors, programming errors, and the power-on state.

The Standard Event Status Register

The conditions monitored by the Standard Event Status register are identified below.

7	6	5	4	3	2	1	0
PON	unused	CME	EXE	DDE	QYE	unused	OPC

Power-on (PON): Bit 7 is set (1) when an off-to-on transition has occurred.

Command Error (CME): Bit 5 is set (1) when an incorrect command header is received, or if an un-implemented common command is received.

Execution Error (EXE): Bit 4 is set (1) when a command parameter is outside its legal range.

Device Dependent Error (DDE): Bit 3 is set (1) when an error other than a command error, execution error, or query error has occurred.

Query Error (QYE): Bit 2 is set (1) when the digitizer output queue is read and no data is present, or when data in the output queue has been lost.

Operation Complete (OPC): Bit 0 is set (1) when the *OPC command is received. *OPC is used to indicate when all pending (or previous) digitizer commands have completed.

Note that bits 7, 5, 4, 3, 2, and 0 have corresponding decimal values of 128, 32, 16, 8, 4, and 1.

Reading the Standard Event Status Register

The settings of the Standard Event Status register can be read with the command:

*ESR?

The bits are cleared at power-on, or by *ESR? or *CLS.

The Standard Event Status Enable Register

The Standard Event Status Enable register specifies which bits in the Standard Event Status register can generate a summary bit which is subsequently used to generate a service request. The digitizer logically ANDs the bits in the Event register with bits in the Enable register, and ORs the results to obtain a summary bit.

The bits in the Enable register that are to be ANDed with bits in the Event register are specified (unmasked) with the command:

*ESE <*unmask* >

<*unmask* > is the decimal, hexadecimal (#H), octal (#Q), or binary (#B) value of the Enable register bit to be unmasked. (Bits 7, 5, 4, 3, 2, and 0 have corresponding decimal values of 128, 32, 16, 8, 4, and 1.)

All unmasked bits in the Enable register can be determined with the command:

*ESE?

The Standard Event Status Enable register is cleared at power-on, or with an <*unmask* > value of 0.

The Status Byte Status Group

The registers in the Status Byte Status Group enable conditions monitored by the other status groups to generate a service request.

The Status Byte Register

The Status Byte register contains the summary bits of the Questionable Signal Status Group (QUES), the Operation Status Group (OPER), and the Standard Event Status Group (ESB). The register also contains the message available bit (MAV) and the service request bit (RQS).

7	6	5	4	3	2	1	0
OPER	RQS	ESB	MAV	QUES	unused		

Questionable Signal Summary Bit (QUES): Bit 3 is set (1) when a condition monitored by the Questionable Signal Status Group is present, when the appropriate bit is latched into the group's Event register, and when the bit is unmasked by the group's Enable register.

Message Available Bit (MAV): Bit 4 is set (1) when data, such as a query response, is in the digitizer's output queue.

Standard Event Summary Bit (ESB): Bit 5 is set (1) when a condition monitored by the Standard Event Status Group is present and the appropriate bit is set in the group's Event register, and when the bit is unmasked by the group's Enable register.

Service Request Bit (RQS): Bit 6 is set (1) when any other bit in the Status Byte register is set.

Operation Status Summary Bit (OPER): Bit 7 is set (1) when a condition monitored by the Operation Status Group is present, when the appropriate bit is latched into the group's Event register, and when the bit is unmasked by the group's Enable register.

Reading the Status Byte Register

The Status Byte register can be read with either of the following commands:

```
*STB?
SPOLL
```

Both commands return the decimal weighted sum of all set bits in the register. The difference between the commands is that *STB? does not clear bit 6 (RQS service request). The serial poll (SPOLL) does clear bit 6. All bits in the Status Byte register with the exception of MAV are cleared with the command:

```
*CLS
```

MAV is cleared when data is read from the output queue.

The Service Request Enable Register

The Service Request Enable register specifies which (status group) summary bit(s) will send a service request message to the computer over HP-IB. The bits are specified (unmasked) with the command:

```
*SRE <unmask>
```

All unmasked bits in the Enable register can be determined with the command:

```
*SRE?
```

The Service Request Enable register is cleared at power-on, or by specifying an <*unmask* > value of 0.

Presetting the Enable Register and Transition Filter

The Enable registers and Transition Filters in the Questionable Signal and Operation Status Groups can be preset (initialized) with the command:

```
STATus:PRESet
```

All bits in the Enable registers are masked (i.e. <*unmask* > is 0), and all bits in the Condition registers set corresponding bits in the Event registers on positive (0-to-1) transitions.

Synchronizing the Digitizer

One method of synchronizing the digitizer with other digitizer's or with other instruments is to determine when the digitizer parser is idle. This is done using the command:

STATus:OPC:INITiate <*state* >

together with the commands:

*OPC
*OPC?
*WAI

The behavior of *OPC, *OPC?, and *WAI depends on the *state* set by STATus:OPC:INITiate. When *state* is ON, sending *OPC, *OPC?, or *WAI following INITiate:IMMediate, VINStrument:CONFigure:VME:MEMory:INITiate, or VINStrument:CONFigure:LBUS:MEMory:INITiate requires the digitizer to complete all measurements or complete the VME or Local bus data transfer before allowing further digitizer operations to continue. ON is the power-on *state*.

When *state* is OFF, the execution of *OPC, *OPC?, or *WAI following the above commands indicates the previous commands have executed and that further operations (command execution) can resume without the digitizer in the idle state. Digitizer commands that can be executed during this time are identified in the command reference (Chapter 4) as being "executable when initiated". Refer also to the command reference for detailed information on *OPC, *OPC?, and *WAI.

Saving Digitizer Configurations

To minimize repeated programming, up to 10 digitizer configurations can be saved and later recalled. The configuration saved is restored, with the exception of the STATus subsystem parameters, the CALibration:SECure command state, or any other parameters not affected by *RST.

A configuration is identified by a number from 0 thru 9. The configuration(s) is saved until power is cycled.

How to Save and Recall a Configuration

Digitizer configurations are saved and recalled with the commands:

*SAV <*register*>
*RCL <*register*>

where *register* is a number from 0 to 9. The following example shows how a configuration can be saved and recalled.

```
CONF1:ARR:VOLT (10),10,(@3)      /* set 10 readings on (+) Diff port 3 */
    ARM:SOUR1 INT1               /* set arm source 1 for level arming */
    ARM:SOUR2 HOLD               /* disable arm source 2 */
    ARM:SLOP1 EITH               /* arm when signal enters window from */
                                 /* either direction */
    ARM:LEV1:POS 4               /* set arm window lower boundary */
    ARM:LEV1:NEG 6               /* set arm window upper boundary */
    ARM:COUN 2                   /* set two reading bursts */
*SAV 0                           /* save the configuration in location 0 */
*RST                             /* reset  the digitizer */
*RCL 0                           /* recall the configuration */
```

This program saves a configuration in register 0. The digitizer is then reset in order to change the current configuration to the power-on configuration. The configuration in register 0 is recalled which also leaves the digitizer in the idle state. By placing the digitizer in the wait-for-arm state (with READ? or INIT), readings are taken when the arm occurs and when trigger signals are received.

Notes

Chapter 4
Command Reference

Chapter Contents

This chapter describes the **Standard Commands for Programmable Instruments** (SCPI) command set and the **IEEE 488.2 Common Commands** for the HP E1429A/B 20 MSa/s 2-Channel Digitizer. Included in this chapter are the following sections:

Command Types

Commands are separated into two types: IEEE 488.2 Common Commands and SCPI Commands.

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 shown below:

*RST, *CLS, *ESE <unmask >, *OPC?, *STB?

SCPI Command Format

The HP E1429 digitizer is programmed with SCPI commands. SCPI commands are based on a hierarchical structure, also known as a tree system. In this system, associated commands are grouped together under a common node or root, thus, forming subtrees or subsystems. An example is the digitizer's 'ARM' subsystem shown on the following page.

```
ARM
    [:STARt | :SEQuence[1]]
        :COUNt <number >
        :DELay < period >
        [:IMMediate]                    [no query]
        :LEVel<n >
            :NEGative < voltage >
            :POSitive < voltage >
        :SLOPe[< n >] <edge >
        :SOURce[< n >] <source >
```

ARM is the root keyword of the command; :STARt|SEQuence1 is the second level keyword; :COUNt, :DELay, ... are third level keywords, and so on.

Keyword Separator

A colon (**:**) always separates one command keyword from a lower level command keyword as shown below:

```
ARM:LEV:NEG  2
```

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, you may want to send the entire command. The digitizer will accept either the abbreviated form or the entire command.

For example, if a command's syntax contains the keyword COUNt, then COUN and COUNT are acceptable forms. Other forms of COUNt such as COU will generate an error.

You can use upper or lower case letters. Therefore, COUNT, coun, or Coun are all acceptable.

Implied (Optional) Keywords

Implied or optional keywords are those which appear in square brackets ([ ]). The brackets are not part of the command syntax and must not be sent to the digitizer. Consider the syntax ARM[:STARt]COUNt. Suppose you send the following command:

ARM:COUN 100

In this case, the digitizer responds as if you had executed the command as:

ARM:STAR:COUN 100

Variable Command Syntax

Some commands will have what appears to be a variable syntax. For example:

OUTPut:ECLTrg<*n* >[:STATe] <*mode* >

In this command, <n > is replaced by a number. No space is left between the keyword (ECLTrg) and the number because the number is part of the keyword.

SCPI Command Parameters

The following information contains explanations and examples of the parameter types found 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. 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. Any non-zero value is considered true:

  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[:STARt]:SOURce <source> command where source can be BUS, ECLTrg0, ECLTrg1,EXTernal1, EXTernal2, ...

- String

  STRING PROGRAM DATA parameters are enclosed with single quotation marks (’) or double quotation marks ("). Examples of string program data parameters are those associated with the OUTPut...FEED commands:

  OUTPut:ECLTrg<n >:FEED <source >

  OUTPut:EXTernal[1]:FEED <source >

  OUTPut:TTLTrg<n >:FEED <source >

  Some of the <source > parameters include:

  "EXTernal[1]"
  "SENSe[1|2]:ROSCillator"
  "SENSe:SWEep:OFFSet:POINts"

  As an example, the syntax for sending this type of command in an HP BASIC program is:

  OUTPUT 70905;"OUTP:ECLT0:FEED ’EXT’"

  or

  OUTPUT 70905;"OUTP:ECLT0:FEED ""EXT"""

  In a C language program, the syntax is:

  "OUTP:ECLT0:FEED ’EXT’"

  or

  "OUTP:ECLT0:FEED \"EXT\""

- Arbitrary Block Program Data

  This parameter type is used to transfer a block of data in the form of bytes. The block of data bytes is preceded by a header which indicates the number of data bytes which follow. The syntax of the block header is as follows:

Definite length block:

#<non-zero digit><digit(s)><data byte(s)>

Where the value of <non-zero digit> equals 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.

Optional Parameters

Command parameters shown within square brackets (**[ ]**) are optional. The brackets are not part of the syntax and are not sent to the digitizer. If you do not specify a value for an optional parameter, the instrument chooses a default value.

For example, consider the command

CONFigure[<*chan* >]:ARRay[:VOLTage][:DC] (<*size* >)
[,<*expected value* >[,<*resolution* >]] [,(@<*input port* >)]

If you send the command without specifying the *expected value, resolution*, or *input port* parameters, the digitizer sets default values. Similarly, if *chan* in the command keyword is not specified, the default channel 1 is used.

Querying Parameter Settings

Unless otherwise noted in the subsystem syntax, parameter settings can be queried by adding a question mark (?) to the command. For example:

TRIG:SOUR HOLD

sets the trigger source to HOLD. The value can be queried by executing:

TRIG:SOUR?

The MINimum or MAXimum value of a parameter is determined by adding the word MIN or MAX to the end of the query. See below.

SENS:SWE:OFFS:POIN? MIN
SENS:SWE:OFFS:POIN? MAX

The minimum and maximum values returned are based on the current settings of other digitizer parameters.

SCPI Command Execution

The following information should be remembered when executing SCPI commands.

Command Coupling

Some of the digitizer SCPI commands are functional or value coupled. Functionally coupled commands are those that for one command to have affect, another command must be set to a particular value. A value coupled command changes the settings of other commands.

Command couplings can often result in “Settings conflict” errors when the program executes. When a coupled command is executed, the command setting is evaluated by the digitizer processor. If the setting causes an illegal digitizer configuration, a "Settings conflict" error occurs. The error message lists the conflicting settings, and then reports the values set by the digitizer processor.

The "Comments" section of each command entry indicates if a command is coupled, and if it is, what the coupling constraints are.

MIN and MAX Parameters in Coupled Commands

When MINimum or MAXimum is a command parameter in a group of coupled commands, that command should be the last command executed. Unlike other parameters that are set when an end-of-line indication is received, MIN and MAX are evaluated by the digitizer processor when the command is parsed. Thus, the value of MIN or MAX is based on the values of the other (coupled) commands at that time. For example, if the following commands are sent:

```
ARM:COUN 1
TRIG:COUN MAX
ARM:COUN 128
```

A “Settings conflict” error will occur when ARM:COUN 128 is executed because the trigger count value set by TRIG:COUN MAX (524,288) is based upon the ARM:COUN 1 setting. For an arm count of 128, the maximum trigger count allowed is 4,096. Because of these types of interactions, MIN and MAX are **not recommended** for specifying a parameter value.

Executable When Initiated Commands

In the "Comments" section of each command listing is the item "**Executable when initiated: Yes/No**". This identifies the command as being executable when the digitizer is in the INITiated state as the result of one of the following commands:

```
INITiate:IMMediate
 VINStrument:CONFigure:VME:MEMory:INITiate
VINStrument:CONFigure:LBUS:MEMory:INITiate
```

Linking Commands

Linking commands means sending multiple commands in the same output string. This is done to avoid "settings conflict" errors since (command) coupling interactions are not resolved until the carriage return (CR) is received.

Linking IEEE 488.2 Common Commands

Use a semicolon between the commands. For example:

```
*RST;*CLS;*OPC?
```

Linking Multiple SCPI Commands

A semicolon (;) is used to separate commands within the same subsystem and saves typing. For example, sending this command message:

```
TRIG:SOUR TIM;TIM1 100E-9;COUN 100
```

Is the same as sending these three commands:

```
TRIG:SOUR TIM
TRIG:TIM1 100E-9
TRIG:COUN 100
```

When linking commands in different subsystems, a semicolon (;) and a colon (:) are used. For example:

```
ARM:SOUR IMM;COUN10;:TRIG:SOUR TIM;TIM1 100E-9;COUN 30000
```

Command Choices

Some commands are listed as two commands separated with a vertical bar ("|"). This means that either command name can be used. For example, you could use either ":STAR" or ":SEQ1" when "[:STARt |SEQuence1]" is shown.

SCPI Command Reference

This section describes the SCPI commands for the HP E1429 Digitizer. Commands are listed alphabetically by subsystem, and alphabetically within each subsystem. A command guide is printed in the top margin of each page. The guide indicates the first command listed on that page.

ABORt

The ABORt command removes the HP E1429 from the wait-for-trigger state and places it in the idle state, irrespective of any other settings. Measurement is halted and can only be restarted by another INITiate[:IMMediate] command.

There is no query form of this command.

Comments

- ABORt does not affect any other settings of the HP E1429.
- Both measurement channels are aborted with this command.
- Local bus and VME bus data transfers are aborted.
- The Pending Operation Flag (as defined by IEEE-488.2) will be set false as a consequence of entering the idle state. *OPC? will therefore return 1 after an ABORt command.
- **Executable when initiated:** Yes
- **Coupled Command:** No
- If an ABORt or power failure occurs during a sequence of measurements, the digitizer will return (FETCh?) or recover (FETCh:RECover?) between one and TRIGger:STARt:COUNt number of readings. Because the digitizer processor does not know when the arm occurs, the readings returned may be pre-arm only, post-arm only, or a combination of both. If less than TRIGger:STARt:COUNt readings have been taken, then that number of readings are returned. If at least TRIGger:STARt:COUNt readings have been taken, then TRIGger:STARt:COUNt readings are returned.
- **Related Commands:** INITiate, *OPC, *OPC?
- ***RST Condition:** After a *RST, the HP E1429 is placed in the trigger idle state, as if ABORt had been executed.

Example

Aborting a measurement

```
ABOR                    Place HP E1429 in idle state
```

The ARM command subsystem controls the third state in a four state measurement process. The four states which occur during a successful reading are idle, initiated, wait-for-arm, and wait-for-trigger. The last two states have event detection associated with them which control when they exit the current state. These four states are more fully described as follows:

- Idle -- In this state, the instrument is not sampling. This is the state where setting changes are done via commands to the instrument. This state is exited when an INITiate command is received. This state is returned to after a reset, after successful completion of measurement, or after a measurement is aborted.

- Initiated -- Once the instrument is initiated with the INITiate command, it passes through this state, and continues down to the wait-for-arm state if all readings are post-arm and ARM:COUNt is not yet satisfied. If pre-arm readings are specified, the digitizer passes through to the wait-for-trigger state.

- Wait-for-arm -- In this state, the instrument waits for the specified ARM event to occur before progressing to the wait-for-trigger state to make a measurement.

- Wait-for-trigger -- In this state, the instrument waits for the specified trigger event to occur, and when it occurs, a reading is taken. After a reading is taken, the cumulative number of readings taken is compared to the count specified in TRIGger:COUNt or SENSe:SWEep:POINts. When the count is reached, the state is exited, otherwise, the instrument waits for another trigger and takes another reading. Upon exit from this state, the instrument returns to the initiated state and checks to see whether or not ARM:COUNt is satisfied. If the arm count is reached, the instrument returns to the idle state. If not, another loop is executed by entering the wait-for-arm state.

The following controls can be specified from the ARM subsystem:

- The event(s) which will cause the transition out of the wait-for-arm state (ARM:SOURce1 and ARM:SOURce2). There is a rich set of event sources to choose from, and when both sources are enabled (neither set to source HOLD), a logical OR of the two sources occurs. The occurrance of the appropriate event on either source will cause the HP E1429 to exit from the wait-for-arm state.

- The number of start arm events to occur before the digitizer returns to the idle state (ARM:COUNt). Another way to think of this is the number of bursts of readings which will occur.

- The active edge for generation of an arm event (ARM:STARt:SLOPe<*n* >) on the selected arm source (n = 1 or n = 2).

- The measurement signal level to attain before allowing a measurement to begin (ARM:STARt:LEVel<*n*>). This is used with ARM:SOURce INTernal<*n*>. It is also possible to create a window bounded by two levels (ARM:LEVel:POSitive and ARM:LEVel:NEGative) such that arming occurs when the signal level either exits or enters the defined window.

- The additional delay between the arm event and entry into the wait-for-trigger state (ARM:DELay).

Subsystem Syntax

```
ARM
    [:STARt |:SEQuence[1]]
        :COUNt <count>
        :DELay <period>
        [:IMMediate]                [no query]
        :LEVel<chan >
            :NEGative <voltage>
            :POSitive <voltage>
        :SLOPe[<n>] <edge>
        :SOURce[<n>] <source>
```

[:STARt]:COUNt

ARM[:STARt]:COUNt <***count***> specifies how many measurement cycles will occur after an INITiate, before the trigger system returns to the idle state. Each post-arm measurement cycle begins when the event specified by the active arm source(s) occurs. Multiple readings may be taken during each cycle, as defined by the TRIGger:STARt:COUNt or SENSe:SWEep:POINts commands.

ARM:STARt:COUNt values greater than 1 in conjunction with pre-arm readings is a special case which causes memory to be partitioned. Partitioning is done because a large number of pre-arm readings may be taken before the arming event, while a smaller number of pre-arm readings will actually be kept for read back. Therefore, for each ARM:STARt:COUNt specified, a separate circular buffer (partition) is set up to hold the specified number of "desired total readings" (TRIGger:STARt:COUNt). With this scheme, pre-arm data may overwrite itself until the arming event occurs, at which time, the post-arm data count will be completed by overwriting the oldest pre-arm data. Due to this complexity, the ARM:COUNt for this type of measurement is restricted to a much lower value than what can be specified if all data is post-arm only.

ARM[:STARt]:COUNt

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
count	numeric	1 through 65535 | MINimum|MAXimum| 9.9E+37|INFinity	none
MINimum selects 1 arm. When only post-arm readings are specified, MAXimum is computed as: 524,288 / TRIGger:COUNt, up to a maximum of 65,535 arms . When pre-arm readings have been specified, MAXimum selects 128 arms. 9.9E+37 is equivalent to INFinity.			

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. This command is coupled to the total reading count (TRIGger:STARt:COUNt). An error will result if TRIGger:STARt:COUNt is too large for the specified ARM:STARt:COUNt. See the following table for the relationship between ARM:STARt:COUNt and TRIG:STARt:COUNt.
- If the count is set to INFinity or 9.9E+37, the ABORt command must be used to return the trigger system to the idle state.
- If pre-arm readings are specified (SENSe:SWEep:OFFSet:POINts < 0), then the maximum number of arms is 128.
- When pre-arm readings are specified (SENSe:SWEep:OFFSet:POINts < 0), an ARM:STARt:COUNt > 1 causes memory to be partitioned, and limits to be placed on TRIGger:STARt:COUNt. See the following table for limits.

ARM:STARt:COUNt	Number of Memory Segments	Maximum Readings (TRIGger:STARt:COUNt)
1	1	524,288
2	2	262,144
3 - 4	4	131,072
5 - 8	8	65,536
9 - 16	16	32,768
17 - 32	32	16,384
33 - 64	64	8,192
65 - 128	128	4096
NOTE: If the non-volatile mode of memory is enabled (MEMory:BATTery:STATe ON), then all of the maximum reading counts shown above decrease by four. These four memory locations in each segment hold the data necessary to recover all readings after a power failure.		

- Normally, when only post-arm readings are specified, an error will occur if ARM:COUNt * TRIG:COUNt exceeds memory size, since post-arm data would be overwritten if this were allowed. There are two exceptions to this rule.

    a. The first exception is when the local bus is enabled (VINS:LBUS:MODE GEN), and VINStrument:LBUS:FEED is the A/D converter ("CONVerter:xxx"). In this case, an ARM:COUNt * TRIG:COUNt greater than memory size is allowed. Readings sent out directly over the local bus are also routed to memory at the same time, and counts greater than memory size simply overwrite older data in memory. Thus a FETCh? command will return a historical record of what was sent over the local bus in the last "memory size" transfers. This block of readings returned will be in chronological order, with the oldest readings first and the most recent readings last.

    b. The second exception is when either TRIGger:STARt:COUNt INF or ARM:STARt:COUNt INF has been specified. It is assumed in this case that the user knows data is being overwritten. After the ABORt is done to stop the measurement, a FETCh? command would bring back the entire memory contents, with the most recent readings at the end of the block of data.

ARM[:STARt]:DELay

- **Related Commands**: ABORt, INITiate:IMMediate, TRIGger:STARt:COUNt, SENSe:SWEep:POINts, SENSe:SWEep:OFFSet:POINts
- ***RST Condition:** ARM:STARt:COUNt 1

Example **Setting the arm count**

```
ARM:COUN 10          Set 10 measurement cycles per INITiate
```

[:STARt]:DELay

ARM[:STARt]:DELay <*period*> specifies how long to delay entering the wait-for-trigger state after the arming pulse is received and processed. Delays of greater than 0 can be specified only if no pre-arm readings are being taken with the post-arm readings (i.e. SENSe:SWEep:OFFSet:POINts is 0).
The delay time is generated using either the reference oscillator period or ten times the reference oscillator period. If we designate the current reference oscillator period as T, then the two ranges of delay can be expressed as:

0 to 65534T in steps of T
65540T to 655350T in steps of 10T

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
period	numeric	MINimum|MAXimum| value(*see below*)	Seconds
MINimum selects 0 delay. MAXimum selects 655350 reference oscillator periods delay. The above values bound the valid range for *period*. The *period* value is rounded to the nearest period that can be produced.			

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. This command is coupled to SENSe:SWEep:OFFSet:POINts and to SENSe:ROSCillator:SOURce.
- If pre-arm readings are specified, then delay will be forced to 0, regardless of what value was specified for delay. A settings conflict error will also occur.

- The actual delay time after receipt of arm event is not precisely the delay time specified above. There is an inherent delay of typically 230 ns plus an amount of sample period uncertainty. Refer to Appendix A - "Specifications" for additional information on the inherent delay.
- **Related Commands:** TRIGger:STARt:COUNt, SENSe:SWEep:OFFSet:POINts
- ***RST Condition:** ARM:STARt:DELay 0

Example **Setting the arming delay**

```
ARM:DEL .001
```
Delay 1 millisecond after arm pulse before arming the start trigger

[:STARt][:IMMediate]

ARM[:STARt][:IMMediate] will cause the start trigger to be armed immediately, regardless of the selected ARM:STARt:SOURce. The selected ARM:STARt:SOURce remains unchanged. The INITiate:IMMediate command must have been sent before this command, otherwise error -212,"Arm ignored" will occur.

There is no query form of this command.

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- If the instrument is in the idle or wait-for-trigger states, the ARM:STARt:IMMediate command will cause error -212,"Arm ignored" to be generated.
- If ARM:STARt:COUNt is greater than 1, only a single measurement cycle is affected by ARM:STARt:IMMediate; the count is decremented by one, and the remaining ARM:STARt:COUNt cycles will be executed with the original arming source active.
- When ARM:STARt:IMMediate is sent, any ARM:STARt:DELay is bypassed for that arm only.
- **Related Commands:** INITiate:IMMediate, ARM:STARt:COUNt, TRIGger subsystem
- ***RST Condition:** none

ARM[:STARt]:LEVel[<*chan*>]:NEGative <*voltage*>

Example **Arming for measurement**

ARM:SOUR1 EXTernal	*Set arming source1 to be the EXT1 BNC connector*
INITiate	*Begin measurement, wait for arming pulse*
ARM	*Arm start trigger immediately, don't wait for external pulse*

[:STARt]:LEVel[<*chan*>]:NEGative <*voltage*>

ARM[:STARt]:LEVel[<*chan*>]:NEGative <*voltage*> selects the input voltage level which will arm a measurement cycle. The ARM:LEVel:NEGative setting is used **only** when either of the ARM:STARt:SOURce(s) is set to INTernal1 or INTernal2, and the corresponding ARM:STARt:SLOPe<n> setting is NEGative or EITHer. The value programmed is retained (but not used) when other sources are selected, or when the corresponding ARM:STARt:SLOPe<n > setting is POSitive.

The allowable levels depend on the measurement range as set by the SENSe:VOLTage:RANGe command.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none
level	numeric	-102.2418 to 102.2418 MINimum | MAXimum	Volts

Comments

- **Executable while initiated:** No
- **Coupled Command:** Yes. The command is coupled to the ARM[:STARt]:SLOPe, ARM[:STARt]:SOURce, and SENSe:VOLTage:RANGe commands. The level value is limited to the range limits.
- When POSitive and NEGative voltage levels are specified, the levels must be separated by an amount defined by:

  (50 mV / 1.0235V) ∗ measurement range

  This accounts for offset errors in the levels specified and enables the levels to be detected.
- **Related Commands:** ARM:STARt:SOURce<n >, ARM:STARt:SLOPe<n >
- ***RST Condition:** ARM:STARt:LEVel<n >:NEG -1.022418

Example **Setup to arm only when the signal on channel 2 falls through -0.48 volts.**

ARM:SOUR1 INT2	*Set arming source 1 to arm when the specified level is met on channel 2*
ARM:SLOP1 NEG	*The signal must fall through the level set by ARM:LEVel2:NEG for arming to occur*
ARM:LEV2:NEG -0.48	*Arm when the signal passes through the -.48V level*

[:STARt]:LEVel[<*chan*>]:POSitive <*voltage*>

ARM[:STARt]:LEVel[<*chan*>]:POSitive <*voltage*> selects the input voltage level which will arm a measurement cycle. The ARM:LEVel:POSitive setting is used **only** when either of the ARM:STARt:SOURce(s) is set to INTernal1 or INTernal2, and the corresponding ARM:STARt:SLOPe<n> setting is POSitive or EITHer. The value programmed is retained (but not used) when other sources are selected, or when the corresponding ARM:STARt:SLOPe<n> setting is NEGative.

The allowable levels depend on the measurement range as set by the SENSe:VOLTage:RANGe command.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none
level	numeric	-102.2418 to 102.2418 MINimum | MAXimum	Volts

Comments

- **Executable while initiated:** No
- **Coupled Command:** Yes. The command is coupled to the ARM[:STARt]:SLOPe, ARM[:STARt]:SOURce, and SENSe:VOLTage:RANGe commands. The level value is limited to the range limits.

- When POSitive and NEGative voltage levels are specified, the levels must be separated by an amount defined by:

  (50 mV / 1.0235V) * measurement range

  This accounts for offset errors in the levels specified, and enables arms at those levels to be accepted.

- **Related Commands:** ARM:STARt:SOURce<n>, ARM:STARt:SLOPe<n>

- ***RST Condition:** ARM:STARt:LEVel<n >:POS 1.022418

Example **Setup to arm when the signal on channel 1 goes outside of a the window bounded by 0.5V and 0V.**

ARM:SOUR1 INT1	*Set arming source 1 to be channel 1 level(s).*
ARM:LEV1:POS 0.50	
ARM:LEV1:NEG 0	
ARM:SLOP1 EITH	*Specify that both ARM:LEVel1:POS and ARM:LEVel1:NEG will be used to form a window. If the signal rises above the 0.5 volt level (the POSitive slope level specified), or falls below the 0 volt level (the NEGative slope level specified), arming will occur and the measurement will proceed.*

[:STARt]:SLOPe[<*n* >]

ARM[:STARt]:SLOPe[<n >] <edge > selects which edge - POSitive, NEGative, or EITHer on arming source <n > will cause the arm event to occur. ARM:STARt:SLOPe is only active when either of the two arm sources is set to EXTernal1, INTernal1, or INTernal2. The "EITHer" setting may be used only when the corresponding ARM:STARt:SOURce is set to INTernal<n >. The "EITHer" setting causes the window specified by ARM:STARt:LEVel<chan>:POSitive and ARM:STARt:LEVel<chan>:NEGative to be in effect.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n	numeric	1|2	none
edge	discrete	NEGative|POSitive| EITHer	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. A settings conflict error occurs if ARM:STARt:SLOPe EITHer is set and ARM:STARt:SOURce EXTernal is set.The EITHer setting is only valid when the associated ARM:STARt:SOURce is set to INTernal<n >.
- ARM:STARt:SLOPe1 controls the slope on ARM:STARt:SOURce1 and ARM:STARt:SLOPe2 controls the slope on ARM:STARt:SOURce2. Note that '1' and '2' refer to the arm source and not a channel number.
- When ARM:STARt:SLOPe is EITHer, having a value of ARM:STARt:LEVel:POSitive greater than the value of ARM:STARt:LEVel:NEGative will cause arming to occur when the signal exits the defined window. If ARM:STARt:LEVel:NEGative is greater than ARM:STARt:LEVel:POSitive, then an arm event will occur when the signal enters into the defined window.
- The edge selected by ARM:STARt:SLOPe<*n* > is ignored when ARM:STARt:SOURce<*n* > is set to any source other than EXTernal1, INTernal1, or INTernal2. Other trigger sources have standardized active edges: NEGative for TTLTrg<*n*> or POSitive for ECLTrg<*n*>.
- **Related Commands:** ARM:STARt:SOURce<*n* >, ARM:STARt:LEVel<*chan* >:POSitive, ARM:STARt:LEVel<*chan*>:NEGative
- ***RST Condition:** ARM:STARt:SLOPe<n > POSitive

Example

Setting the arm slope

ARM:SOUR1 EXT1	*Set arming source1 to "Ext 1" BNC*
ARM:SLOP1 NEG	*Arm on the falling edge of source1*

ARM[:STARt]:SOURce[<*n*>]

[:STARt]:SOURce[<*n*>]

ARM[:STARt]:SOURce[<*n* >] <*source* > configures the arm system to respond to the specified source(s). Unless one of the two sources is set to HOLD, both will be active, and an occurrence of the selected event on either source will arm the system for measurement.

The sources available are:

- **BUS**: The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG common command.
- **ECLTrg0** and **ECLTrg1**: The VXIbus ECL trigger lines.
- **TTLTrg0** through **TTLTrg7**: The VXIbus TTL trigger lines.
- **EXTernal1**: The HP E1429's front panel "Ext 1" BNC connector.
- **INTernal[1]**: Arms the start trigger when the signal on channel 1 meets the conditions specified by ARM:STARt:LEVel1.
- **INTernal2**: Arms the start trigger when the signal on channel 2 meets the conditions specified by ARM:STARt:LEVel2.
- **HOLD**: Disable this arming source. If both sources are set to HOLD, then the ARM:STARt:IMMediate command must be sent before measurements will proceed.
- **IMMediate**: Arm the start trigger as soon as the INITiate:IMMediate command is received. This choice is only valid for ARM:STARt:SOURce1, and requires that ARM:STARt:SOURce2 be set to HOLD.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n	decimal	1|2	none
source	discrete	BUS|ECLTrg0| ECLTrg1|IMMediate| EXTernal1|INTernal[1]| INTernal2|HOLD TTLTrg0 to TTLTrg7	none
Choice IMMediate is only available for ARM:STARt:SOURce1.			

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. The command is coupled to the ARM[:STARt]:SLOPe[<n >], ARM[:STARt]:LEVel:NEGative, and ARM[:STARt]:LEVel:POSitive commands.
- The active edge for the EXTernal1 source is specified by the ARM:STARt:SLOPe[<*n* >] command.
- The active edge(s) for level detection (ARM:STARt:SOUR:LEVel<*chan* >) is specified by the ARM:STARt:SLOPe[<*n* >] command.
- **Related Commands:** ARM:STARt:SLOPe[<*n* >], ARM:STARt:LEVel[<*chan* >]
- ***RST Condition:** ARM:STARt:SOURce1 IMMediate, ARM:STARt:SOURce2 HOLD

Example

Setting two arm start sources

ARM:SOUR1 EXT1	*Set one arming source to be the front panel "Ext 1" BNC.*
ARM:SOUR2 ECLT0	*Set the second arming source to be the ECLTRG line.*

CALibration[<*chan*>]

The CALibration command subsystem is used in the calibration of the HP E1429. The HP E1429 has commands to prevent and detect accidental or unauthorized calibration of the instrument. The CALibration subsystem includes both these security-related commands and the actual calibration commands.

Subsystem Syntax

```
CALibration[<chan >]
    :COUNt?                                    [query only]
    :DATA<block data >
    :DELay                                     [no query]
    :GAIN [<readings >[,<period >]]            [no query]
    :SECure
        :CODE <code >                          [no query]
        :STATe <mode >[,<code >]
    :STORe
        :AUTO <mode >
    :VALue <number >
    :ZERO [<readings>[,<period >,[<mode >]]]   [no query]
```

:COUNt?

CALibration[<*chan* >]:COUNt? returns a number that shows how often the HP E1429 has been calibrated. Since executing the CAL:GAIN, CAL:ZERO, or CAL:STORe commands increments the number, the CALibration:COUNt? command detects any accidental or unauthorized HP E1429 calibration.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- Your HP E1429 was calibrated before it left the factory. Before using, read the calibration count to determine its initial value.
- Both channels share the same counter, so it does not matter whether channel 1 or channel 2 is specified; the same answer is returned for either.
- The HP E1429 stores the calibration number in its non-volatile calibration memory which remains intact even with power off.

- Executing the CALibration:GAIN, or CALibration:ZERO commands with calibration security disabled (CALibration:SECure:STATe OFF set) and with CALibration:STORe:AUTO ON, increments the calibration number by one. With CALibration:STORe:AUTO ON, a complete calibration of all input ranges increments the number by several counts. It is possible by setting CALibration:STORe:AUTO to OFF, to defer the storing of calibration constants until explicitly told to do so by invoking the CALibration:STORe command.

- The count increments whenever either channel stores calibration data to memory. The maximum value of the number is 2,147,483,647, after which it wraps around to 0.

- **Related commands:** CALibration:SECure:STATe, CALibration:GAIN, CALibration:ZERO, CALibration:STORe:AUTO, CALibration:STORe

- ***RST Condition:** unaffected

Example Querying the calibration count

```
CAL:COUN?                    Query calibration count, the count is shared
                             by both channels
```

:DATA

CALibration[<*chan* >]:DATA <*block data* > manually sets or queries the calibration constants. The query form of this command returns the calibration constants in IEEE-488.2 definite length arbitrary block format. The command (non-query) form is used to send calibration constants to the digitizer in indefinite or definite length arbitrary block format. The new calibration constants take effect immediately, but are not saved to non-volatile calibration memory unless the CALibration:STORe command is executed.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none
block data	IEEE 488.2 block data	-2046 to 2047	none

CALibration[<*chan*>]:DATA

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Sending calibration constants with this command will cause the HP E1429 to calibrate to these constants. A check is done on the values before usage and an error (-222 "Data out of range") results if they are not within a valid range.
- It is possible with this command for the user to have tables of calibration constants which are downloaded for usage whenever the load characteristics of what is connected to the front panel of the HP E1429 change. The storage to, and retrieval from memory of these tables would be under the control of the host controller, external to the HP E1429.
- The query form of this command will return the calibration constants that the digitizer is currently using; note that these may not be the same values which are stored in non-volatile calibration memory unless the CALibration:STORe command has been previously executed on these constants.
- **Related commands:** CALibration[<*chan*>]:STORe
- ***RST Condition:** none

Example 1 Sending an array of new cal constants (HP BASIC program)

```
ASSIGN @X TO 70905;FORMAT OFF
                                    Turn format off for array data
OUTPUT 70905 USING "#,K";"CAL:DATA #3124"
                                    Specify 124 bytes coming (62
                                    constants)
OUTPUT @X;Array(*),CHR$(10),END     Send the array of calibration constants
```

Example 2 Querying the calibration constants on channel 2 (HP BASIC program)

```
DIM Ndig$[1],Count$[9]              Dimension parameters for header
ASSIGN @To TO 70905                 I/O path to digitizer
ASSIGN @From TO 70905;FORMAT OFF    I/O path from digitizer. Turn format off for
                                    array data
OUTPUT @To;"FORM PACK"              Set packed data format
OUTPUT @To;"CAL2:DATA?"             Query for calibration data
ENTER @From USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                    Strip the header preceeding the data
ALLOCATE INTEGER Cal_data(1:VAL(Count$)/2)
                                    Allocate an array to hold the data
ENTER @From;Cal_data(*)             Read in the calibration constants
ENTER @To USING "B";Junk            Need to remove left over line feed
```

Each channel contains 62 calibration constants. The following list describes what the constant is at each array index location. The array is assumed to start with index number 0.

Index	Contents	Index	Contents
0	offset for 1.0235V range	31	gain lsb for single-ended .51175 V range
1	A to D chip internal setting	32	offset for differential .10235 V range
2	A to D chip internal setting	33	gain msb for differential .10235 V range
3	A to D chip internal setting	34	gain lsb for differential .10235 V range
4	A to D chip internal setting	35	offset for differential .2047 V range
5	A to D chip internal setting	36	gain msb for differential .2047 V range
6	linearity bit 5 left	37	gain lsb for differential .2047 V range
7	linearity bit 5 right	38	offset for differential .51175 V range
8	linearity bit 6 left	39	gain msb for differential .51175 V range
9	linearity bit 6 right	40	gain lsb for differential .51175 V range
10	linearity bit 7 left	41	offset for differential 1.0235 V range
11	linearity bit 7 right	42	gain msb for differential 1.0235 V range
12	linearity bit 8 left	43	gain lsb for differential 1.0235 V range
13	linearity bit 8 right	44	offset for differential 2.047 V range
14	linearity bit 9 left	45	gain msb for differential 2.047 V range
15	linearity bit 9 right	46	gain lsb for differential 2.047 V range
16	linearity bit 10 left	47	offset for differential 5.1175 V range
17	linearity bit 10 right	48	gain msb for differential 5.1175 V range
18	gain msb	49	gain lsb for differential 5.1175 V range
19	gain lsb	50	offset for differential 10.235 V range
20	conversion delay adjust	51	gain msb for differential 10.235 V range
21	trigger level negative	52	gain lsb for differential 10.235 V range
22	trigger level positive	53	offset for differential 20.47 V range
23	offset for single-ended .10235 V range	54	gain msb for differential 20.47 V range
24	gain msb for single-ended .10235 V range	55	gain lsb for differential 20.47 V range
25	gain lsb for single-ended .10235 V range	56	offset for differential 51.175 V range
26	offset for single-ended .2047V range	57	gain msb for differential 51.175 V range
27	gain msb for single-ended .2047 V range	58	gain lsb for differential 51.175 V range
28	gain lsb for single-ended .2047 V range	59	offset for differential 102.35 V range
29	offset for single-ended .51175 V range	60	gain msb for differential 102.35 V range
30	gain msb for single-ended .51175 V range	61	gain lsb for differential 102.35 V range

:DELay

CALibration[<*chan* >]:DELay will calibrate the delay constant for the A to D converter. Both channels are calibrated with this command, regardless of which <chan> value is specified. This calibration only needs to be done once or twice in the lifetime of the instrument. Also, this calibration determines a nominal value for convert time at room temperature (25 degrees C). It is, therefore, important that this command be executed at an ambient temperature near 25 degrees C.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none

CALibration[<*chan*>]:GAIN

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Before executing this command, both channels must be set to the single-ended setting and 1.0235 volt range. This can be accomplished by executing either *RST or the combination of CONF:ARR:VOLT (1),1.0,(@1) with CONF2:ARR:VOLT (1),1.0,(@2).
- This setting was calibrated before the HP E1429 left the factory. Under normal conditions, it is not necessary to execute this command again. The symptom of needing this calibration is that the fastest sample rate will appear to be 10 MHz instead of 20 MHz, especially at higher ambient operating temperatures (such as 50 or 60 degrees C).
- Both channels are calibrated with a single call to this command, so a single call will be sufficient to calibrate.
- If CALibration:STORe:AUTO is ON, then the new settings will be stored to non-volatile calibration ram. Calibration security must also be turned off for the new constants to be permanently stored in non-volatile calibration ram.
- CALibration:COUNt will be incremented with this command when the values are stored to non-volatile calibration RAM.
- **Related commands:** CALibration:SECure:STATe, CALibration:STORe:AUTO, CALibration:STORe
- ***RST Condition:** unaffected

Example

Querying the calibration delay

```
CAL:DEL?                    Query calibration delay, the delay is shared
                            by both channels
```

:GAIN

CALibration[<*chan* >]:GAIN [<*readings* >[,<*period* >[,<*flag* >]]] performs a calibration for gain using the specified number of readings and sample rate. The CALibration:VALue voltage specified is used as the full scale value to calibrate to, and must be between 85.0 and 99.5 percent of the full scale reading for the current configuration. The 99.5 percent upper limit is to insure that noise will not cause erroneous full scale (overload) readings. A linearity calibration is also done on the 1V measurement range with the single ended port (0 or 2). This linearity calibration may be disabled by setting the *flag* parameter to OFF. Omitting the optional *flag* parameter will cause linearity to be performed.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none
readings	numeric	100 to 32767 |DEFault	none
period	numeric	reference period to reference period * 4E8 |DEFault	Seconds
flag	boolean	ON | OFF | defaults to ON	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Before executing CALibration[<*chan*>]:GAIN you must do the following steps. Note that if you use the CONFigure command, it must be sent first; otherwise several of the settings such as CALibration:SECure:STATe and CALibration:VALue will be reset to their *RST values.

  a. Disable the calibration security by setting CALibration[<*chan*>]:SECure:STATe OFF.

  b. Use the CONFigure command, or the SENSe:VOLTage:RANGe and SENSe:FUNCtion commands to place the HP E1429 in the desired range and using the desired port: i.e CONF<*channel*>:ARR:VOLT (1000),<*range*>,(@<*port*>).

  c. Program the input filter and impedance (if applicable) to the desired settings (use the INPut:FILTer and INPut:IMPedance commands).

  d. Specify the voltage which will be applied using the CALibration:VALue command. This voltage must be 85% to 99.5% of the full scale input for the range being calibrated. The CONFigure command will reset the value of this setting, so this step must be done after any CONFigure command is used.

  e. Connect a fixed standard DC voltage to the input to be calibrated, where the standard is between 85.0% and 99.5% of the full scale reading for the range being calibrated.

- MINimum and MAXimum are not allowed with this command.
- Optional parameters that are left blank are filled from left to right. Therefore, it may be necessary to use the syntax DEFault to note that a particular parameter has been defaulted. For example, to default the number of readings and specify a

sample rate, the command would appear as:

CAL:GAIN DEF, .05, one

- CALibration:GAIN forces the internal reference (20 MHz) oscillator to be used. Sample rates are attained using that reference.
- The default number of readings is 1000, and the default period is 1.0E-4 seconds. These numbers were chosen such that the product of the two is a period that is an integral multiple of both 50 Hz and 60 Hz line cycles (.1 seconds in this case).
- When calibrating gain on the differential ports, an error could occur which contains the text "All readings have same value in cal_mean routine". The most likely cause of this error is that the two differential inputs on the port are not grounded properly, and a common mode overload is occurring.
- The product of the period and number of readings will be checked to see if it exceeds 10 seconds, and if so, error -221;"Settings conflict; Calibration time too long" occurs.
- Normally, upon completion the new gain values would be stored to the non-volatile calibration memory. This can be overridden by setting CALibration:STORe:AUTO OFF, in which case the new gain values will be stored to calibration memory only when the CALibration:STORe command is executed.
- **Related commands:** CALibration:VALue, CALibration:SECure:STATe, CALibration:STORe:AUTO, CALibration:STORe
- ***RST Condition:** none

Example Performing a gain calibration

CONF:ARR:VOLT (100),4.8,DEF,(@1)	*Configure for 5 volt range. If CONF is used, it must be the first step because it performs a soft reset of most other settings.*
CAL:SEC:STAT OFF,E1429	*Disable security, assuming factory-set security code*
CAL:STOR:AUTO OFF	*Disable automatic storage of calibration constants*
CAL1:VAL 5.05	*Set value to > 85% of positive full scale on 5 volt range*
CAL1:GAIN DEF,DEF	*Calibrate channel 1 for gain using default sample rate and number of points. Note that linearity will not be done because this is not the 1 volt range.*
CAL1:STOR	*Force the gain settings just calculated to be stored into calibration RAM.*

:SECure:CODE

CALibration[<*chan* >]:SECure:CODE <*code* > sets the code which is required to disable calibration security. Calibration security must have been previously disabled.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none
code	character data	1 to 12 characters	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- The calibration security code must begin with a letter, and can contain letters, digits, and underscores. Lower case letters are converted to upper case.
- The calibration code is shared by both channels. It does not matter which channel is specified (if any), the same code is shared by both and gives access to calibrate either.
- If calibration security has not been previously disabled by CALibration[<*chan*>]:SECure:STATe OFF, the HP E1429 generates the error 311,"Calibration security on". To disable the calibration security requires knowledge of the previous security code.
- Before shipping, the factory sets the calibration security code to E1429. You should change it before you use your HP E1429 to prevent unauthorized calibration. Record the new security code and store in a secure place. If you forget the new code, defeating the security involves instrument disassembly.
- The HP E1429 stores the security code in its non-volatile calibration memory which remains intact even with power off.
- **Related commands:** CALibration[<*chan*>]:SECure:STATe
- ***RST Condition:** unaffected

Example

Changing the factory-shipped security password

```
CAL:SEC:STAT OFF,E1429      Disable security for both channels
CAL:SEC:CODE NEWCODE        Set new security code for both channels
CAL:SEC ON                  Re-enable security on both channels
```

:SECure:STATe

CALibration[<*chan* >]:SECure:STATe <*mode* >,[<*code* >] enables or disables calibration security. Disable the calibration security to calibrate the HP E1429, change the security code, or change the protected user data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none
mode	boolean	OFF|0|ON|1	none
code	character data	1 to 12 characters	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- Either channel may be referenced, enabling/disabling is done to both channels at once with this command.
- The *code* parameter must be present to disable the security, or it generates error -109,"Missing parameter". The value supplied must match the currently programmed security code or it generates error -224,"Illegal parameter value". A 1 second delay will then occur before the HP E1429 executes any subsequent commands.
- To enable security, the *code* parameter is *not* required, but is checked if it is present. If a code is given and is incorrect, error -224, "Illegal parameter value" will be generated.
- Security must be disabled to calibrate the HP E1429, or to use the *PUD command.
- **Related commands:** CALibration:GAIN, CALibration:ZERO, CALibration:STORe, CALibration:SECure:CODE, *PUD
- ***RST Condition:** unaffected

Example

Disabling calibration security

```
CAL:SEC:STAT OFF,E1429          Disable security, assuming factory-set
                                security code
```

:STORe

CALibration[<*chan* >]:STORe stores the current calibration constants into non-volatile calibration memory.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- CALibration:SECure:STATe must be OFF before executing this command.
- **Related commands:** CALibration[<*chan*>]:DATA, CALibration:SECure:STATe, CALibration:COUNt?, CALibration:STORe:AUTO
- ***RST Condition:** none

Example

Sending and storing an array of new cal constants

```
ASSIGN @X TO 70905;FORMAT OFF
                                        Turn format off for array data
OUTPUT 70905 USING "#,K";"CAL:DATA #3124"
                                        Specify 124 bytes coming (62
                                        constants)
OUTPUT @X;Array(*),CHR$(10),END         Send the array of calibration constants
OUTPUT 70905;"CAL:SEC:OFF,E1429"
                                        Disable security
OUTPUT 70905;"CAL:STOR"                 Store the calibration data in non-volatile
                                        RAM
```

:STORe:AUTO

CALibration[<*chan* >]:STORe:AUTO <mode > selects whether or not the calibration constants will be automatically stored when commands like CALibration:GAIN and CALibration:ZERO complete.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none
mode	boolean	ON|1|OFF|0	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- Enabling CALibration[<chan >]:STORe:AUTO for either channel enables it for both channels.
- **Related commands:** CALibration:GAIN, CALibration:SECure:STATe, CALibration:ZERO, CALibration:STORe
- ***RST Condition:** CALibration:STORe:AUTO ON

Example

Turn automatic storage of calibration values off

```
CAL:STOR:AUTO OFF          Disable automatic storage
```

:VALue

CALibration[<*chan* >]:VALue <*number* > specifies the voltage level supplied at the input. This voltage value is then used in subsequent CALibration:GAIN or CALibration:ZERO commands. The HP E1429 can not distinguish between a value which is exactly full scale, and one which is an overload -- both cases would generate the same measured value. Therefore, the voltage specified for CALibration:VALue is not allowed to be closer than 10 counts from full scale (approximately 99.5% of full scale). The following table shows the allowable CALibration:VALues which are closest to full scale on their respective voltage ranges.

Maximum Gain Calibration Values (V)	Voltage Range (Volts)	Allowable Ports
-.10180 and .10185	0.10235	1,2,3,4
-.2036 and .2037	0.2047	1,2,3,4
-.5090 and .50925	0.51175	1,2,3,4
-1.0180 and 1.0185	1.0235	1,2,3,4
-2.036 and 2.037	2.037	3,4
-5.090 and 5.0925	5.1175	3,4
-10.180 and 10.185	10.235	3,4
-20.360 and 20.370	20.37	3,4
-50.900 and 50.925	51.175	3,4
-101.80 and 101.85	102.35	3,4
< -48.925 and > 48.975	102.35	3,4

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
number	numeric	-101.80 to 101.85	volts

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Calibration values on the 102.35 volt range have a special low end allowed, so that voltages much less than full scale may be used to calibrate gain on this range. Values < -48.975 and > 48.975 are accepted as legal values for calibrating the 102.35 volt range.
- CALibration:VALues used for GAIN calibration are at least 85% of full scale and ≤ 10 counts away from absolute full scale. The previous table lists the gain calibration values.
- **Related commands:** CALibration:GAIN
- ***RST Condition:** CALibration<chan >:VALue 1.0185

CALibration[<*chan*>]:ZERO

Example **Setting the calibration value**

```
CAL2:VAL 5.00          Input on channel 2 is 5.00 V
```

:ZERO

CALibration[<*chan* >]:ZERO [<*readings* >[,<*period* >,[<*mode* >]]] performs a calibration of the zero offset using the specified number of readings and sample rate on the specified range(s). When this command completes, the new calibration constants will be automatically stored to non-volatile calibration memory unless the CALibration:STORe:AUTO command is set to OFF.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 or 2	none
readings	numeric	100 to 32767 |DEFault	none
period	numeric	reference period to reference period * 4E8 |DEFault	Seconds
mode	discrete	ALL |ONE	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Before executing CALibration[<*chan*>]:ZERO, you must do the following steps.

  a. If *mode* is not ALL, use the CONFigure command or the SENSe:VOLTage:RANGe and SENSe:FUNCtion commands to place the HP E1429 in the desired range and using the desired port: i.e CONF<*channel*>:ARR:VOLT (1000),<*range*>,(@<*port*>).

  b. Program the input filter and impedance (if applicable) to the desired settings (use the INPut:FILTer and INPut:IMPedance commands).

  c. If it is desirable for the new constants to be stored to non-volatile calibration memory upon completion of the CALibration:ZERO command, then set CALibration:STORe:AUTO to ON, and turn calibration security OFF. If a complete calibration involving gain and linearity is to be done, then it may be more desirable to only store the calibration constants when everything is complete; for this case, set CALibration:STORe:AUTO to OFF, and use the CALibration:STORe command to force storage when all calibrations are complete.

- MINimum and MAXimum are not allowed with this command.
- Optional parameters that are left blank are filled from left to right. Therefore, it may be necessary to use the syntax DEFault to note that a particular parameter has been defaulted. For example, to default the number of readings and specify a sample rate, the command would appear as:

  CAL:ZERO DEF, .05, one
- CALibration:ZERO forces the internal reference (20 MHz) oscillator to be used. Sample rates are attained using that reference.
- The default number of readings is 1000, and the default period is 1.0E-4 seconds. These numbers were chosen such that the product of the two is a period that is an integral multiple of both 50 Hz and 60 Hz line cycles (0.1 seconds in this case).
- The product of the period and number of readings will be checked to see if it exceeds 10 seconds, and if so, error -221;"Settings conflict; Calibration time too long" occurs.
- The default <*mode*> is ONE, which calibrates using the current settings of SENSe:VOLTage:RANGe and SENSe:FUNCtion. Specifying <*mode*> ALL will do a zero calibration on all voltage range settings for *both* of the ports on the specified channel.
- **Related commands:** CALibration:GAIN, CALibration:VALue, CALibration:SECure:STATe
- ***RST Condition:** none

Example Performing a zero calibration

CAL:SEC:STAT OFF,E1429	*Disable security, assuming factory-set security code*
CAL1:ZERO DEF,DEF,ALL	*Calibrate channel 1 using default sample rate and number of points, calibrate all gain ranges on both ports.*

The CONFigure subsystem provides a fast way to place the HP E1429 into a known state, ready to take measurements. An INITiate;FETCh? is then all that is necessary to take a measurement after the CONFigure command has been executed. If desired, CONFigure may be used to quickly get to a known state, from which other states such as TRIGger:STARt:SOURce or TRIGger:STARt:TIMer can be "fine tuned" to desired values before the INITiate;FETCh? sequence is executed.

Subsystem Syntax

CONFigure[<*chan*>]
:ARRay
[:VOLTage]
[:DC] (<*size*>)[,<*expected value*>[,<*resolution*>]] [,(@<*input port*>)]

:ARRay:[VOLTage][:DC]

CONFigure[<*chan*>]:ARRay[:VOLTage][:DC] (<*size*>)[,<*expected value*>[,<*resolution*>]] [,(@<*input port*>)] will configure for taking <*size*> number of readings on the specified channel and <*input port*>. The <*expected value*> and <*resolution*> parameters are used to set SENSe:VOLTage:RANGe to an appropriate setting for making the measurement on the specified input port and channel. If no expected value is given, the 1V range (1.0235 V peak) is used. If a resolution is specified, it is checked for correctness against what is possible with the expected value range, and if too fine for the given expected value, error -231,"Data questionable;CONF or MEAS unable to attain resolution specified" will occur.

Each channel consists of two ports; one port is single ended, and the other is differential. The two ports on channel 1 have odd numbers: port 1 is the single ended input and port 3 is the differential input. On channel 2, the single ended input is port 2 and the differential input is port 4.

The expected value parameter specified should be the ***maximum*** expected measurement value. The voltage range is set according to the expected value supplied. If the expected value is greater than 98% of a given range, the next higher range is automatically chosen. The table under the "Settings" heading gives the crossover points for range changes.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none
size	numeric	1 | 7 to 524288	none
expected value	numeric	-102.30 to 102.35 | DEFault | MINimum | MAXimum	volts
resolution	numeric	.00005 to .05 | DEFault | MINimum | MAXimum	volts
input port	numeric	1|3 with CONFigure1 2|4 with CONFigure2 1 and 2 are single ended 3 and 4 are differential	none
The maximum size parameter will be 524284 if the battery is enabled instead of 524288.			
For expected value, MINimum selects the 0.100 V range and MAXimum selects either the 1.0 volt range (single ended ports) or the 100 V range (differential ports). DEFault selects the 1V range.			
For resolution, MINimum, MAXimum, and DEFault select the same value.			

Settings

Maximum expected value settings per range are shown in the following table, along with the resolution associated with each range. The highlighted area shows the setting used when the expected value is not specified or DEFault is used.

Maximum Expected Value (V)	Voltage Range (Volts)	Resolution (Volts)
± .1	0.10235	.00005
± .2	0.2047	.00010
± .5	0.51175	.00025
± 1	1.0235	.0005
± 2	2.047	.0010
± 5	5.1175	.0025
± 10	10.235	.005
± 20	20.47	.010
± 50	51.175	.025
±100	102.35	.05

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- If no input port is given, CONFigure1 defaults to (@1) and CONFigure2 defaults to (@2).
- Resolution varies by range, and is constant for each range. For any given range, specifying MINimum, MAXimum, or DEFault for resolution yields the same result.
- It is important to note that the expected value determines the resolution and not vice versa. The digitizer always uses 12-bit resolution and a coarser resolution value has no effect. If, for some reason a coarser resolution is desired, specify a larger expected value. See the previous table for expected values and resolutions.

- The CONFigure command configures the HP E1429 to do the measurement specified by the parameters given with CONFigure. All instrument settings will be forced to their *RST values, with the following exceptions. In particular, note that all OUTPut signals are turned off and their FEED's are changed to reset values, and that both the local bus (VINStrument:LBUS) and VME bus (VINS:VME) are turned off. Be aware that these new states will still be in effect after the CONFigure command is complete. See Appendix B, Table B-2 for a complete list of the reset settings for the HP E1429. The following states are set up after the reset, and thus in most cases, will not be the *RST value for that command:

  a. SENSe<n>:FUNCtion is set to "VOLTage[:DC] <port>"

  b. SENSe<n>:VOLTage:DC:RANGe is set to the value implied by the expected value given.

  c. INPut[<*port*>]:FILTer is set to ON.

  d. SENSe:SWEep:POINts and TRIGger:STARt:COUNt are set to the <*size*> parameter.

- **Related commands:** SENSe:FUNCtion, SENSe:VOLTage:RANGe, SENSe:SWEep:POINts, SENSe:ROSCillator:SOURce, TRIGger:STARt subsystem, ARM:STARt subsystem.

- ***RST Condition:** none

Example **Measuring 1.5 V on the differential input of channel 1**

CONF:ARRay:VOLT (20),1.5,(@3)	*Set up to take 20 readings of 1.5 Volts peak on port 3 of channel 1.*
INIT	*Start the measurement*
FETC?	*Get channel 1 readings*

The DIAGnostic subsystem contains several commands which were developed to test the instrument at the factory. Several of these commands may prove useful for trouble shooting or special applications, and so they are written up here.

Subsystem Syntax

```
DIAGnostic
    :CALibration[<chan >]
        :CONVerge                                   [query only]
        :GAIN
            :SENSitivity                            [query only]
        :ZERO
            :SENSitivity                            [query only]
    :CHANnel[<chan >]
        :LABel <label >
    :FETCh[<chan >] <start_addr >, <count >         [query only]
    :MEMory[<chan >]
        :FILL <num_segments >, <count >
        :ADDresses?                                 [query only]
    :PEEK <address >, <bits >                       [query only]
    :POKE <address >, <bits >
    :SGET <bit >                                    [query only]
    :SPUT <bit >, <value>
    :TEST?                                          [query only]
```

:CALibration[<chan >]:CONVerge?

DIAGnostic:CALibration[<*chan* >]:CONVerge? returns a real array containing convergance data from the latest CAL:ZERO or CAL:GAIN command. This array contains pairs of data points, with the first data point in a pair being the average of the readings taken, and the second data point is the new calibration constant which will be used on the next convergance iteration. The channel number passed in is irrelevant, as the data brought back always pertains to the most recently attempted CAL:GAIN or CAL:ZERO.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	numeric	1 | 2	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No

:CALibration[<chan >]:GAIN:SENSitivity?

DIAGnostic:CALibration[<*chan* >]:GAIN:SENSitivity? returns a real number which is the sensitivity constant calculated for use during the most recently executed CAL:GAIN command. The channel parameter is irrelevant because the data returned always applies to the last executed CAL:GAIN command regardless of which channel was involved.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	numeric	1 | 2	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No

:CALibration[<chan >]:ZERO:SENSitivity?

DIAGnostic:CALibration[<*chan* >]:ZERO:SENSitivity? returns a real number which is the sensitivity constant calculated for use during the most recently executed CAL:ZERO command. The channel parameter is irrelevant because the data returned always applies to the last executed CAL:ZERO command regardless of which channel was involved.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	numeric	1 | 2	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No

:CHANnel[<chan >]:LABel

DIAGnostic:CHANnel[<*chan* >]:LABel <*label* > puts the bit pattern specified into the lower 4 bits of all data on the specified channel. These four bits are ignored when data is fetched or read with FORMat:DATA ASCii or REAL, but they are sent with FORMat:DATA is PACKed, and they are transmitted out over the local bus or VME bus when data is sent out via those routes. This command could be

useful for tagging data which is going out over the local bus or VME bus when multiple channels or HP E1429's are involved.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	numeric	1 | 2	none
label	numeric	0 through 15| MINimum | MAXimum	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- **Related Commands:** ABORt, FETCh?
- ***RST Condition:** DIAG:CHAN1:LAB 0, DIAG:CHAN2:LAB 0

Example 1 Tagging the data with channel number

```
DIAG:CHAN1:LAB 1        Make the lower 4 bits of all channel 1 data
                        contain the bit pattern 0001
DIAG:CHAN2:LAB 2        Make the lower 4 bits of all channel 2 data
                        contain the bit pattern 0010
```

:FETCh?

DIAGnostic:FETCh[<*chan* >]? <*start_addr* >, <*count* > returns count number of readings starting with the one at start_addr. The data is returned in PACKed format (block of 16 bit integers).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	numeric	1 | 2 | 3	none
start_addr	numeric	0 through 524287	none
count	numeric	1 through 2000	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No

- Refer to the "Memory Management" section in Chapter 3 for information on where segmented readings are stored in memory.

- Both channels can be fetched interleaved by specifying DIAG:FETC3?. When both channels are fetched, the channel 1 data is the first data point, and the count specified is the number of readings to be taken on each channel, and is thus limited to 1000 per channel.

- The start_addr must be divisible by 4. If it is not, then the next lowest memory location divisible by 4 will be the actual start_addr used, i.e. a start_addr of 511 would be rounded down to be 508.

- Data is sent back in binary block format with a header string preceeding the data. The header is made of the ascii string: #,number_of_digits,number_of_bytes -- where # indicates binary block data, number_of_digits is how many ascii digits make up the following byte count, and number_of_bytes is the byte count. After the header, the stream of data bytes occurs. Example 2 below shows how to accomodate the header string using the HP Rocky Mountain Basic programming language.

- **Related Commands:** ABORt, FORMat, INITiate:IMMediate, ARM:STARt:COUNt, TRIGger:STARt:COUNt, SENSe:SWEep:POINts:DELay

- ***RST Condition:** none

Example 1 Examining readings in a portion of memory

```
DIAG:FETC? 5200,300          Get 300 readings from channel 1, beginning
                             at address 5200.
```

Example 2 Reading back PACKed data (HP BASIC program)

```
DIM Ndig$[1],Count$[9]                    Dimension parameters for header
ASSIGN @X TO 70905;FORMAT OFF
                                          Turn format off for array data
OUTPUT 70905;"DIAG:FETC1? 5200,300"       Query for channel 1 measurement data
ENTER @X USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                          Strip the header preceeding the data
ALLOCATE INTEGER Meas_data(1:VAL(Count$)/2)
                                          Allocate an array to hold the data
ENTER @X;Meas_data(*)                     Read in the measurement data
ENTER 70905 USING "B";Junk                Need to strip off left over line feed
```

:MEMory[<chan>]:FILL

DIAGnostic:MEMory[<chan >]:FILL <***num_segments, count***> sets up num_segments in memory and fills each with count readings. Each data point is calculated from the formula:

data<n> = (((current_segment - 1) * 10 + n) modulo 2000) * 16

Where n is the data point of interest between 1 and count. Only the specified channel is filled with these data points, the other channel is filled with reading values of 0.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
channel	numeric	1 | 2	none
num_segments	numeric	0 thru 128	none
count	numeric	1 thru 524288	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- ***RST Condition:** none

:MEMory[<chan>]:ADDResses?

DIAGnostic:MEMory[<chan >]:ADDResses? returns a binary block of integers which comprise a list of memory address data for each segment in memory. If there is no valid data, then an error occurs and no values are returned; otherwise, a 32 bit value is returned for each segment in memory containing valid data. Bits 0 and 1 contain flags for segment wrapped and segment aborted respectively; the next 19 bits are the final value of the address counter (A0 to A18) when the segment was completed or aborted, and bits 21 through 31 are filled with 0. Be aware that the address counter value returned is actually 1 greater than the location of the last data point taken in the segment.

Comments

- **Executable when initiated:** No
- **Coupled Command:** No

- The wrapped bit (bit 0) usually indicates that enough pre-arm data was taken to cause data to be overwritten, since each segment is a circular buffer. It is possible for a wrapped bit to be set even though no data was actually overwritten. This occurs because the address counter always points to the next location in memory that is to be filled, and therefore a false wrap indication will occur if exactly "buffer size" data points were taken. The buffer size is a number divisable by 4 which for post-arm only measurements is ARM:COUN * TRIG:COUN (padded to a multiple of 4), or TRIG:COUN (padded to a multiple of 4) for pre-arm with post-arm measurements.

- ***RST Condition:** none

Example Querying the memory segment address(es), HP BASIC Program

```
DIM Ndig$[1],Count$[9]                    Dimension parameters for header
ASSIGN @X TO 70905;FORMAT OFF
                                          Turn format off so we can enter
                                          unformatted bytes with this path
OUTPUT 70905;"DIAG:MEM:ADDR?"             Query for memory addresses
ENTER @X USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                          Obtain the header information
                                          preceeding the data
ALLOCATE INTEGER Mem_addrs(1:VAL(Count$)/2)
                                          Allocate an array to hold the data.
                                          Note that HP BASIC's integers are 16
                                          bits and not 32, hence the divide by 2
                                          instead of by 4
ENTER @X;Mem_addrs(*)                     Read in the memory addresses
ENTER 70905 USING "B";Junk                Need to strip off left over line feed
```

:PEEK?

DIAGnostic:PEEK? <***address, bits***> shows the specified number of bits from the memory location specified.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
address	numeric	0 thru $FFFFFF_{16}$	none
bits	numeric	8 | 16 | 32	none

DIAGnostic:POKE

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- The specified address is assumed to be relative to the local processor and is **not** the A24 offset, but is instead the full address (i.e. $E0000C_{16}$ would specify the data register, as opposed to the VME A24 offset of C_{16}).

Example **Examining the setting of the traffic register**

```
DIAG:PEEK? #HE00002,8          Retrieve the contents of the 8 bit traffic register
```

:POKE

DIAGnostic:POKE <*address, bits, value*> places the specified value into the memory location specified.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
address	numeric	0 thru $FFFFFF_{16}$	none
bits	numeric	8 | 16 | 32	none
value	numeric	-2147483648 thru 2147483647	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- The specified address is assumed to be relative to the local processor and is **not** the A24 offset, but is instead the full address (i.e. $E0000C_{16}$ would specify the data register, as opposed to the VME A24 offset of C_{16}).

Example **Changing the contents of the traffic register**

```
DIAG:POKE #HE00002,8,#H4B      Set traffic register to value of 4B hex
```

:SGET?

DIAGnostic:SGET? <*bit* > returns the state (0 or 1) of the specified bit of the serial control register. This register is the one which controls things such as signal conditioning, signal routing, input filter state, and input impedance.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
bit	numeric	0 through 55	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No

:SPUT

DIAGnostic:SPUT <*bit, value* > changes the state (0 or 1) of the specified bit of the serial control register to the value given. This register is the one which controls things such as signal conditioning, signal routing, input filter state, and input impedance.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
bit	numeric	0 through 55	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No

:TEST?

DIAGnostic:TEST? returns additional data on a failed self-test. The string returned may be up to 40 characters in length.

The FETCh? command is used in conjunction with the INITiate:IMMediate command to obtain readings. The FETCh:COUNt command is used to indicate how many readings are available.

Subsystem Syntax

```
FETCh[<chan>]?      [query only]
    :COUNt?         [query only]
    :RECover?       [query only]
```

FETCh?

FETCh[<*chan*>]? returns readings from the specified channel in the format set by the FORMat:DATA command. An INITiate:IMMediate command must have been issued previously or else error -230, "Data corrupt or stale" will occur.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none

Comments

- If measurements are still being taken when the FETCh? command is sent, the command will wait until readings have completed and will then return the data. Note that this could generate a deadlock error message if either the arming source or the trigger source is set to BUS, because a software trigger could not break in after FETCh? is sent.

- If desired, the negative transition of the BUSY bit (bit 8) in the STATus:OPERation:CONDition register can be used to determine when measurements have completed so a FETCh? could execute without delay or threat of deadlock. This status register bit can be enabled to generate an SRQ when readings are complete, and thus interrupt the controller. See the STATus subsystem in the Command Reference for more details.

- If error -230 "Data corrupt or stale" occurs and you must read the data, the FETCh:RECover? command can be used to force data read with no error.

- **FETCh?** or **FETCh1?** will return readings from channel 1 only. **FETCh2?** will return readings from channel 2 only.

- The data may be read any number of times, as long as no parameters have changed which could affect new data (such as changing TRIGger:STARt:COUNt, ARM:STARt:COUNt, SENSe:SWEep:OFFSet:POINts, *RST, *RCL, etc.). If any measurement parameters are changed, then error -230;"Data corrupt or stale" will occur.

- The number of readings that FETCh? is going to return for each channel can be determined two ways. The simplest way is to use the FETCh:COUNt? query. This will return how many readings will be fetched. The second way is to calculate how many readings should have been taken. This number is calculated as:

  (number of start triggers) * (pre-arm readings + post-arm readings)

  The above equation may be built from the following queries:

  (ARM:STARt:COUNt?) * (TRIG:STARt:COUNt?)

  The number of pre-arm readings may be determined by taking the absolute value of the query SENSe:SWEep:OFFSet:POINts? which returns either 0 (no pre-arm readings), or a negative number which is the pre-arm count.
  If the measurement was ABORted, then FETCh:COUNt? is the only reliable way to determine how many readings will be returned by the FETCh<chan >?

- As noted immediately above, it is possible to calculate the end of pre-arm data and the beginning of post-arm data when both are present in a measurement. The data will always be returned with the specified number of pre-arm readings followed by post-arm readings.

- If the measurement was aborted and pre-arm readings were specified, there is no way to determine how many readings (if any) are post-arm. The HP E1429 will attempt to bring back TRIGger:STARt:COUNt number of readings and if there are not that many, it will bring back as many readings as were taken.

- **Related Commands:** ABORt, FORMat:DATA, INITiate:IMMediate, READ?, ARM:STARt:COUNt, TRIGger:STARt:COUNt, SENSe:SWEep:OFFSet:POINts, STATus subsystem

- ***RST Condition:** none

Example 1 Obtaining readings from the HP E1429

CONF1:ARR:VOLT (30),.6,(@1)	*Configure for 30 readings on channel 1, port 1*
INIT	*Take the measurement*
FETC?	*Get readings from channel 1.*

Example 2 **Reading back PACKed data (HP BASIC program)**

```
DIM Ndig$[1],Count$[9]                    ! Dimension parameters for header
ASSIGN @X TO 70905;FORMAT OFF             ! Turn format off for array data
OUTPUT 70905;"FETC1?"                     ! Query for channel 1 measurement data
ENTER @X USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                          ! Strip the header preceeding the data
ALLOCATE INTEGER Meas_data(1:VAL(Count$)/2)
                                          ! Allocate an array to hold the data
ENTER @X;Meas_data(*)                     ! Read in the measurement data
ENTER 70905 USING "B";Junk                ! Need to strip off left over line feed
```

:COUNt?

FETCh[<*chan*>]:COUNt? returns the total number of readings stored in memory for the channel specified. The count is the same for both channels, so either channel may be queried.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none

Comments

- This command is most useful after an ABORt command has been issued on a measurement in progress. The number returned will be the number of readings which were taken before the ABORt command forced measurements to stop. If an infinite measurement was aborted, then the number returned will be the maximum memory size, if more than that number of readings were taken.
- **Related Commands:** ABORt, INITiate:IMMediate, ARM:STARt:COUNt, TRIGger:STARt:COUNt, SENSe:SWEep:OFFSet:POINts
- ***RST Condition:** none

Example

Determining how many readings are in memory

```
FETC1:COUN?                               ! Query how many readings are available for
                                          ! channel 1 -- this is also how many are
                                          ! available on channel 2.
```

:RECover?

FETCh[<*chan*>]:RECover? returns readings from the specified channel in the format set by the FORMat:DATA command. This command is used to fetch data that was saved during a power failure and which cannot be fetched by the FETCh<chan >? command due to the "Data corrupt or stale" error.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No.
- As mentioned above, this command is intended to retrieve battery backed data after a power failure. Attempting to use FETCh<chan >? in this case may cause an error if any measurement parameters were changed after power is returned. With FETCh:RECover? it is also possible to retrieve readings that were taken previously, but which are not accessable to FETCh? because of a setting change or reset.
- If error 1018 "Battery backed data corrupt" occurs, FETCh:RECover? will not return the readings since a problem with the data was detected. If the data **must** be recovered, use the DIAGnostic:FETCh? command.
- **Related Commands:** INITiate:IMMediate, ARM:STARt:COUNt, TRIGger:STARt:COUNt, SENSe:SWEep:OFFSet:POINts:
- ***RST Condition:** none

Example

Recovering readings in memory

```
FETC1:REC?
```

Do a fetch on the data on channel 1. The battery must be enabled for this to work.

The FORMat command subsystem is used to specify the output format of the readings from the HP E1429 Digitizer.

Subsystem FORMat

Syntax [:DATA] <*type*>*[,<length>]*

[:DATA]

FORMat[:DATA] <*type*>*[,<length>]* specifies the output format for measurement data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
type	discrete	ASCii|PACKed| REAL	none
length	numeric	9 (ASCii)|16 (PACKed)| 64 (REAL)	none

Comments

- PACKed format is signed 16 bits, however, the original 12-bit reading is shifted left 4 bits and the lower most 4 bits are 0 filled. Thus, to get the actual value of the reading, the packed value should be divided by 16. Packed readings always represent a value between -1.023 and +1.0235 (i.e. there is no correction done internally for the range setting), and further post-processing by the user is necessary to get the actual reading value if the SENSe<chan >:VOLTage:RANGe setting is not 1V.

- REAL,64 format sends data back as IEEE-754 64-bit real numbers. The data is converted internally to reflect the SENSe<chan >:VOLTage:RANGe setting, and no further conversion is needed by the user.

- Both PACKed,16 and REAL,64 formats return data preceded by the IEEE-488.2 definite length arbitrary block header. The header is as follows:

  # <num_digits > <num_bytes >

  # signifies a block transfer
  <num_digits > is a single digit (1 through 9) which specifies how many digits (ASCII characters) are in the <num_bytes > descriptor which follows
  <num_bytes > is the number of data bytes which immediately follow the <num_bytes > field

- ASCii,9 format sends data back as comma separated ASCII numbers. The data is converted internally to reflect the SENSe<chan >:VOLTage:RANGe setting, and no further conversion is needed by the user. The optional parameter 9 is the number of significant digits in the data. This parameter is always 9 and may not be changed. To read this data into an array, it is first necessary to determine how many readings are coming back from the FETCh? or READ? . If using FETCh?, then the FETCh:COUNt? command will specify how many readings will be returned. READ? requires the user to calculate in advance how many readings to expect. See the next comment below for details on how to calculate number of readings.

- The number of readings that FETCh? or READ? will return for each channel can be calculated as:

  (number of start triggers) * (pre-arm readings + post-arm readings)

  The above equation may be built from the following queries:

  (ARM:COUNt?) * (TRIGger:COUNt?)

- To determine how many readings are pre-arm, take the absolute value of the number returned by the SENSe:SWEep:OFFSet:POINts? query.

- **Related Commands:** READ?, FETCh?

- ***RST Condition:** FORMat:DATA ASCii,9

Example 1 Setting the data format to 64 bit reals

```
FORM REAL                    Output data in 64 bit real format, the ",64" is defaulted
```

Example 2 Reading back PACKed data (HP BASIC program)

```
DIM Ndig$[1],Count$[9]                          Dimension parameters for header
ASSIGN @X TO 70905;FORMAT OFF
                                                Turn format off for array data
OUTPUT 70905;"FETC1?"                           Query for channel 1 measurement data
ENTER @X USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                                Strip the header preceeding the data
ALLOCATE INTEGER Meas_data(1:VAL(Count$)/2)
                                                Allocate an array to hold the data
ENTER @X;Meas_data(*)                           Read in the measurement data
ENTER 70905 USING "B";Junk                      Need to strip off left over line feed
```

The INITiate subsystem controls the initiation of the trigger subsystem and prepares the HP E1429 to take voltage measurements. Once initiated, triggers are armed on both channels, and a trigger received from the programmed source (TRIGger:STARt:SOURce command) will cause voltage measurements to begin on both channels.

Normally, all measurement setup (setting measurement ranges, arm and trigger sources, etc.) should be done before this command is sent. Sending this command will cause the HP E1429 to begin the measurement process, using the currently active sources and settings.

Subsystem Syntax

INITiate
[:IMMediate] [no query]

[:IMMediate]

INITiate[:IMMediate] initiates the trigger system and places all trigger sequences in the wait-for-arm or wait-for-trigger state, as appropriate. When the number of readings specified by ARM:COUNt and TRIGger:COUNt have been taken, the trigger system returns to the idle state, and measurements are no longer taken.

This command is an overlapped command as described in IEEE-488.2, Section 12. The exit from idle state caused by INITiate:IMMediate shall cause the Pending Operation Flag to be set true (*OPC will return 0). The Pending Operation Flag will be set false when the idle state is re-entered, either when the trigger cycle completes or when an ABORt or *RST command is executed.

The STATus:OPC:INITiate command controls whether *OPC, *OPC? and *WAI will test the Pending Operation Flag and wait until it is false (trigger system in the idle state).

Readings may not be obtained from memory until the trigger system has returned to the idle state. The FETCh? and READ? commands may be invoked before the trigger system is in the idle state. These commands will wait until the trigger system is idle before returning readings. The trigger idle state may be determined externally from the high-to-low transition of the BUSY bit (bit 8) of the STATus:OPERation:CONDition register.

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Both measurement channels are affected by this command.
- If the trigger system is not in the idle state, error -123,"Init ignored" will be generated, and the trigger system will be unaffected.
- The ABORt command may be used to prematurely halt the trigger system and place the HP E1429A in the idle state.
- **Related Commands:** *OPC, *OPC?, *RST, *WAI, ABORt, ARM subsystem, STATus:OPC:INITiate, TRIGger subsystem
- ***RST Condition:** The trigger system is in the idle state.

Example

Placing the HP E1429A in the wait-for-arm state

```
INIT                                    Initiate signal measurement
```

INPut[<*port*>]

The INPut command subsystem controls characteristics of the input signal, including state (on/off), low-pass filtering, and input source impedance. There are two types of input ports on the HP E1429, single ended and differential, for a total of 4 input ports.

Since this is a two channel instrument, two input ports share each channel. The differential input ports are labeled ports 3 and 4. Input ports 1 and 2 are single ended. Ports 1 and 3 share channel 1, and ports 2 and 4 share channel 2.

Since IMPedance, FILTer, and STATe are settable for each input port, they are "remembered" so that changes made to FILTer on input port 3 for example, will not change the settings made previously on input port 1 or vice versa. Just note that the currently active settings may change when switching between input ports on a given channel.

The selection mechanism for specifying which of two ports is connected to a given channel is the SENSe<*chan*>:FUNCtion "VOLTage:DC <*port*>" command. See the SENSe:FUNCtion command for more information.

Note The input filter, impedance, and state can be changed while the HP E1429 is taking readings (initiated). Due to settling times associated with changing these parameters, readings taken during this period may have unexpected values.

Subsystem Syntax

```
INPut[<port>]
    :FILTer
        [:LPASs]
            [:STATe] <mode>
    :IMPedance <impedance>
    [:STATe] <mode>
```

:FILTer[:LPASs][:STATe]

INPut[<*port*>]:FILTer[:LPASs][:STATe] <*mode*> enables or disables the 10 MHz input filter.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
port	numeric	1|2|3|4	none
mode	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- Enabling or disabling the filter while the digitizer is taking readings (initiated) may cause unexpected reading values due to relay settling during the transition.
- Enabling the filter on an input port does not affect the input filter setting on the channel's other port.
- **Related commands:** SENSe:FUNCtion "VOLTage <*port*>"
- ***RST Condition:** INPut<port >:FILTer:LPASs:STATe OFF

Example

Enabling the 10 MHz low-pass filter

```
INP1:FILT:LPAS ON          Enable input filtering on input port 1
```

:IMPedance

INPut[<*port*>]:IMPedance <*impedance*> selects the input impedance for the HP E1429. Either 50 or 75Ω may be selected on input ports 1 and 2, while 1 MΩ is the only setting allowed for input ports 3 and 4. Values other than 50 or 75Ω on input ports 1 and 2 will generate error -222,"Data out of range". Similarly, attempts to set input ports 3 or 4 to an impedance value other than 1 MΩ will generate error -222,"Data out of range".

INPut[<*port*>][:STATe]

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
port	numeric	1|2|3|4	none
impedance	numeric	50|75|1.0E6|1MOHM| MINimum|MAXimum	Ohms
Input ports 1 and 2 : MINimum selects 50Ω; MAXimum selects 75Ω. Input ports 3 and 4 : MINimum selects 1 MΩ; MAXimum selects 1 MΩ.			

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- Changing the input impedance while the digitizer is taking readings (initiated) may cause unexpected reading values due to relay settling during the transition.
- **Related commands:** SENSe:FUNCtion "VOLTage <*port*>"
- ***RST Condition:** INPut1|2:IMPedance 50 OHM, INPut3|4:IMPedance 1 MOHM .

Example

Setting 75 Ω input impedance

```
INP:IMP 75 OHM          Set 75Ω impedance (defaults to input 1)
```

[:STATe]

INPut[<*port*>][:STATe] <*mode*> connects/disconnects the input ports to/from the measurement signal path. All ports have a special relay which is used to isolate the input from the signal path, regardless of whether the sensor channel is currently connected to that port or not.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
port	numeric	1|2|3|4	none
mode	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- Enabling or disabling the input port while the digitizer is taking readings (initiated) may cause unexpected reading values due to relay settling during the transition.
- **Related commands:** SENSe:FUNCtion "VOLTage:DC <*port*>"
- ***RST Condition:** INPut<port >:STATe ON

Example

Disabling input port 2

```
INP2 OFF                    Disconnect input port 2 from the
                            measurement signal path
```

The MEASure? query subsystem provides a complete measurement sequence, including configuration and reading of the data. MEASure? is used when the generic measurement is acceptable and default triggering and timebase values may be used. The MEASure? query is the same as doing the command sequence of ABORt;CONFigure;INITiate;FETCh?.

Subsystem Syntax

MEASure[<*chan*>]
:ARRay
[:VOLTage]
[:DC]? (<*size*>)[,<*expected value*>[,<*resolution*>]][,(@<*input port*>)]

:ARRay[:VOLTage][:DC]?

MEASure[<*chan*>]:ARRay[:VOLTage][:DC] (<*size*>)[,<*expected value*>[,<*resolution*>]][,(@<*input port*>)] will configure for taking <*size*> number of readings on the specified channel and <*input port*>. The <*expected value*> and <*resolution*> parameters are used to set SENSe:VOLTage:RANGe to an appropriate setting for making the measurement on the specified input port and channel. If no expected value is given, the 1V range (1.0235 V peak) is used. If a resolution is specified, it is checked for correctness against what is possible with the expected value range, and if too fine for the given expected value, error -231,"Data questionable;CONF or MEAS unable to attain resolution specified" will occur.

Each channel consists of two ports; one port is single ended, and the other is differential. The two ports on channel 1 have odd numbers: port 1 is the single ended input and port 3 is the differential input. On channel 2, the single ended input is port 2 and the differential input is port 4.

The expected value parameter specified should be the ***maximum*** expected measurement value. The voltage range is set according to the expected value supplied. If the expected value is greater than 98% of a given range, the next higher range is automatically chosen. The table under the "Settings" heading gives the crossover points for range changes.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none
size	numeric	1 | 7 to 524288	none
expected value	numeric	-102.30 to 102.35 | DEFault | MINimum | MAXimum	volts
resolution	numeric	.00005 to .05 | DEFault | MINimum | MAXimum	volts
input port	numeric	1|3 with MEASure1 2|4 with MEASure2 1 and 2 are single ended 3 and 4 are differential	none
The maximum size parameter will be 524284 if the battery is enabled instead of 524288.			
For expected value, MINimum selects the 0.100 V range and MAXimum selects either the 1.0 volt range (single ended ports) or the 100 V range (differential ports). DEFault selects 1V.			
For resolution, MINimum, MAXimum, and DEFault select the same value.			

Settings

Maximum expected value settings per range are shown in the following table, along with the resolution associated with each range. The highlighted area shows the setting used when the expected value is not specified or DEFault is used.

MEASure[<*chan*>]:ARRay[:VOLTage][:DC]?

Maximum Expected Value (V)	Voltage Range (Volts)	Resolution (Volts)
± .1	0.10235	.00005
± .2	0.2047	.00010
± .5	0.51175	.00025
± 1	1.0235	.0005
± 2	2.047	.0010
± 5	5.1175	.0025
± 10	10.235	.005
± 20	20.47	.010
± 50	51.175	.025
±100	102.35	.05

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- If no input port is given, MEASure1 defaults to (@1) and MEASure2 defaults to (@2).
- Resolution varies by range, and is constant for each range. For any given range, specifying MINimum, MAXimum, or DEFault for resolution yields the same result.
- It is important to note that the expected value determines the resolution and not vice versa. The digitizer always uses 12-bit resolution and a coarser resolution value has no effect. If, for some reason a coarser resolution is desired, specify a larger expected value. See the previous table for expected values and resolutions.
- The MEASure command is equivalent to the command sequence: ABORt;CONFigure;INITiate:IMMediate;FETCh?. Because of the ABORt, VME (VXI data transfer) bus or Local bus transfers in progress will be aborted. This includes the pipelining of data.

- The MEASure command configures the HP E1429 to do the measurement specified by the parameters given with MEASure. All instrument settings will be forced to their *RST values, with the following exceptions. In particular, note that all OUTPut signals are turned off and their FEED's are changed to reset values, and that both the local bus (VINStrument:LBUS) and VME bus (VINS:VME) are turned off. Be aware that these new states will still be in effect after the MEASure command is complete. See the -- mumble which section -- for a complete list of the reset settings for the HP E1429. The following states are set up after the reset, and thus in most cases, will not be the *RST value for that command:

  a. SENSe<n>:FUNCtion is set to "VOLTage[:DC] <port>"

  b. SENSe<n>:VOLTage:DC:RANGe is set to the value implied by the expected value given.

  c. INPut[<*port*>]:FILTer is set to ON.

  d. SENSe:SWEep:POINts and TRIGger:STARt:COUNt are set to the <*size*> parameter.

- **Related commands:** SENSe:FUNCtion, SENSe:VOLTage:DC:RANGe, SENSe:SWEep:POINts, SENSe:ROSCillator:SOURce), TRIGger:STARt subsystem, ARM:STARt subsystem.

- ***RST Condition:** none

Example

Measuring 1.5 V on the differential input of channel 1

```
MEAS1:ARRay:VOLT? (20),1.5,(@3)
```

Set up, take, and bring back 20 readings of 1.5 Volts peak on port 3 of channel 1.

The MEMory subsystem controls whether memory will be non-volatile and determines the battery charge.

Subsystem Syntax

```
MEMory
    :BATTery
        [:STATe] <state>
        CHARge?                    [query only]
```

:BATTery[:STATe]

MEMory:BATTery[:STATe] <*state*> enables or disables non-volatile memory. If *state* ON (1) is set, then measurement data will be preserved in the HP E1429A's internal memory when power fails or the instrument is turned off.

Note For data to be saved, MEMory:BATTery:STATe ON must be set **before** the readings are taken (i.e. before the INITiate:IMMediate command is sent).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
state	discrete	OFF|ON|0|1	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. The state is coupled to the maximum trigger count.
- ***RST Condition:** not affected
- **Power-On condition:** The factory setting for MEMory:BATTery:STATe is OFF. The battery state does not change at power-down or power-on.
- Enabling the battery abbreviates the power-on selt-test. A complete self-test is executed with *TST?, which erases memory.

Example **Enable memory to be non-volatile**

```
MEM:BATT ON                    Turn the battery on
```

:BATTery:CHARge?

MEMory:BATTery:CHARge? returns 1 if the battery has sufficient charge to maintain memory, and returns 0 if the battery has insufficient charge to maintain memory.

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- **Related commands:** MEMory:BATTery:STATe
- ***RST Condition:** none

Example

Check the battery charge

```
MEM:BATT:CHAR?
```

A 1 is returned if the battery has enough charge

The OUTPut subsystem controls which output path (if any) will receive the synchronization pulses generated by the HP E1429. Sync pulses can be sent to the "Ext 1" BNC, the VXI TTL trigger lines, the VXI ECL trigger lines, or any combination of these three.

Subsystem Syntax

```
OUTPut
    :ECLTrg<n>
        :FEED<source>
        [:STATe] <mode>
    :EXTernal[1]
        :FEED<source>
        [:STATe] <mode>
    :TTLTrg<n>
        :FEED<source>
        [:STATe] <mode>
```

:ECLTrg<*n*>:FEED

OUTPut:ECLTrg<*n*>:FEED <*source*> specifies the source for the synchronization pulse which will be routed to ECLTrg0 and/or ECLTrg1. The available sources are:

"**EXTernal[1]"**: Outputs the signal currently specified by the OUTPut:EXTernal[1]:FEED command with the polarity inverted. OUTPut:EXTernal1:STATe ON must be set for output to occur. See the OUTPut:EXTernal[1]:FEED command for descriptions of the possible sources.

"**[SENSe[1|2]]:ROSCillator"**: The signal level will go high with the falling edge of an external reference oscillator (SENSe:ROSCillator:SOURce EXTernal), and will go high with the rising edge of all other reference oscillator sources: SENSe:ROSCillator:SOURce (INT | ECLT<n> | CLK10). When the output state is enabled, the signal is output as soon as this feed is selected, and will be output continuously until the feed source is changed to some other selection.

"**TRIGger[:STARt|:SEQuence[1]]"**: Outputs an approximately 25 nanosecond wide positive going pulse each time a convert pulse (TRIGger) is sent to the A to D converter.

Note These pulses will also be generated when data is read out of memory from the VINStrument:LBUS or VINStrument:VME subsystems.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n	numeric	0|1	none
source	string	"EXTernal[1]"| "[SENSe[1|2]]:ROSCillator"| "TRIGger[:STARt|:SEQuence[1]]"	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- Note that if the feed is EXTernal1, then OUTPut:EXTernal1:STATe must be ON in addition to OUTPut:ECLTrg<n>:STATe ON for an output to occur on the chosen trigger line.
- The source of the synchronization pulse is independent for each ECLTRG trigger line.
- **Related Commands:** OUTPut:ECLTrg<*n*>:STATe, OUTPut:EXTernal1:FEED, OUTPut:EXTernal1:STATe
- ***RST Condition:**
  OUTPut:ECLTrg0:FEED "TRIGger:STARt|SEQuence1",
  OUTPut:ECLTrg1:FEED "EXTernal1"

Example

Setting the ECLTrg0 sync pulse source

```
OUTP:ECLT0:FEED "TRIG"          Output a pulse whenever a reading is taken
```

:ECLTrg<*n*>[:STATe]

OUTPut:ECLTrg<*n*>[:STATe] <*mode*> enables or disables the routing of the selected synchronization pulse to the specified VXIbus ECLTRG trigger line (ECLTrg0 or ECLTrg1).

OUTPut:EXTernal[1]:FEED

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n	numeric	0|1	none
mode	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes, this command will cause a settings conflict error if the same ECLTRG trigger line is used in any of the following: SENSe:ROSCillator:SOURce, ARM:STARt|SEQuence1:SOURce1, ARM:STARt|SEQuence1:SOURce2 or TRIGger:STARt|SEQuence1:SOURce.
- Routing synchronization pulses to the ECLTRG trigger lines is independently enabled/disabled for each line.
- **Related Commands:** OUTPut:ECLTrg<*n*>:FEED
- ***RST Condition:** OUTPut:ECLTrg<n >:STATe OFF

Example

Enabling sync pulse output to ECLTRG0 and ECLTRG1

```
OUTP:ECLT0 ON          Enable ECLTRG0
OUTP:ECLT1 ON          Enable ECLTRG1
```

:EXTernal[1]:FEED

OUTPut:EXTernal[1]:FEED <*source*> specifies the source for the synchronization pulse which will be output on the "Ext 1" BNC. The available sources are:

"**ARM[:START|:SEQuence[1]]"**: Changes the normally high output level to low as soon as an ARM event (ARM:SOURce) is processed, and before any programmed ARM:DELay occurs. The level remains low until the ARM cycle is completed by TRIGger:STARt:COUNt readings being taken. This signal begins at the detection of the arm event, and does not include any programmed delay (ARM:STARt:DELay). The expected use of this signal is to allow a master module to detect an arm event and then arm other modules by using this signal.

"**RFTRigger"**: When **R**eady **F**or **TR**igger is selected, the normally high output level goes low after an ARM event occurs and the ARM:DELay specified has been met. At this point, the HP E1429 is ready to accept sample triggers. The level stays low until all sample triggers (TRIGger:STARt:COUNt) associated with the current arm cycle have completed.

"**[SENSe[1|2]]:SWEep:OFFSet:POINts"**: Changes the normally high output level to low after the pre-arm count (SENSe:SWEep:OFFSet:POINts) has been met. This would be used to determine when an arming event could occur without getting an "Arm ignored" error due to the pre-arm count not being satisfied. The level would not return to high again until either the next arm cycle (if ARM:STARt:COUNt > 1) or the next INITiate command if this is the last or only arm cycle. This source is only useful when pre-arm and post-arm readings are being taken; it is allowed without error in post-arm measurements, but is not of any real use.

"**[SENSe[1|2]]:ROSCillator"**: The reference oscillator clock is output. For all reference oscillator sources except EXTernal, the resulting output signal is the inverse of the actual reference (i.e. the output pulse goes low on the rising edge of the reference oscillator). For SENSe:ROSCillator:SOURce EXTernal, the output level goes low on the falling edge. Note that this signal begins being output as soon as this feed is selected, and will be output continuously until the feed source is changed to some other selection.

"**TRIGger[:STARt|:SEQuence[1]]"**: Outputs an approximately 25 nanosecond wide negative going pulse each time a convert pulse (TRIGger) is sent to the A to D converter. Note that these pulses will also be generated when data is read out of memory from the VINStrument:LBUS or VINStrument:LBUS subsystems.

Note These pulses will also be generated when data is read out of memory from the VINStrument:LBUS or VINStrument:VME subsystems.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
source	string	"ARM[:STARt|:SEQuence[1]]"| "RFTRigger"| "[SENSe[1|2]]:ROSCillator"| "[SENSe]:SWEep:OFFSet:POINts"| "TRIGger[:STARt|:SEQuence[1]]"	none

OUTPut:EXTernal[1][:STATe]

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- **Related Commands:** OUTPut:EXTernal[1]:[:STATe]
- ***RST Condition:** OUTPut:EXTernal:FEED "TRIGger:STARt"

Example **Setting the sync pulse source**

```
OUTP:EXT:FEED "ROSC"          Output the reference oscillator pulses
```

:EXTernal[1][:STATe]

OUTPut:EXTernal[1][:STATe] <***mode*** > enables or disables output of the synchronization pulse on the front panel "Ext 1" BNC.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes, this command will cause a settings conflict error if the EXTernal1 BNC connector is already in use. The following commands can use the EXTernal1 connector: ARM:STARt:SOURce1 or ARM:STARt:SOURce2
- ***RST Condition:** OUTPut:EXTernal[1][:STATe] OFF

Example **Enabling sync pulse output to the front panel "Ext 1" BNC**

```
OUTP:EXT ON                   Enable sync pulse output on "Ext 1" BNC
```

:TTLTrg<*n*>:FEED

OUTPut:TTLTrg<*n*>:FEED<*source* > selects which event will cause a pulse or level change on the TTL trigger line(s). Note that unlike the ECL trigger lines, there is only a single FEED that goes to all TTL trigger lines. Therefore, it does not matter what value of <n> is specified with this command, because all TTL trigger lines share the same feed. The available feed sources are:

"**ARM[:START|:SEQuence[1]]"**: Changes the normally high output level to low as soon as an ARM event (ARM:STARt:SOURce) is processed. The level remains low until the ARM cycle is completed by TRIGger:STARt:COUNt readings being taken. This signal begins at the detection of the arm event, and does not include any programmed delay (ARM:STARt:DELay). The expected use of this signal is to allow a master module to detect an arm event and then arm other modules by using this signal.

"**READy"**: Changes the normally high output level to low whenever the trigger system is initiated. If multiple groups (ARM:STARt:COUNt > 1) of pre-arm and post-arm measurements are being taken, the level changes back to high while the microprocessor changes which memory segment will receive the next burst of data, and then the level goes low again when the HP E1429 is again initiated. If only one burst of pre-arm and post-arm readings are taken, or if all data is post-arm, then the level changes back to high when the measurement is complete.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n	numeric	0 through 7	none
source	string	"ARM[:STARt|:SEQuence[1]]"| "READy"	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** No
- **Related Commands:** OUTPut:TTLTrg<*n*>[:STATe]
- ***RST Condition:** OUTPut:TTLTrg<*n*>:FEED "ARM:STARt"

Example

Enabling sync pulse for READy on TTLTrg5*

```
OUTP:TTLT5:FEED "READ"      Change output level when initiated
OUTP:TTLT5 ON               Enable TTLTRG5* line to transmit the signal
```

OUTPut:TTLTrg<*n*>[:STATe]

:TTLTrg<*n*>[:STATe]

OUTPut:TTLTrg<*n*>[:STATe] <*mode*> enables or disables routing of the synchronization pulse to the specified VXIbus TTL trigger lines (TTLTRG0* through TTLTRG7*).

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
n	numeric	0 through 7	none
mode	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. This command will cause a settings conflict error if the same TTL trigger line is used in any of the following: ARM:STARt:SOURce1, ARM:STARt:SOURce2 or TRIG:STARt:SOURce.
- Unlike the ECLTRG trigger lines, the TTLRG trigger lines are not independent with regard to FEED. All TTLTRG triggers enabled will be outputting the same synchronization signal specified by the OUTPut:TTLTrg<n>:FEED command.
- Routing synchronization pulses to the TTLTRG trigger lines is independently enabled/disabled for each line.
- ***RST Condition:** OUTPut:TTLTrg<*n*>:STATe OFF

Example

Enabling sync pulse output to TTLTRG0* and TTLTRG5*

```
OUTP:TTLT0 ON          Enable TTLTRG0*
OUTP:TTLT5 ON          Enable TTLTRG5*
```

The READ? command is used to cause a measurement and to retrieve the readings from that measurement. It is equivalent to executing the commands: ABORt, INITiate:IMMediate, FETCh<chan >?.

Subsystem Syntax READ[<*chan* >]? [query only]

READ?

READ[<*chan* >]*?* returns readings from the specified channel in the format specified by the FORMat:DATA command.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1 | 2	none

Comments

- **READ?** or **READ1?** will return data from channel 1 only. **READ2?** will return data from channel 2 only.
- A READ1? followed by a READ2? will take two sets of measurements since an INITiate is done with each read. If it is desired to look at each channel separately for readings taken during the same time period, use the INITiate and FETCh? sequence of commands instead of READ?.
- There are a few settings which, if allowed, would cause READ? to deadlock and never complete. TRIGger:COUNt INFinite and ARM:COUNt INFinite are not allowed because the measurement is continuous and therefore would never finish. Also, ARM:SOURce BUS or HOLD and TRIG:SOURce BUS or HOLD are not allowed because the parser would be unable to accept any commands to arm or trigger the instrument during measurement because the READ? command would not have completed. Any of the above settings active during a READ? command will cause an error to occur and no measurements to be taken.
- The number of readings that READ? is going to return for each channel can be calculated as

  (number of start triggers) * (pre-arm readings + post-arm readings)

  The above equation may be built from the following queries:

  (ARM:COUNt?) * (TRIG:COUNt?)

The number of pre-arm readings may be determined by taking the absolute value of the query SENSe:SWEep:OFFSet:POINts? which returns either 0 (no pre-arm readings) or a negative number which is the pre-arm count. If the measurements were ABORted, use FETCh:COUNt? to determine the number of readings.

- Using the above method, it is also possible to calculate the end of pre-arm data and the beginning of post-arm data when both are present in a measurement. The data will always be returned with the specified number of pre-arm readings followed by post-arm readings.

- The READ command is equivalent to the command sequence: ABORt;INITiate:IMMediate;FETCh?. Because of the ABORt, VME (VXI data transfer) bus or Local bus transfers in progress will be aborted. This includes the pipelining of data.

- **Related Commands:** ABORt, FETCh?, FORMat:DATA, INITiate:IMMediate, ARM:STARt:COUNt, TRIGger:STARt:COUNt, SENSe:SWEep:OFFSet:POINts

- ***RST Condition:** none

Example 1 Obtaining readings from the HP E1429

```
CONF1:ARR:VOLT (30),.6,DEF,(@1)      Configure for 30 readings on channel 1, port 1
READ1?                               Start measurement and get readings from channel 1.
```

Example 2 Reading back PACKed data (HP BASIC program)

```
DIM Ndig$[1],Count$[9]                          Dimension parameters for header
ASSIGN @X TO 70905;FORMAT OFF
                                                Turn format off for array data
OUTPUT 70905;"READ1?"                           Query for channel 1 measurement data
ENTER @X USING "#,X,K,K";Ndig$;Count$[1;VAL(Ndig$)]
                                                Strip the header preceeding the data
ALLOCATE INTEGER Meas_data(1:VAL(Count$)/2)
                                                Allocate an array to hold the data
ENTER @X;Meas_data(*)                           Read in the measurement data
ENTER 70905 USING "B";Junk                      Need to strip off left over line feed
```

The SENSe subsystem is used to specify the reference oscillator frequency (if external) and source, the input port used on a particular channel, and the pre-arm and post-arm reading counts. Each section of the subsystem is separately documented in the following sections of the Command Reference.

Subsystem Syntax

The first level SENSe syntax tree is:

```
[SENSe[<chan>]]
    :FUNCtion <function >
    :ROSCillator
        :EXTernal
            :FREQuency <frequency >
        :SOURce <source >
    :SWEep
        :OFFSet
            :POINts <count >
        :POINts <count >
    :VOLTage
        [:DC]
            :RANGe <range >
            :RESolution?            [query only]
```

The SENSe:FUNCtion commands select which of the input ports gets connected to a particular channel. SENSe1 selects the input for the sensor on channel 1 (ports 1 or 3), and SENSe2 selects the input for the sensor on channel 2 (ports 2 or 4).

Subsystem Syntax

```
[SENSe[<chan >]]
    :FUNCtion "<function >"
```

[SENSe[<*chan*>]]:FUNCtion

[SENSe[<*chan*>]]:FUNCtion "<*function*>" determines which input port will be connected to the specified sensing channel. Each sense channel has two input ports; one port is single ended and the other is differential. Odd numbered ports (1 and 3) connect to channel 1, and the even numbered ports (2 and 4) connect to channel 2.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
function	quoted string	"VOLTage[:DC]1" | "VOLTage[:DC]2" | "VOLTage[:DC]3" | "VOLTage[:DC]4"	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. This command is coupled to SENSe:VOLTage:DC:RANGe, ARM:STARt:LEVel:NEGative, and ARM:STARt:LEVel:POSitive.
- Function choices ending in 1 and 3 ("VOLTage:DC1" or "VOLTage:DC3") are the only choices for channel 1 (SENSe1:FUNCtion). Function choices ending in 2 and 4 ("VOLTage:DC2" or "VOLTage:DC4") are the only legal choices for channel 2 (SENSe2:FUNCtion). Specifying an illegal function choice on a sense channel will result in error -151, "Invalid string data".
- **Related Commands:** SENSe:VOLTage:DC:RANGe, ARM:STARt:LEVel:NEGative, ARM:STARt:LEVel:POSitive

- ***RST Condition:** SENSe1:FUNCtion "VOLT1", SENSe2:FUNCtion "VOLT2"

Example **Selecting the single ended input on channel 2**

SENS2:FUNC “VOLT4” *Connect port 4 to channel 2*

[SENSe[<*chan*>]]:ROSCillator

The SENSe:ROSCillator subsystem controls the reference oscillator source and frequency used to generate sample rates for taking measurements.

Subsystem Syntax

```
[SENSe[<chan>]]
    :ROSCillator
        :EXTernal
            :FREQuency <frequency>
        :SOURce <source>
```

Since the triggering and timebase circuits of the HP E1429 are shared between both sensing channels, the settings for [SENSe[<chan>]]:ROSCillator are the same for both values of chan (1 and 2). Setting SENSe1:ROSCillator values will also set SENSe2:ROSCillator values and vice versa.

:EXTernal:FREQuency

[SENSe[<*chan*>]]:ROSCillator:EXTernal:FREQuency <*frequency*> indicates to the HP E1429 the frequency of an external reference oscillator signal. The SENSe:FREQuency subsystem uses this value to generate the sample rate when SENSe:ROSCillator:SOURce is set to EXTernal, ECLTrg0, or ECLTrg1.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
frequency	numeric	10 KHz through 20 MHz MINimum|MAXimum	Hz
MINimum selects 10 KHz; MAXimum selects 20 MHz.			

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes. This command is coupled to the commands in the TRIGger subsystem.
- Indicating an incorrect frequency for an external reference oscillator will cause the sample rate and trigger delay to be incorrect.
- The amplitude of the external reference signal should be a TTL level: low = 0.0V to 0.8V, high = 2.5V to 5.0V.

- Using MINimum or MAXimum to specify this frequency is not recommended unless the external reference frequency is 20 kHz (MINimum) or 20 MHz (MAXimum). In order for the digitizer processor to produce the intended sample rates, the exact frequency value must be specified.
- Related Commands: SENSe:ROSCillator:SOURce, TRIGger:STARt:TIMer, ARM:STARt:DELay
- ***RST Condition:** SENSe:ROSCillator:EXTernal:FREQuency 20 MHZ

Example **Specifying the external reference oscillator frequency**

```
SENS:ROSC:EXT:FREQ 5 MHZ        External oscillator is 5 MHz
```

:SOURce

[SENSe[<*chan*>]]:ROSCillator:SOURce <*source*> selects the reference oscillator source. The available sources are:

- **CLK10**: The VXIbus CLK10 (10 MHz) line.
- **ECLTrg0**: The VXIbus ECLTRG0 line.
- **ECLTrg1**: The VXIbus ECLTRG1 line.
- **EXTernal2**: The HP E1429's front panel "Ext 2" BNC.
- **INTernal**: The internal 20 MHz oscillator.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
source	discrete	CLK10|EXTernal2| ECLTrg0|ECLTrg1 INTernal	none

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes. This command is coupled to the TRIGger subsystem, the OUTPut subsystem, and ARM:STARt:DELay.
- The reference oscillator is used to generate the sample rate and trigger delay.

- Use SENSe:ROSCillator:EXTernal:FREQuency to indicate the frequency of an external reference oscillator.
- **Related Commands:** SENSe:ROSCillator:EXTernal:FREQuency, ARM:STARt:DELay, TRIGger:STARt:TIMer[1|2]
- ***RST Condition:** SENSe:ROSCillator:SOURce INTernal

Example **Setting the Reference Oscillator Source**

```
SENS:ROSC:SOUR CLK10          Select VXI CLK10 line as oscillator source
```

The SENSe:SWEep commands select how many readings will be taken during a measurement, and how many of the readings occur before (pre-arm) and after (post-arm) the arm event.

Since the triggering and timebase circuits of the HP E1429 are shared between both sensing channels, the settings for [SENSe[<chan>]]:SWEep are the same for both values of chan (1 and 2). Setting SENSe1:SWEep values will also set SENSe2:SWEep values and vice versa.

Memory Usage Measurements which specify multiple bursts (ARM:STARt:COUNt > 1) of both pre-arm and post-arm readings (SENSe:SWEep:OFFSet:POINts ≤ -3 with SENSe:SWEep:POINts > 9) cause memory to be partitioned into segments to hold each burst of readings. The HP E1429 will automatically allocate ARM:STARt:COUNt memory partitions large enough to hold the specified number of pre-arm and post-arm readings. Since a large number of pre-arm readings may occur before the arm event causes post-arm readings to be taken, each memory partition is treated like a circular buffer where pre-arm readings may "wrap" or overwrite each other multiple times before the arm event occurs and the burst of readings completes with the post-arm readings being taken. After all post-arm readings have been taken in a partition, if ARM:STARt:COUNt is not yet satisfied, the instrument directs the next burst of readings into the next memory partition.

Note There is a time window of typically 630 µs where pre-arm readings will be lost while the HP E1429 arranges for readings to be directed into the next memory partition. If an ARM event (ARM:STARt:SOURce) or TRIGger event (sample) occurs during this time window, it will be ignored with no error reported.

[SENSe[<*chan* >]]:SWEep:OFFSet:POINts <*count*>

Number of Memory Segments	Maximum Number of Readings
1	524,288
2	262,144
4	131,072
8	65,536
16	32,768
32	16,384
64	8,192
128	4,096
NOTE: If the non-volatile mode of memory is enabled then the maximum number of readings for each memory partition decreases by four. These four memory locations in each segment hold the data necessary to recover all readings after a power failure.	

Subsystem Syntax

```
[SENSe[<chan>]]
  :SWEep
    :OFFSet
      :POINts <count >
    :POINts <count >
```

:OFFSet:POINts <*count*>

[SENSe[<*chan*>]]:SWEep:OFFSet:POINts <*count*> specifies how many pre-arm readings will be taken. When pre-arm readings are not 0, then there must always be at least 7 post-arm readings taken. Note that *count* is specified as a negative number.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
count	numeric	-65535 to -3 | 0 | MINimum | MAXimum	none
MINimum and MAXimum vary depending on ARM:STARt:COUNt and SENSe:SWEep:POINts.			

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. This command is coupled to TRIGger:STARt:COUNt, SENSe:SWEep:POINts, ARM:STARt:COUNt, ARM:STARt:DELay, and MEMory:BATTery:STATe.
- Readings which are taken before the arm event occurs are called "pre-arm" readings. Readings taken after the arm event are called "post-arm" readings.
- The INITiate command "triggers" pre-arm readings. If any arming events occur before the pre-arm count is satisfied, the arms are ignored and an error occurs. When the pre-arm count is satisfied and a legal arm event occurs, post-arm sampling begins. The remaining number of TRIGger:STARt:COUNt (SENSe:SWEep:POINts) readings are then taken.
- If pre-arm readings are not 0, then there must be at least 7 post-arm readings. Therefore, (SENSe:SWEep:POINts + SENSe:SWEep:OFFSet:POINts) must be ≥ 7. (Note that in the previous equation, :OFFSet:POINts will be a negative number).
- Pre-arm reading count values between -3 and 0 will be rounded to -3 or 0, whichever is closer to the specified count.
- If an ABORt or power failure occurs during a sequence of measurements, the digitizer will return (FETCh?) or recover (FETCh:RECover?) between one and TRIGger:STARt:COUNt number of readings. Because the digitizer processor does not know when the arm occurs, the readings returned may be pre-arm only, post-arm only, or a combination of both. If less than TRIGger:STARt:COUNt readings have been taken, then that number of readings are returned. If greater than TRIGger:STARt:COUNt readings have been taken, then TRIGger:STARt:COUNt readings are returned.
- **Related Commands:** SENSe:SWEep:POINts, TRIGger:STARt:COUNt, ARM:STARt:COUNt
- ***RST Condition:** SENSe:SWEep:OFFSet:POINts 0

Example

Setting 50 pre-arm readings on channel 1, input port 3

CONF1:ARR:VOLT (100),5,(@3)	*Configure channel 1 for 100 readings, 5V range*
SENS1:SWE:OFFS:POIN -50	*Of the 100 total readings, set 50 to be pre-arm*

[SENSe[<*chan* >]]:SWEep:POINts <*count*>

:POINts <*count*>

[SENSe[<***chan***>**]]:SWEep:POINts** <***count***> specifies how many readings will be taken during each ARM:STARt:COUNt cycle of the trigger system. This command is the same as (and is coupled to) TRIGger:STARt:COUNt. Changing either changes the value of the other as well.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
count	numeric	1 | 7 through 16,777,215 | MINimum|MAXimum| 9.9E+37|INFinity	none

MINimum selects 1 reading.

Memory, post-arm readings only: MAXimum = 524,288 / ARM:STARt:COUNt
Memory battery enabled: MAXimum = (524,288 / ARM:STARt:COUNt) - 4
A/D converter to VME (VXI data transfer) bus: MAXimum = 16,777,215
A/D converter to Local bus: MAXimum = 16,777,215
Digitizer memory, pre- and post-arm readings: see below

ARM:STARt:COUNt	Number of Memory Segments	Maximum Readings (TRIGger:STARt:COUNt)
1	1	524,288
2	2	262,144
3 - 4	4	131,072
5 - 8	8	65,536
9 - 16	16	32,768
17 - 32	32	16,384
33 - 64	64	8,192
65 - 128	128	4096

NOTE: If the non-volatile mode of memory is enabled (MEMory:BATTery:STATe ON), then all of the maximum reading counts shown above decrease by four. These four memory locations in each segment hold the data necessary to recover all readings after a power failure.

Comments

- **Executable when initiated:** No
- **Coupled Command:** Yes. This command is coupled to TRIGger:STARt:COUNt, SENSe:SWEep:OFFSet:POINts, ARM:STARt:COUNt, and MEMory:BATTery:STATe.
- SENSe:SWEep:POINts values between 1 and 7 will be rounded to 1 or 7, whichever is closer to the specified count.
- If the count is set to INFinity or 9.9E+37, the ABORt command must be used to return the trigger system to the idle state before any readings taken may be read from memory. Due to this, the READ? command can not be used when SENSe:SWEep:POINts is set to INFinity; attempts to do so will result in error -214, "Trigger deadlock".
- The count is the total of both pre-arm readings. The number of pre-arm readings is specified by the SENSe:SWEep:OFFSet:POINts command as a negative count. If SENSe:SWEep:OFFSet:POINts is 0, then all readings will be post-arm.
- **Related Commands:** ABORt, INITiate:IMMediate, ARM subsystem, SENSe:SWEep:OFFSet:POINts, TRIGger:STARt:COUNt.
- ***RST Condition:** SENSe1:SWEep:POINts 1

Example

Setting 500 readings.

```
SENS1:SWE:POIN 500                    set reading count
```

[SENSe[<*chan*>]]:VOLTage[:DC]

The SENSe:VOLTage[:DC] commands select the voltage range, and report back the resolution associated with the selected range.

Subsystem Syntax

```
[SENSe[<chan >]]
    :VOLTage
        [:DC]
            :RANGe <range >
            :RESolution?            [query only]
```

:RANGe

[SENSe[<*chan* >]]:VOLTage[:DC]:RANGe <*range* > selects the range for voltage measurement on the specified channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
range	numeric	-102.30 to 102.35 | MINimum | MAXimum	volts

The various range settings are given in the following table, along with the input ports supported on each range, and the measurement range spanned by the given range.

Range Setting (V)	Measurement Range (V)	Resolution (Volts)	Allowable Ports
0.10235	-.10230 to .10235	.00005	1,2,3,4
0.2047	-.2046 to .2047	.00010	1,2,3,4
0.51175	-.5115 to .51175	.00025	1,2,3,4
1.0235	-1.0230 to 1.0235	.0005	1,2,3,4
2.047	-2.046 to 2.047	.0010	3,4
5.1175	-5.115 to 5.1175	.0025	3,4
10.235	-10.230 to 10.235	.005	3,4
20.470	-20.460 to 20.470	.010	3,4
51.175	-51.150 to 51.175	.025	3,4
102.35	-102.30 to 102.35	.05	3,4

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** Yes. The command is coupled to ARM[:STARt]:LEVel[<chan >]:NEGative, ARM[:STARt]:LEVel[<chan >]:POSitive, and SENSe[<chan >]:FUNCtion.
- Though the range setting may be changed while the HP E1429 is taking readings (INITiated), it will take at least 3 ms for the relay to settle. Also, it is up to the user to determine where in the data archive the new range setting was switched in, as the HP E1429 does not stamp the data in any way. If the data format is not PACKed, then the HP E1429 will use the resolution associated with the final range setting to convert all readings into voltage values when FETCh? is executed.
- MAXimum values for the range setting will depend on the current SENSe:FUNCtion setting for that channel. For single-ended ports (1 and 2), MAXimum is 1.0235, and for differential ports (3 and 4) MAXimum is 102.35.
- ***RST Condition:** SENSe1:VOLTage:DC:RANGe 1.0235, and SENSe2:VOLTage:DC:RANGe 1.0235

[SENSe[<*chan*>]]:VOLTage[:DC] :RESolution?

Example **Selecting the 102.35 Volt range on channel 2**

```
SENS2:FUNC “VOLT4”                  Connect differential port 4 to channel 2
SENS2:VOLT:RANG 75                  Select 102.35 volt range. Any number
                                    greater than 51.175 forces the next highest
                                    (102.35 V) range.
```

:RESolution?

[SENSe[<*chan* >]]:VOLTage[:DC]:RESolution? returns the resolution associated with the current SENS:VOLT:RANGe setting, on the specified channel.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- ***RST Condition:** None

Example **Querying the current resolution on channel 2**

```
SENS2:VOLT:RES?                     Query channel 2 for resolution setting
```

The STATus subsystem controls the SCPI-defined Operation and Questionable Signal status registers. Each is comprised of a condition register, an event register, an enable mask, and negative and positive transition filters.

Each status register works as follows: when a condition occurs, the appropriate bit in the condition register is set or cleared. If the the corresponding transition filter is enabled for that bit, the same bit is set in the associated event register. The contents of the event register and the enable mask are logically ANDed bit-for-bit; if any bit of the result is set, the summary bit for that register is set in the status byte. The status byte summary bit for the Operation status register is bit 7; for the Questionable Signal status register, bit 3.

Operation Status Register

Only bits 0 (CALibrating), 6 (Waiting for ARM), 8 (BUSY), and 9 (READy) are defined for the HP E1429. All other bits are always zero.

Bit 0 - CALibrating: Set (1) during the execution of the CALibration:ZERO, CALibration:GAIN, or CALibration:DELay command. Cleared (0) otherwise.

Bit 6 - Waiting for ARM: Set (1) when waiting for a start arm. Cleared (0) when a start arm is accepted or when measurement is aborted.

Bit 8 - BUSY: Set (1) by the INITiate:IMMediate, VINStrument:CONFigure:VME:MEMory:INITiate, or VINStrument:CONFigure:LBUS:MEMory:INITiate command. Cleared (0) when the measurement is complete or is aborted, returning the digitizer to the idle state.

Bit 9 - READy: Set(1) when the digitizer memory segment is ready for data storage. Cleared (0) while the digitizer is partitioning the next memory segment.

Questionable Signal Status Register

Only bits 0 (VOLTage), 2 (TIME), and 8 (CALibration) are defined. All other bits are always 0.

Bit 0 - VOLTage: Set (1) if an overload voltage is detected during a measurement. Cleared (0) otherwise.

Bit 2 -TIME: Set (1) if the divide-by-n of the reference oscillator source can not generate a sample rate that is within 1% of the rate specified by TRIG:TIMer1 or TRIG:TIMer2. Cleared (0) otherwise.

Bit 8 - CALibration: Set (1) if an error has been detected in the non-volatile calibration memory. Cleared (0) otherwise.

Subsystem Syntax

```
STATus
    :OPC
        :INITiate <state>
    :OPERation|:QUEStionable
        :CONDition?                    [query only]
        :ENABle <unmask>
        [:EVENt]?                      [query only]
        :NTRansition <unmask>
        :PTRansition <unmask>
    :PRESet                            [no query]
```

:OPC:INITiate

STATus:OPC:INITiate <***state*** > controls whether the *OPC, *OPC?, and *WAI commands will complete immediately or whether they will wait for all measurements to complete. With STATe OFF set, these commands will complete immediately, which indicates that the parser is idle and ready for the next command, even though the HP E1429 may still be in the INITiated state and taking a measurement. With STATe ON set, these commands will wait for the Pending Operation Flag set true by INITiate:IMMediate, VINStrument:CONFigure:VME:MEMory:INITiate, or VINStrument:CONFigure:LBUS:MEMory:INITiate to return false, indicating that the trigger system is in the idle state and that all measurements have completed or been aborted by the ABORt or *RST commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
state	boolean	OFF|0|ON|1	none

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *OPC, *OPC?, *RST, *WAI, ABORt, INITiate:IMMediate, VINStrument:CONFigure:VME:MEMory:INITiate, VINStrument:CONFigure:LBUS:MEMory:INITiate, STATus:PRESet
- ***RST Condition:** unaffected
- **Power-on Condition:** STATus:OPC:INITiate ON

Example **Setting immediate completion mode**

```
STAT:OPC:INIT OFF          Complete immediately for *OPC, etc.
```

:OPERation|:QUEStionable:CONDition?

STATus:OPERation|:QUEStionable:CONDition? returns the contents of the appropriate condition register. Reading the register does not affect its contents.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** all bits of both condition registers are cleared as a result of the state present after *RST, except for the CALibration bit in the Questionable Signal register, which will remain set if the error condition persists.

Example **Querying the Operation condition register**

```
STAT:OPER:COND?            Query Operation condition register
```

:OPERation|:QUEStionable:ENABle

STATus:OPERation|:QUEStionable:ENABle <*unmask*> specifies which bits of the associated event register are included in its summary bit. The summary bit is the bit-for-bit logical AND of the event register and the unmasked bit(s).

STATus:OPERation|:QUEStionable[:EVENt]?

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unmask	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUEStionable:ENABLE 0

Example

Setting the Operation register enable mask

```
STAT:OPER:ENAB #H0040          Enable summary on Waiting for ARM bit
```

:OPERation|:QUEStionable[:EVENt]?

STATus:OPERation|:QUEStionable[:EVENt]? returns the contents of the appropriate event register. Reading the register clears it to 0.

Comments

- Both event registers are also cleared to 0 by the *CLS common command.
- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** Both event registers are cleared to 0.

Example

Querying the Operation event register

```
STAT:OPER?                     Query Operation event register
```

:OPERation|:QUEStionable:NTRansition

STATus:OPERation|:QUEStionable:NTRansition <*unmask*> sets the negative transition mask. For each bit unmasked, a 1-to-0 transition of that bit in the associated condition register will set the same bit in the associated event register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unmask	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUEStionable:NTRansition 0

Example

Setting the Operation register negative transition mask

```
STAT:OPER:NTR 64          Set event bit when wait-for-arm state is entered
```

:OPERation|:QUEStionable:PTRansition

STATus:OPERation|:QUEStionable:PTRansition <*unmask*> sets the positive transition mask. For each bit unmasked, a 0-to-1 transition of that bit in the associated condition register will set the same bit in the associated event register.

STATus:PRESet

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
unmask	numeric or non-decimal numeric	0 through +32767	none

The non-decimal numeric forms are the #H, #Q, or #B formats specified by IEEE-488.2.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-on Condition:** STATUS:OPERation|QUEStionable:PTRansition 32767

Example

Setting the Operation register positive transition mask

```
STAT:OPER:PTR 64          Set event bit when wait-for-arm state is entered
```

:PRESet

STATus:PRESet initializes the enable registers and transition masks for the Operation and Questionable Signal status registers and sets STATus:OPC:INITiate ON. For both status registers, the enable registers are set to 0, the negative transition masks are set to 0, and the positive transition masks are set to 32767.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** STATus subsystem, *SRE, *STB?
- ***RST Condition:** none

Example

Presetting the STATus subsystem

```
STAT:PRES                 Preset STATus subsystem
```

The SYSTem command subsystem returns error messages and the SCPI version number to which the HP E1429A/B complies.

Subsystem Syntax

```
SYSTem
    :ERRor?          [query only]
    :VERSion?        [query only]
```

ERRor?

SYSTem:ERROR? returns the error messages in the error queue. See Appendix B for a listing of the digitizer error numbers and messages.

Comments

- As errors are detected, they are placed in the error queue. The queue is first-in, first out, meaning that if several error messages are waiting in the queue, SYSTem:ERRor? returns the oldest unread error message.
- The error queue can hold 30 error messages. If more than 30 messages are generated without being read, the last error message in the queue is replaced with error -350,"Too many errors". No additional messages are placed into the queue until SYSTem:ERRor? reads some messages or until the queue is cleared using the *CLS (clear status) command.
- When the error queue is empty, SYSTem:ERRor? returns +0,"No error".
- ***RST Condition:** unaffected
- **Power-On Condition:** no errors are in the error queue

Example **Reading the error queue**

```
SYST:ERR?                    Query the error queue
```

:VERSion?

SYSTem:VERSion? returns the SCPI version number to which the HP E1429A/B complies: "1992.0".

Example **Querying the SCPI revision**

```
SYST:VERS?                   Query SCPI revision
```

- ***RST Condition:** none

The TRIGger command subsystem controls the fourth state in a four state measurement process. The four states which occur during a successful reading are idle, initiated, wait-for-arm, and wait-for-trigger. The last two states have event detection associated with them which control when they exit the current state. These four states are more fully described as follows:

- Idle -- In this state, the instrument is not measuring. This is the state where setting changes are done via commands to the instrument. This state is exited when an INITiate command is received. This state is returned to after a reset, successful completion of measurement, or abort of measurement.

- Initiated -- Once the instrument is initiated with the INITiate command, it passes through this state, and continues down to the wait-for-arm state if ARM:STARt:COUNT is not yet satisfied.

- Wait-for-arm -- In this state, the instrument waits for the specified ARM event to occur before exiting to the wait-for-trigger state to make a measurment.

- Wait-for-trigger -- In this state, the instrument waits for the specified trigger event to occur, and when it occurs, a reading is taken. After a reading is taken, the cumulative number of readings taken thus far is compared to the count specified in TRIGger:STARt:COUNt or SENSe:SWEep:POINts. When the count is met, the state is exited, otherwise, the instrument waits for another trigger and takes another reading. Upon exit from this state, the instrument goes to the initiated state and checks to see whether or not ARM:STARt:COUNt is satisfied.

The following controls can be specified from the TRIGger:STARt subsystem:

- The number of triggers (readings) to occur before the digitizer returns to the initiated state (TRIGger:STARt:COUNt).

- The source of the trigger (TRIGger:STARt:SOURce).

- The sample time for each reading (TRIGger:STARt:TIMer1) and (TRIGger:STARt:TIMer2).

Memory Usage

Measurements which specify multiple bursts (ARM:STARt:COUNt > 1) of both pre-arm and post-arm readings (SENSe:SWEep:OFFSet:POINts $\leq$ -3 with SENSe:SWEep:POINts > 9) cause memory to be partitioned into segments to hold each burst of readings. The HP E1429 will automatically allocate ARM:STARt:COUNt memory partitions large enough to hold the specified number of pre-arm and post-arm readings. Since a large number of pre-arm readings may occur before the arm event causes post-arm readings to be taken, each memory partition is treated like a circular buffer where pre-arm readings may "wrap" or

overwrite each other multiple times times before the arm event occurs and the current cycle of readings completes with the post-arm readings being taken. After all post-arm readings have been taken in a partition, if ARM:STARt:COUNt is not yet satisfied, the instrument directs the next burst of readings into the next memory partition. There is a time window of typically 630 µs between bursts, where no readings will be taken, while the HP E1429 arranges for readings to be directed into the next memory partition. If an ARM event (ARM:STARt:SOURce) occurs during this time window, it will be ignored with no error reported.

The number of partitions allowed is a function of ARM:STARt:COUNt, and is shown in the table below along with the maximum number of readings (TRIGger:STARt:COUNt) allowed in each partition.

ARM:STARt:COUNt	Number of Memory Segments	**Maximum Readings (TRIGger:STARt:COUNt)**
1	1	524,288
2	2	262,144
3 - 4	4	131,072
5 - 8	8	65,536
9 - 16	16	32,768
17 - 32	32	16,384
33 - 64	64	8,192
65 - 128	128	4,096
NOTE: If the non-volatile mode of memory is enabled (MEMory:BATTery:STATe ON), then all of the maximum reading counts shown above decrease by four. These four memory locations in each segment hold the data necessary to recover all readings after a power failure.		

Subsystem Syntax

```
TRIGger
    [:STARt|:SEQuence[1]]
        :COUNt <number>
        [:IMMediate]                [no query]
        :SOURce <source>
        :TIMer1 <period>
        :TIMer2 <period>
```

[:STARt]:COUNt

TRIGger[:STARt]:COUNt <*number*> specifies the total number of readings which will be taken during each ARM:STARt:COUNt cycle of a measurement. This command is identical to (and coupled to) SENSe:SWEep:POINts; setting either will set both to the same count.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
chan	numeric	1|2	none
number	numeric	1 | 7 through 16,777,215 | MINimum|MAXimum| 9.9E+37|INFinity	none

MINimum selects 1 reading.

Memory, post-arm readings only: MAXimum = 524,288 / ARM:STARt:COUNt
Memory battery enabled: MAXimum = (524,288 / ARM:STARt:COUNt) - 4
A/D converter to VME (VXI data transfer) bus: MAXimum = 16,777,215
A/D converter to Local bus: MAXimum = 16,777,215
Digitizer memory, pre- and post-arm readings: see below

ARM:STARt:COUNt	Number of Memory Segments	Maximum Readings (TRIGger:STARt:COUNt)
1	1	524,288
2	2	262,144
3 - 4	4	131,072
5 - 8	8	65,536
9 - 16	16	32,768
17 - 32	32	16,384
33 - 64	64	8,192
65 - 128	128	4096

NOTE: If the non-volatile mode of memory is enabled (MEMory:BATTery:STATe ON), then all of the maximum reading counts shown above decrease by four. These four memory locations in each segment hold the data necessary to recover all readings after a power failure.

Comments

- **Executable when initiated:** No
- **Coupled Command:** This command is coupled to SENSe:SWEep:POINts, SENSe:SWEep:OFFSet:POINts, ARM:STARt:COUNt, and MEMory:BATTery:STATe.
- TRIGger:STARt:COUNt values between 1 and 7 will be rounded to 1 or 7, whichever is closer to the specified count.
- If the count is set to INFinity or 9.9E+37, the ABORt command must be used to return the trigger system to the idle state before any readings taken may be read from memory. Due to this, the READ? command can not be used when TRIGger:STARt:COUNt is set to INFinity, attempts to do so will result in error -214, "Trigger deadlock".
- Multiple bursts of a measurement process involving both pre-arm and post-arm readings (ARM:STARt:COUNt > 1 and SENSe:SWEep:OFFSet:POINts < 0) is a special case which causes memory partitioning to occur. This partitioning of memory is handled automatically by the instrument, and is a function of ARM:STARt:COUNt and TRIGger:STARt:COUNt (SENSe:SWEep:POINts). TRIGger:STARt:COUNt is satisfied first, and then ARM:STARt:COUNt is attempted.
- **Related Commands:** ABORt, INITiate:IMMediate, ARM subsystem, SENSe:SWEep:POINts, SENSe:SWEep:OFFSet:POINts
- ***RST Condition:** TRIGger:STARt:COUNt 1

Example

Taking 20 readings (8 pre-arm and 12 post-arm)

```
ARM:COUNt 1                 ARM the reading trigger once
TRIG:COUN 20                Take 20 readings each arm cycle
SENS:SWE:OFFS:POIN -8       Set 8 readings as pre-arm
```

[:STARt][:IMMediate]

TRIGger[:STARt][:IMMediate] will cause a reading to be taken immediately when the digitizer is in the wait-for-trigger state (ARM event has occurred), regardless of the selected trigger source. The number of triggers (set by TRIGger:STARt:COUNt) will be decremented by one. The selected trigger source remains unchanged.

There is no query form of this command.

Comments

- **Executable when initiated:** Yes
- **Coupled Command:** No
- If the trigger system is in the idle or wait-for-arm state, TRIGger:STARt:IMMediate will cause error -211,"Trigger ignored" to be generated, and no action will be taken.
- **Related Commands:** INITiate:IMMediate
- ***RST Condition:** none

Example

Forcing a measurement to occur

ARM:SOUR IMM	*Arm trigger immediately after INIT command received*
TRIG:SOUR HOLD	*Set trigger source to hold*
INIT	*Initiate trigger system, trigger will go to hold*
TRIG	*Override the hold and take a reading*

[:STARt]:SOURce

TRIGger[:STARt]:SOURce <*source*> configures the trigger system to respond to the specified source for taking readings.

The available sources are:

- **BUS**: The Group Execute Trigger (GET) HP-IB command or the IEEE-488.2 *TRG common command.
- **ECLTrg0** and **ECLTrg1**: The VXIbus ECL trigger lines.
- **DECLtrg**: Samples are taken at a dual rate, using the VXIbus ECLTRG0 trigger line to pace pre-arm readings, and the ECLTRG1 trigger line to pace the post-arm readings.

- **DEXTernal**: Samples are taken at a dual rate, using the signal on the Ext 1 input to pace pre-arm readings, and the signal on the Ext 2 input to pace the post-arm readings.
- **DTIMer**: Samples are taken at a dual rate, using the TRIGger:STARt:TIMer1 rate to pace pre-arm readings, and the TRIGger:STARt:TIMer2 rate to pace the post-arm readings.
- **TTLTrg0** through **TTLTrg7**: The VXIbus TTL trigger lines.
- **EXTernal1**: The HP E1429's front panel "Ext 1" BNC connector.
- **EXTernal2**: The HP E1429's front panel "Ext 2" BNC connector.
- **HOLD**: Suspend triggering. The TRIGger:STARt:IMMediate command must be used to trigger a reading.
- **TIMer**: Use the period specified by TRIGger:STARt:TIMer1 as the sample rate.
- **VME**: This source is used to either trigger readings from VME A24 register accesses, or to read measurement data out of the HP E1429 internal memory by reading a VME A24 address. This allows for faster data transfer rates over the VME bus than would be possible over HPIB. For more information on how to transfer data out over the VME bus using this command, see the VINStrument:VME commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
source	discrete	BUS|HOLD| DECLtrg|DEXTernal |DTIMer| ECLTrg0|ECLTrg1| IMMediate| EXTernal1|EXTernal2| TIMer| TTLTrg0 to TTLTrg7| VME	none

Comments

- **Executable when initiated:** No
- **Coupled Command:** This command is coupled to the TRIGger, OUTPut, and SENSe subsystems.

- The active edges for the various sources are as follows:

  ECLtrg0, ECLtrg1 and DECLtrg active edge is the rising edge.

  TTLTrg<n> and DTTLtrg active edge is the falling edge.

  EXTernal1, EXTernal2, and DEXTernal active edge is the falling edge.

- When using DECLtrg, DEXTernal, or DTIMer, at least one pre-arm pulse must occur **after** the arming signal has been received. This means that after the pre-arm count is reached and the arm is accepted, there must be another pre-arm pulse which arrives ([reference period * 3] + 60 ns) after the arming signal. This would typically be 210 ns if SENSe:ROSCillator:SOURce is INTernal. The digitizer does not sample on the additional pulse.

- When TIMer is the source, the desired period must be specified by the TRIGger:STARt:TIMer1 and/or the TRIGger:STARt:TIMer2 commands. See these commands for a table of allowable values.

- **Related Commands:** SENSe:ROSCillator:SOURce, ARM:STARt:SOURce, OUTPut subsystem, TRIGger:STARt:TIMer1, TRIGger:STARt:TIMer2, SENSe:SWEep:POINts

- ***RST Condition:** TRIGger:STARt:SOURce TIMer

Example

Setting the start trigger source

TRIG:SOUR ECLT0	*A reading will be taken with each ECLT0 pulse*

[:STARt]:TIMer[1]

TRIGger[:STARt]:TIMer[1] <*period*> specifies the time period between each sampling event. The time period must be a multiple of the reference oscillator period, with allowable multiples being 1,2,4,10,20,40,.....1E8, 2E8, 4E8. See the table below for the exact periods available with the internal 20 MHz reference.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
period	numeric	reference period to reference period * 4E8 | MINimum | MAXimum	seconds

Comments

- **Executable when initiated:** No
- **Coupled Command:** This command is coupled to the TRIGger:STARt:TIMer2 command and TRIGger:STARt:SOURce command as noted below:

  Unless TRIGger:STARt:SOURce is DTIMer, the settings of TRIGger:STARt:TIMer1 and TRIGger:STARt:TIMer2 are not coupled, and changing one will not affect the setting of the other.

  If TRIGger:STARt:SOURce is DTIM, then both TRIGger:STARt:TIMer1 and TRIGger:STARt:TIMer2 are used, and there is a coupling between the two settings. The relationship between the two settings is such that one of these two values must be exactly one reference oscillator period and the other must be a multiple (greater than 1.0) of the reference oscillator period. A record is kept of which setting was changed the most recently, and that setting (TRIGger:STARt:TIMer1 or TRIGger:STARt:TIMer2) is assumed to be the desired setting. For example, consider a reference oscillator period of 1µs; if TRIGger:STARt:TIMer2 was last changed to a value of 1µs while TRIGger:STARt:SOUR was set to DTIM, and the TRIGger:STARt:TIMer1 setting was also 1µs, TRIGger:STARt:TIMer1 would be changed to 2µs (the multiple 2.0 was chosen arbitrarily) so that both settings are not 1.0 times the reference period. Similarly, if TRIGger:STARt:TIMer2 were set to 4µs, and TRIGger:STARt:TIMer1 was some value greater than 1µs (like 2µs), TRIGger:STARt:TIMer1 would be automatically changed to1µs so that one of the two values is 1.0 times the reference period.

- If dual rate sampling is enabled (TRIGger:STARt:SOUR is DTIM), then the sample period specified by TIMer1 will be the sample rate for the pre-arm readings of the dual rate measurement, and TRIGger:STARt:TIMer2 will be the post-arm sample rate.
- If TRIGger:STARt:SOUR is TIMer, then only the sample rate specified by TIMer1 is used.
- Note that it is only necessary to set the longest sample rate if TRIGger:STARt:SOURce is DTIM. The other setting will be automatically forced to be one reference oscillator period due to the coupling between the two rates when TRIGger:STARt:SOURce is DTIM.
- If TRIGger:STARt:SOUR is neither TIMer or DTIM, then the sample rates specified by TRIGger:STARt:TIMer1 and TRIGger:STARt:TIMer2 are retained, but are not used nor coupled to each other.
- If the HP E1429 can not sample within 1 percent of the period specified by TRIGger:STARt:TIMer1, then the TIME bit (bit 2) in the QUEStionable Status register is set.

- **Related Commands:** TRIGger:STARt:TIMer2, TRIGger:STARt:SOURce
- ***RST Condition:** 5.0E-8 seconds.

PERIOD VALUE TABLE

The following sample periods are available with the digitizer's internal 20 MHz reference oscillator.

Multiple	Period (Seconds)	Multiple	Period (Seconds)	Multiple	Period (Seconds)
1	5.0E-8	2	1.0E-7	4	2.0E-7
10	5.0E-7	20	1.0E-6	40	2.0E-6
100	5.0E-6	200	1.0E-5	400	2.0E-5
1000	5.0E-5	2000	1.0E-4	4000	2.0E-4
10,000	5.0E-4	20,000	1.0E-3	40,000	2.0E-3
100,000	5.0E-3	200,000	1.0E-2	400,000	2.0E-2
1,000,000	0.050	2,000,000	0.10	4,000,000	0.20
10,000,000	0.50	20,000,000	1.0	40,000,000	2.0
100,000,000	5.0	200,000,000	10.0	400,000,000	20

[:STARt]:TIMer2

TRIGger[:STARt]:TIMer2 <*period*> specifies the time period between each sampling event of the post-arm portion of a dual rate measurement. The time period must be a multiple of the reference oscillator period, with allowable multiples being 1,2,4,10,20,40,.....1E8, 2E8, 4E8.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
period	numeric	reference period to reference period * 4E8 | MINimum | MAXimum	seconds

Comments

- **Executable when initiated:** No
- **Coupled Command:** This command is coupled to the TRIGger:STARt:TIMer1 command and TRIGger:STARt:SOURce command as noted below:

  Unless TRIGger:STARt:SOURce is DTIMer, the settings of TRIGger:STARt:TIMer1 and TRIGger:STARt:TIMer2 are not coupled, and changing one will not affect the setting of the other.

  If TRIGger:STARt:SOURce is DTIM, then both TRIGger:STARt:TIMer1 and TRIGger:STARt:TIMer2 are used, and there is a coupling between the two settings. The relationship between the two settings is such that one of these two values must be exactly one reference oscillator period and the other must be a multiple (greater than 1.0) of the reference oscillator period. A record is kept of which setting was changed the most recently, and that setting (TRIGger:STARt:TIMer1 or TRIGger:STARt:TIMer2) is assumed to be the desired setting. For example, consider a reference oscillator period of 1µs; if TRIGger:STARt:TIMer2 was last changed to a value of 1µs while TRIGger:STARt:SOUR was set to DTIM, and the TRIGger:STARt:TIMer1 setting was also 1µs, TRIGger:STARt:TIMer1 would be changed to 2µs (the multiple 2.0 was chosen arbitrarily) so that both settings are not 1.0 times the reference period. Similarly, if TRIGger:STARt:TIMer2 were set to 4µs, and TRIGger:STARt:TIMer1 was some value greater than 1µs (like 2µs), TRIGger:STARt:TIMer1 would be automatically changed to1µs so that one of the two values is 1.0 times the reference period.

- If dual rate sampling is enabled (TRIGger:STARt:SOUR is DTIM), then the sample period specified by TIMer1 will be the sample rate for the pre-arm readings of the dual rate measurement, and TRIGger:STARt:TIMer2 will be the post-arm sample rate.
- Note that it is only necessary to set the longest sample rate if TRIGger:STARt:SOURce is DTIM. The other setting will be automatically forced to be one reference oscillator period due to the coupling between the two rates when TRIGger:STARt:SOURce is DTIM.
- If TRIGger:STARt:SOUR is TIMer, then only the sample rate specified by TIMer1 is used.
- If dual rate sampling is not enabled (TRIGger:STARt:SOUR DTIM), then the TRIGger:STARt:TIMer2 setting is retained, but not used.
- If the HP E1429 can not sample within 1 percent of the period specified by TRIGger:STARt:TIMer2, then the TIME bit (bit 2) in the QUEStionable Status register is set.

- **Related Commands:** TRIGger:STARt:TIMer1, TRIGger:STARt:SOURce
- ***RST Condition:** 1.0E-7 seconds.

PERIOD VALUE TABLE

The following sample periods are available with the digitizer's internal 20 MHz reference oscillator.

Multiple	Period (Seconds)	Multiple	Period (Seconds)	Multiple	Period (Seconds)
1	5.0E-8	2	1.0E-7	4	2.0E-7
10	5.0E-7	20	1.0E-6	40	2.0E-6
100	5.0E-6	200	1.0E-5	400	2.0E-5
1000	5.0E-5	2000	1.0E-4	4000	2.0E-4
10,000	5.0E-4	20,000	1.0E-3	40,000	2.0E-3
100,000	5.0E-3	200,000	1.0E-2	400,000	2.0E-2
1,000,000	0.050	2,000,000	0.10	4,000,000	0.20
10,000,000	0.50	20,000,000	1.0	40,000,000	2.0
100,000,000	5.0	200,000,000	10.0	400,000,000	20

The VINStrument subsystem operates with the ARM and TRIGger subsystems to control the virtual instrument features of the HP E1429A/B. These features include the ability to use the Local bus and the VME (VXI data transfer) bus to obtain buffered measurement data from memory or real time measurement data directly from the analog to digital converter.

Local Bus transfers

There are two ways to transfer data over the Local bus:

Data can be transferred directly from the analog to digital converter(s) using the ARM and TRIGger subsystems in conjunction with the INITiate command. Everything proceeds exactly the same as if readings were going to HP E1429B internal memory, except in this case, readings are going to the Local bus as well as to internal memory. If the consumer on the Local Bus is unable to maintain the data transfer (sampling) rate, then the data going out over the Local bus is lost and an error is indicated.

The second method of transferring data over the Local bus is to empty the HP E1429B internal memory after the measurements have occurred. This transfer will automatically proceed after the measurements are completed if the user has previously set the VINStrument:CONFigure:LBUS:MODE to APPend, GENerate, or INSert, and if VINStrument:CONFigure:LBUS:FEED is one of the "MEM:xxx" choices. If the measurement is aborted with the ABORt command, or if VINStrument:LBUS:MODE is OFF or PIPline during the measurement, then no automatic transfer is attempted. Instead, the VINStrument:CONFigure:LBUS:MEMory:INITiate command must be used to start the transfer after the MODE and FEED have been set to the desired values. When using this method, the ARM source is automatically set to IMMediate, and the trigger source is set to LBUS. This allows the receiving module(s) on the right to control the data transfer, and assures that transfers occur only when the receiving module is ready to receive data; thus no data will ever be lost. The trigger and arm sources are returned to their previous values with the next INITiate command.

VME (VXI data transfer) Bus transfers

When data is transferred over the VME bus directly from the A/D converter, a read of VME A24 address space, offset 12 ($0C_{16}$), causes a measurement to be taken and transferred all in the same read cycle. During the read cycle, the HP E1429 takes a reading and puts the data into the register before the read cycle completes. The ARM sources may be set to any legal source for this mode, but TRIGger:START:SOURce must be set to VME. Selections can be made using the VINStrument:CONFigure:VME:FEED command such that a single read produces data from only one channel (16 bits), or both channels simultaneously (32 bits).

When the data is transferred post measurement, completion of the INITiate:IMMediate command will automatically configure for a VME transfer from memory, based on the settings of the VINStrument:VME:FEED command. When the measurement has completed and the VME transfer has been set up by the HP E1429, bit 1 (Memory Read Enable) of the A24 memory control register (base + 21_{16}) will go high (1). At this point, data transfer can be initiated by the receiver by reading the A24 data register (base + $0C_{16}$). Again, the VINStrument:VME:FEED command is used to specify whether a single read will produce one channel (16 bits) of data, or two channels (32 bits) of data. The VINStrument:VME:MEMory:INITiate command will also configure for a post measurement VME data transfer, but it need not be sent unless it is desirable to read the same data multiple times.

Subsystem Syntax

```
VINStrument
    [:CONFigure]
        :LBUS
            :FEED <source>
            :MEMory
                :INITiate                   [no query]
            [:MODE] <mode>
            :RESet                          [no query]
            :SEND
                :POINts <count>
                    :AUTO <mode>
        :TEST
            :DATA <voltage_list>                [no query]
        :VME
            :FEED <source>
            :MEMory
                :INITiate                   [no query]
            [:MODE] <mode>
            :SEND
                :ADDRess
                    :DATA?                  [query only]
    :IDENtity?
```

[:CONFigure]:LBUS:FEED

VINStrument[:CONFigure]:LBUS:FEED <*source* > indicates the source of the data which will be output to the Local bus. The data source may be channel 1, channel 2, or both channels. The data may come from memory or directly from the A/D converter(s). Sources beginning with "MEMory: " are the post measurement modes, sources beginning with "CONVerter: " are the real time modes. The possible sources are:

" MEMory:CHANnel1" : Channel 1 memory is the data source for the Local bus. Two bytes per reading will be output to the bus.

" MEMory:CHANnel2" : Channel 2 memory is the data source for the Local bus. Two bytes per reading will be output to the bus.

" MEMory:BOTH" : Both channels of memory are the data source for the Local bus. In this mode, the channel 2 reading is output first, followed by the channel 1 reading. Four bytes for each set of readings will be output to the bus.

" CONVerter:CHANnel1" : The channel 1 A/D converter is the data source for the Local bus. Two bytes per reading will be output to the bus.

" CONVerter:CHANnel2" : The channel 2 A/D converter is the data source for the Local bus. Two bytes per reading will be output to the bus.

" CONVerter:BOTH" : Both A/D converters are the data source for the Local bus. In this mode, the channel 2 reading is output first, followed by the channel 1reading. Four bytes for each set of readings will be output to the bus.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
source	string	"CONVerter:BOTH" | "CONVerter:CHANnel1" | "CONVerter:CHANnel2" | "MEMory:BOTH" | "MEMory:CHANnel1" | "MEMory:CHANnel2" |	none

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes, this command is set but ignored if VINStrument:CONFigure:LBUS:MODE is not GENerate, APPend, or INSert.

- When VINStrument:CONFigure:LBUS:FEED is one or both A/D converters, care must be taken that other active instruments in the pipeline can maintain the data generation rate. If data is available from the A/D converter but the Local bus is busy and can not accept it, the data is lost and error 1019; "Data loss detected during LBUS transfer" is reported.

- For VINStrument:CONFigure:LBUS:FEED "MEMory:BOTH" and VINStrument:CONFigure:LBUS:FEED "CONVerter: BOTH", data is sent interleaved with channel 2 reading 1 as the first point.

- **Related Commands:** VINStrument:CONFigure:LBUS:MODE, ARM subsystem, TRIGger subsystem, VINStrument:CONFigure:LBUS:MEMory:INITiate

- ***RST Condition:** VINStrument:CONFigure:LBUS:FEED "MEMory:BOTH"

Example **Send channel 1 memory data to consumer automatically after measurement**

VINS:CONF:LBUS:MODE GEN	*Set this module's mode to GENerate data for the Local bus*
VINS:CONF:LBUS:FEED "MEM:CHAN1"	*Set data source to channel 1 memory*
Set up other modules to right of this one	*Last module on right must be in VINS:LBUS:MODE CONSume*
INIT	*Start the measurement, data will be transferred over the Local bus as soon as the measurement completes*

[:CONFigure]:LBUS:MEMory:INITiate

VINStrument[:CONFigure]:LBUS:MEMory:INITiate causes the instrument to begin the process of transferring data from memory out over the Local bus. If the FEED and MODE are set correctly (e.g. MODE = APPend, GENerate, or INSert) before the measurement is taken, this command is not necessary because the data will automatically be sent. If, however, the measurement was aborted, or if it is necessary to change the MODE or FEED after the measurement has completed, then executing this command will start the data transfer.

Comments

- **Executable when initiated:** No

- **Coupled command:** Yes. This command will error if VINStrument:CONFigure:LBUS:FEED is not "MEMory:xxx", if VINStrument:CONFigure:LBUS:MODE is OFF, or if the HP E1429 is already INITiated.

- This command results in an error if VINStrument:CONFigure:LBUS:FEED is not one of the "MEMory" choices (i.e. "MEMory:BOTH", etc.).
- If the data in memory is in multiple segments (ARM:STARt:COUNt > 1 and SENSe:SWEep:POINts:DELay < 0), then there will be a small time delay between transfer of each segment while the CPU switches the memory address to point to the next segment.
- This command has no query form.
- **Related Commands:** VINStrument:CONFigure:LBUS:MODE
- ***RST Condition:** None

Example **Send both channels of memory data to consumer**

VINS:CONF:LBUS:MODE GEN	*Set this module's mode to GENerate data for the Local bus*
VINS:CONF:LBUS:FEED "MEM:BOTH"	*Set data source to be both channels*
Set up modules to the right of this one	*Last module on right must be in VINS:LBUS:MODE CONSume*
VINS:LBUS:MEM:INIT	*Begin sending data from memory out over the Local bus*

[:CONFigure]:LBUS[:MODE]

VINStrument[:CONFigure]:LBUS[:MODE] <*mode*> selects the operating mode for the VXI Local bus. The available modes are:

APPend: Local bus data is received from the left, and passed on to the right until an end of frame is detected. When end of frame is received from the left side, all data from this module is appended, followed by an end of block flag and a new end of frame flag. After sending the end of frame flag, the module enters the paused state. This mode requires a module to the left that is in GENerate mode. The mode is not active until either an INITiate command or a VINStrument:LBUS:MEMory:INITiate command is sent.

GENerate: Local bus data originates in this module and is passed to the right, followed by an end of frame flag. The mode is not active until either an INITiate command or a VINStrument:LBUS:MEMory:INITiate command is sent.

INSert: Local bus data is inserted onto the bus from this module. The module will place its data out onto the Local bus with an end of block flag at the end and no end of frame flag. The module will then pass through (pipeline) any data it receives from the left, and will enter the paused state when an end of frame flag is received from the left. This mode requires at least one module to the left

which is in GENerate mode. The mode is not active until either an INITiate command or a VINStrument:CONFigure:LBUS:MEMory:INITiate command is sent.

OFF: The Local bus interface is disabled immediately upon receipt of this command. Local bus data is neither used nor passed through.

PIPeline: Local bus data is passed through and not altered. This mode becomes effective immediately upon receipt of this command. Select this mode when data should be transparently passed through the HP E1429B. The module will remain in the PIPeline mode even after an end of frame flag is received; therefore, it is necessary to change modes to take the module out of PIPeline mode.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	discrete	APPend|GENerate| INSert|OFF| PIPeline	none

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes
- **Related Commands:** ARM subsystem, TRIGger subsystem, VINStrument:CONFigure:LBUS:FEED, VINStrument:CONFigure:LBUS:MEMory:INITiate, INITiate, VINStrument:CONFigure:VME:MODE
- ***RST Condition:** VINStrument:CONFigure:LBUS:MODE OFF

Example

Setting the Local Bus operation mode

```
VINS:LBUS PIP
```

Set pipeline (pass through) mode; this becomes active immediately

[:CONFigure]:LBUS:RESet

VINStrument[:CONFigure]:LBUS:RESet will reset the digitizer's Local bus chip. This command should be used when it is necessary to put the Local bus chip into a known state without altering any other digitizer settings (TRIGger:STARt:COUNt, TRIGger:STARt:SOURce, etc.). The *RST command also resets the Local bus chip, but *RST forces all instrument settings to initial values.

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- The HP E1429B Local bus chip must be reset after **each** data transfer. When resetting the Local bus chip, the Local bus chips on all devices to the right of the HP E1429B must also be reset in a left-to-right sequence. Refer to the product documentation for information on how a particular device's Local bus chip is reset.
- **Related Commands:** *RST, ABORt
- ***RST Condition:** none

Example **Reset the Local bus chip**

```
VINS:LBUS:RES                    Reset the chip
```

[:CONFigure]:LBUS:SEND:POINts

VINStrument[:CONFigure]:LBUS:SEND:POINts<*count*> specifies how many readings will be output over the Local bus per block. There are two possible ***count*** settings: 1 point per feed channel or all points per feed channel. Normally, there is no need to set a ***count*** with this command because with VINStrument:CONFigure:LBUS:SEND:POINts:AUTO ON (the power-on and *RST setting), the number of points will automatically be matched to the current setting of VINStrument:CONFigure:LBUS:FEED. Therefore, the count will normally be TRIGger:COUNt * 2 if VINS:LBUS:FEED is "xxx:BOTH" or TRIGger:COUNt * 1 if the feed is "xxx:CHAN1" or "xxx:CHAN2".

If VINStrument:CONFigure:LBUS:SEND:POINts:AUTO is set to OFF, then the number of points sent per block must either be 2 (for feed "CONV:BOTH") or 1 (for feeds "CONV:CHAN1 and "CONV:CHAN2"). This setting is only allowed when the feed is one of the "CONV:xxx" (direct from A/D converter) settings. The combination of VINStrument:CONFigure:LBUS:SEND:POINts:AUTO OFF and VINStrument:CONFigure:LBUS:SEND:POINts 1 or 2 is needed only when the goal is to multiplex readings directly from multiple digitizers onto the Local bus.

Parameters MINimum and MAXimum will both set the same value, which is the number of readings that will be transferred as determined by the current VINStrument:CONFigure:LBUS:SEND:POINts:AUTO and VINStrument:CONFigure:LBUS:FEED settings.

Comments

- **Executable when initiated:** No

VINStrument[:CONFigure]:LBUS:SEND:POINts:AUTO

- **Coupled command:** VINStrument:CONFigure:LBUS:SEND:POINts:AUTO and VINStrument:CONFigure:LBUS:FEED settings determine the allowable values of this command.
- For a more complete discussion and example of when this command should be used, see "Local Bus Data Transfers" in Chapter 3.
- If a measurement was halted using the ABORt command and "MEM:xxx" is one of the VINStrument:CONFigure:LBUS:FEED choices, the query form of this command will return the number of points that will actually be transferred (assuming VINStrument:CONFigure:LBUS:SEND:POINts:AUTO ON) when the VINStrument:CONFigure:LBUS:MEMory:INITiate command is invoked.
- **Related Commands:** VINStrument:CONFigure:LBUS:FEED
- ***RST Condition:** 2

Example **Query number of readings output per block**

VINS:LBUS:SEND:POIN? *Query points*

[:CONFigure]:LBUS:SEND:POINts:AUTO

VINStrument[:CONFigure]:LBUS:SEND:POINts:AUTO <*mode*> determines how the digitizer will send its data over the Local bus. In most cases AUTO ON would be used, which sends all of a measurement's data over the Local bus followed by an end-of-block flag and an end-of-frame flag.
The AUTO OFF setting is a special setting which is only used when it is desirable for multiple digitizers to output A/D data from each (common) trigger event onto the Local bus at approximately the same time (i.e. in parallel). In this mode, the end-of-block and end-of-frame flags are sent with each reading (trigger event). This mode requires that all digitizers are set to the same trigger count and have their trigger sources synchronized such that each digitizer's data is available to the Local bus before the next trigger event can occur on any digitizer in the data stream. Otherwise, data loss will occur and an error will be indicated by one or more of the digitizers.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	Boolean	ON | OFF | 1 | 0	none

Comments

- **Executable when initiated:** No

- **Coupled command:** Yes, VINStrument:CONFigure:LBUS:FEED and VINStrument:CONFigure:LBUS:SEND:POINts are coupled to this command.

- If VINStrument:CONFigure:LBUS:SEND:POINts:AUTO is OFF, the measurement does not complete normally. **After the data has been taken and transferred, the Local bus is left in a running state. Therefore, it is necessary to send the ABORt command to each digitizer in the data stream (in a left to right sequence) after the data is transferred and before proceeding with the next setup or INITiate**. Additionally, any other Local bus devices (non-digitizers) will need to reset their Local bus chips in a left to right sequence. See "Local Bus Data Transfers" in Chapter 3 for more information.

- **Related Commands:** VINStrument[:CONFigure]:LBUS:FEED, VINStrument[:CONFigure]:LBUS:SEND:POINts, ABORt

- ***RST Condition:** ON

Example **Setup for multiple transfers of direct A/D data**

```
VINS:LBUS:FEED "CONV:BOTH"          Send both channels directly from A/D converter
VINS:LBUS:SEND:POIN 2; POIN:AUTO OFF
                                    Set up to send end-of-block and end-of-frame every two readings
```

[:CONFigure]:TEST:DATA

VINStrument[:CONFigure]:TEST:DATA <*voltage_list*> configures the HP E1429B for Local bus testing, and transmits the data given in *voltage_list*. This data will be temporarily stored in internal memory before being output at the fastest possible speed over the Local bus. The data will be stored into alternate channels, beginning with channel 2. The data is output over the Local bus the same way (interleaved, with channel 2 data point 1 first). Regardless of their current settings, VINStrument:CONFigure:LBUS:FEED is set to "MEM:BOTH", VINStrument:CONFigure:LBUS:MODE is set to GENerate, and VINStrument:CONFigure:VME:MODE is set to OFF. These changes will remain in effect after the command has completed.

Parameters The *voltage_list* must be an IEEE-488.2 definite length block containing values in 16-bit integer format, with the most significant byte being the first byte in a pair (i.e. Motorola format).

The legal range of values is -32768 to 32767. Only the upper 12 bits are stored into memory. The lower 4 bits are ignored, and the actual values sent over the Local bus for these lower 4 bits are determined by the current setting of the DIAGnostic:CHANnel:LABel command.

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes. VINStrument:CONFigure:LBUS:FEED will be changed to "MEMory:BOTH", VINStrument:CONFigure:LBUS:MODE will be set to GENerate, and VINStrument:CONFigure:VME:MODE will be set to OFF.
- The module receiving the data must have been previously set up so that it is ready to CONSume data.
- When testing the Local bus, you must send a minimum of 28 bytes and the byte count must be a multiple of 4. If less than 28 bytes are sent, error -109 "Missing parameter" occurs. If the byte count is not a multiple of 4, error -161 "Invalid block data" occurs.
- Because MEMory:BOTH is the FEED, the first data point is sent from channel 2, the second data point is sent from channel 1, the third data point from channel 2, and so on.
- As mentioned above, the lower 4 bits of each 16-bit reading sent out over the Local bus are determined by the current setting of DIAGnostic:CHANnel:LABel for the channel the data was stored into. These 4 bits may not have the same value as the data sent with this command. Therefore, the consuming module may receive slightly different data than what was downloaded with this command.
- **Related Commands:** VINStrument:CONFigure:LBUS:FEED, VINStrument:CONFigure:LBUS:MODE
- ***RST Condition:** none

Example

Testing Local Bus operation

VINS:TEST:DATA #240<40 bytes of data>

Send 40 bytes (10 readings per channel), using IEEE-488.2 definite length block

[:CONFigure]:VME:FEED

VINStrument[:CONFigure]:VME:FEED <*source*> selects which data source will feed the VME (VXI data transfer) bus. The bus is driven by reading the data register, offset 12 ($0C_{16}$) in A24 address space. Sources beginning with "MEMory: " are the post measurement modes, sources beginning with "CONVerter: " are the real time modes. The possible sources are:

" MEMory:CHANnel1" : Channel 1 memory is the data source for the VME bus. One 16-bit reading is returned.

" MEMory:CHANnel2" : Channel 2 memory is the data source for the VME bus. One 16-bit reading is returned.

" MEMory:BOTH" : Both channels of memory are the data source for the VME bus. In this mode, channel 1 will be output the first time the data register is accessed, channel 2 is output the second time the data register is accessed. One 16-bit reading is returned with each access.

" MEMory:BOTH32" : Both channels of memory are the data source for the VME bus. In this mode, accessing the data register returns a 32-bit number where the high order 16 bits are the channel 2 reading and the low order 16 bits are the channel 1 reading.

" CONVerter:CHANnel1" : The channel 1 A/D converter is the data source for the VME bus. One 16-bit reading is returned.

" CONVerter:CHANnel2" : The channel 2 A/D converter is the data source for the VME bus. One 16-bit reading is returned.

" CONVerter:BOTH" : Accessing the data register triggers both A/D converters at the same time, and one 16-bit reading (channel 1) is returned. Accessing the data register a second time returns the second 16-bit reading (channel 2), but does not trigger the A/Ds.

" CONVerter:BOTH32" : Accessing the data register triggers both A/D converters at the same time, and one 32-bit number is returned. The high order 16 bits are the channel 2 reading, and the low order 16 bits are the channel 1 reading.

VINStrument[:CONFigure]:VME:MEMory:INITiate

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
source	string	"CONVerter:BOTH" | "CONVerter:BOTH32" | "CONVerter:CHANnel1" | "CONVerter:CHANnel2" | "MEMory:BOTH" | "MEMory:BOTH32" | "MEMory:CHANnel1" | "MEMory:CHANnel2" |	none

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes. This command is coupled to the TRIGger subsystem.
- If the data in memory is in multiple segments, then there will be a small delay (630 µs) between segments while the CPU switches the memory address to point to the next segment. It is possible to determine when data is available again by monitoring bit 1 of the A24 arm status register (base + 43_{16}). This bit goes high when the data is again ready for transfer. See "VME Bus Data Transfers" in Chapter 3 for additional information.
- **Related Commands:** TRIGger:STARt:COUNt, VINStrument:CONFigure:VME:MODE, VINStrument:CONFigure:VME:MEMory:INITiate, TRIGger:STARt:SOURe
- ***RST Condition:** VINStrument:VME:FEED "MEMory:BOTH32"

Example

Reading both channels out to the VME (VXI data transfer) bus

TRIG:SOUR VME	*Set up trigger system for VME transfer*
VINS:VME GEN	*Set GENerate mode*
VINS:VME:FEED " MEM:BOTH32 "	*Set data source to be both channels*
INIT	*Begin the transfer*

[:CONFigure]:VME:MEMory:INITiate

VINStrument[:CONFigure]:VME:MEMory:INITiate configures the HP E1429 for data transfer over the VME bus when the data register in A24 address space is read. The configuration is done automatically if the INITiate:IMMediate command is executed while VINStrument:CONFigure:VME:MODE is GENerate and any of the "MEMory:xxx" settings is the selection for

VINStrument:CONFigure:VME:FEED. However, if these settings were not in effect when the measurement was taken, then VINStrument:CONFigure:VME:MEMory:INITiate must be sent before data is retrieved from memory via the data register.

This command results in an error if VINStrument:CONFigure:VME:MODE is OFF.

This command has no query form.

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes, this command will error if VINStrument:CONFigure:VME:MODE is OFF, or if the HP E1429 is already initiated or transferring data.
- If the data in memory is in multiple segments, then there will be a small time delay (630 µs) between segments while the CPU switches the memory address to point to the next segment. It is possible to determine when data is available again by monitoring bit 1 of the A24 arm status register (base + 43_{16}). This bit goes high when the data is again ready for transfer. See Chapter 3 for additional information.
- **Related Commands:** VINStrument:CONFigure:VME:MODE, VINStrument:CONFigure:VME:FEED, ARM subsystem, TRIGger subsystem
- ***RST Condition:** None

Example

Send channel 2 of memory data out over the VXIbus, whenever the VME data register is accessed

VINS:VME:MODE GEN	*Set mode to GENerate*
VINS:VME:FEED "MEM:CHAN2"	*Set data source to be memory channel 2*
VINS:VME:MEM:INIT	*Begin sending data to VXIbus when the VME A24 register is read*

[:CONFigure]:VME[:MODE]

VINStrument[:CONFigure]:VME[:MODE] <*mode*> selects the operating mode for the VME bus. The only available modes are GENerate and OFF.

This command is used when it is desirable to transfer data over the VME bus by reading the data register at offset 12 ($0C_{16}$) in A24 address space.

VINStrument[:CONFigure]:VME:SEND:ADDRess:DATA?

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mode	discrete	GENerate|OFF	none

Comments

- **Executable when initiated:** No
- **Coupled command:** Yes, VINStrument:CONFigure:VME:MODE GENerate is not allowed unless VINStrument:CONFigure:LBUS:MODE is set to OFF or PIPeline.
- **Related Commands:** VINStrument:CONFigure:VME:FEED
- ***RST Condition:** VINStrument:CONFigure:VME:MODE OFF

Example

Setting the VXIbus data transfer bus operation mode

```
VINS:VME GEN                    Set GENerate mode
```

[:CONFigure]:VME:SEND:ADDRess:DATA?

VINStrument[:CONFigure]:VME:SEND:ADDRess:DATA? returns two values: A24,12. A24 indicates that the HP E1429's A24 address space should be used for reading measurement data, and 12 ($0C_{16}$) is the offset of the data register in A24 address space.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** VINStrument:CONFigure:VME:MODE, TRIGger:STARt:SOURce, VINStrument:CONFigure:VME:MEMory:INITiate
- ***RST Condition:** A24,12

Example

Querying the A24 address space offset

```
VINS:VME:SEND:ADDR:DATA?        Query A24 offset for data reads
```

:IDENtity?

VINStrument:IDENtity? returns a response consisting of 4 fields, indicating the virtual instrument capability of the HP E1429:

```
HEWLETT-PACKARD VIRTUAL INSTRUMENT,ANY  ATOD,0,A.01.00
```

The first and last fields indicate that the HP E1429 conforms to revision A.01.00 of HP's Virtual Instrument/Local Bus System Specification. The second field indicates that the HP E1429 is a analog-to-digital converter. The third field is reserved for future use.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** none

Example

Querying virtual instrument capability

```
VINS:IDEN?                    Query capability
```

IEEE-488.2 Common Commands

This section describes the IEEE-488.2 Common Commands implemented in the HP E1429. The table below shows the commands listed by functional group; however, commands are listed alphabetically in the reference. Examples are shown in the reference when the command has parameters or returns a non-trivial response; otherwise, the command string is as shown in the table. For additional information, refer to IEEE Standard 488.2-1987.

Category	Command	Title
System Data	*IDN?	Identification Query
	*PUD <*data*>	Protected User Data Command
	*PUD?	Protected User Data Query
Internal Operations	*LRN?	Learn Device Setup Query
	*RST	Reset Command
	*TST?	Self Test Query
Synchronization	*OPC	Operation Complete Command
	*OPC?	Operation Complete Command
	*WAI	Wait-to-Continue Command
Macro	*DMC <*name*>,<*data*>	Define Macro Command
	*EMC <*enable*>	Enable Macro Command
	*EMC?	Enable Macro Query
	*GMC? <*name*>	Get Macro Contents Query
	*LMC?	Learn Macro Query
	*PMC	Purge Macros Command
	*RMC <*name*>	Remove Individual Macro Command
Status & Event	*CLS	Clear Status Command
	*ESE <*mask*>	Standard Event Status Enable Command
	*ESE?	Standard Event Status Enable Query
	*ESR?	Standard Event Status Register Query
	*SRE	Service Request Enable Command
	*SRE?	Service Request Enable Query
	*STB?	Read Status Byte Query
Trigger	*TRG	Trigger Command
Stored Settings	*RCL	Recall Command
	*SAV	Save Command

*CLS

***CLS** clears the Standard Event Status Register, the Operation Status Register, the Questionable Signal Register, and the error queue. This clears the corresponding summary bits (3, 5, & 7) in the Status Byte Register. *CLS does not affect the enable masks of any of the status registers.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** STATus:PRESet
- ***RST Condition:** none

*DMC

***DMC <*name*>,<*data*>** creates a macro with the specified name and assigns zero, one, or a sequence of commands to the name. The sequence may be composed of SCPI and/or Common Commands.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	string data	1 through 12 characters	none
data	block data *or* string	any valid command sequence	none

Comments

- Legal macro names must start with an alphabetic character and contain only alphabetic, numeric, and underscore ("_") characters. Alphabetic character case (upper vs. lower) is ignored.

  The name is allowed to be the same as a SCPI command, but may be not be the same as a Common Command. When the name is the same as a SCPI command, the macro rather than the command will be executed when the name is received if macro usage is enabled. The SCPI command will be executed if macro usage is disabled.

- The <*data* > in the *DMC command is parsed by the digitizer when the *DMC command is executed.

- **Executable when initiated:** Yes

- **Coupled command:** No
- **Related Commands:** *EMC, *GMC, *LMC, *RMC
- ***RST Condition:** none; macro definitions are unaffected
- **Power-On Condition:** no macros are defined

Example **Define macro to start measurement**

```
*DMC  " RESTART","ABOR;INIT"          Define macro
```

*EMC and *EMC?

***EMC <enable>** enables and disables macro usage. When *enable* is zero, macros usage is disabled. Any non-zero value enables macro usage.

The query form returns 1 if macro usage is enabled, 0 if disabled.

Comments

- Macro definitions are not affected by this command.
- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** macro usage is disabled
- **Power-On Condition:** macro usage is enabled

*ESE and *ESE?

***ESE <mask>** enables one or more event bits of the Standard Event Status Register to be reported in bit 5 (the Standard Event Status Summary Bit) of the Status Byte Register. *Mask* is the sum of the decimal weights of the bits to be enabled.

The query form returns the current enable mask.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mask	numeric	0 through 255	none

A 1 in a bit position enables the corresponding event; a 0 disables it.

Comments

- **Executable when initiated:** Yes

- **Coupled command:** No
- **Related Commands:** *ESR?, *SRE, *STB?
- ***RST Condition:** unaffected
- **Power-On Condition:** no events are enabled

Example **Enable all error events**

```
*ESE 60                    Enable error events
```

*ESR?

***ESR?** returns the value of the Standard Event Status Register. The register is then cleared (all bits 0).

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** none
- **Power-On Condition:** register is cleared

*GMC?

***GMC? <name>** returns the definition of the specified macro in IEEE-488.2 definite block format.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
name	string data	defined macro name	none

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** *DMC
- ***RST Condition:** none
- **Power-On Condition:** no macros are defined

Example **Query macro definition**

```
*GMC? " RESTART"                    Query macro definition
```

*IDN?

***IDN?** returns indentification information for the HP E1429. The response consists of four fields:

```
HEWLETT-PACKARD,E1429,0,A.01.00
```

The first two fields identify this instrument as model number E1429 manufactured by Hewlett-Packard. The third field is 0 since the serial number of the HP E1429 is unknown to the firmware. The last field indicates the revision level of the firmware.

Note The firmware revision field will change whenever the firmware is revised. A.01.00 is the initial revision. The first two digits indicate the major revision number, and increment when functional changes are made. The last two digits indicate bug fix level.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** none
- **Power-On Condition:** register is cleared

*LMC?

***LMC?** returns a comma-separated list of quoted strings, each containing the name of a macro. If no macros are defined, a single null string ("") is returned.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** *DMC
- ***RST Condition:** none
- **Power-On Condition:** no macros are defined

*LRN?

***LRN?** returns a sequence of commands that may be re-sent to the HP E1429 to return it to its current programming state.

Only those commands that are affected by *RST are included in the sequence. Exceptions include MEMory:BATTery:STATe, the STATus subsystem commands, and the CALibration:SECurity command state.

Note *LRN? should be sent singly in a program message, since the number of commands in the returned sequence is large, and may vary depending on firmware revision.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *RCL, *RST, *SAV
- ***RST Condition:** none

*OPC

***OPC** causes the HP E1429 to wait for all pending operations to complete. The Operation Complete bit (bit 0) in the Standard Event Status Register is then set.

If STATus:OPC:INITiate OFF is set, the Operation Complete bit will be set when all commands received prior to the *OPC have been executed. If ON is set, *OPC waits for the digitizer to return to the idle state before setting the Operation Complete bit. No other commands will be executed until the Operation Complete bit is set.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *OPC?, *WAI
- ***RST Condition:** none

*OPC?

***OPC?** causes the HP E1429 to wait for all pending operations to complete. A single ASCII "1" is then placed in the output queue.

If STATus:OPC:INITiate OFF is set, the ASCII "1" will be placed in the output queue when all commands received prior to the *OPC? have been executed. If ON is set, *OPC? waits for the digitizer to return to the idle state before placing the "1" in the output queue. No other commands will be executed until the "1" is placed in the output queue.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *OPC, *WAI
- ***RST Condition:** none

*PMC

***PMC** purges all macro definitions.

Comments

- Use the *RMC command to purge an single macro definition.
- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *DMC, *RMC
- ***RST Condition:** none

*PUD and *PUD?

***PUD <*data*>** stores the specified data in the HP E1429's non-volatile calibration memory. The data must be sent in IEEE-488.2 definite or indefinite block format. Calibration security must have been previously disabled.

The query form returns the current protected user data in IEEE-488.2 definite block format. The query form may be executed regardless of the state of calibration security.

Note When shipped from the factory, the protected user data area contains information regarding when the HP E1429 was last calibrated.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mask	block data *or* string	0 through 63 characters	none

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** unaffected
- **Power-On Condition:** unaffected

Example

Setting the protected user data

```
*PUD #17Unit #5                    Set data to "Unit #5"
```

*RCL

***RCL <*number*>** restores a previously stored programming state from one of the 10 possible stored state areas. *Number* indicates which of the stored state areas should be used.

This command affects the same command settings as does *RST. Notable exceptions include MEMory:BATTery:STATe, the STATus subsystem commands, and the CALibration:SECurity command state.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
number	numeric	0 through 9	none

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- **Related Commands:** *LRN?, *RST, *SAV

- ***RST Condition:** all saved states set to the same state as the *RST state

*RMC

RMC** <name***> purges only the specified macro definition.

NOTE: At printing time, *RMC is a command proposed and accepted for a revision and re-designation of IEEE-488.2.

Comments

- Use the *PMC command to purge all macro definitions in one command.
- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *DMC, *PMC
- ***RST Condition:** none

*RST

***RST** resets the HP E1429 as follows:

- Sets all commands to their *RST state.
- Aborts all pending operations including VME bus or Local bus transfers.
- Loads calibration constants from non-volatile calibration memory.

*RST does not affect:

- The state of VXIbus word serial protocol
- The output queue
- The Service Request Enable Register
- The Standard Event Status Enable Register
- The enable masks for the OPERation Status and Questionable Signal registers
- Calibration security state
- Protected user data
- The memory backup battery

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- ***RST Condition:** none

*SAV

***SAV <_number_>** stores the current programming state into one of the 10 possible stored state areas. *Number* indicates which of the stored state areas should be used.

This command stores the states of all commands affected by *RST. Exceptions include MEMory:BATTery:STATe, the STATus subsystem commands, and the CALibration:SECurity command state.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
number	numeric	0 through 9	none

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related Commands:** *LRN?, *RCL, *RST
- ***RST Condition:** unaffected
- **Power-on Condition:** all saved states set to the same state as the *RST state

*SRE and *SRE?

***SRE <_mask_>** specifies which bits of the Status Byte Register are enabled to generate a service request (VXIbus *reqt* signal). Event and summary bits are always set and cleared in the Status Byte Register regardless of the enable mask. *Mask* is the sum of the decimal weights of the bits to be enabled.

The query form returns the current enable mask.

Parameters

Parameter Name	Parameter Type	Range of Values	Default Units
mask	numeric	0 through 255	none

A 1 in a bit position enables service request generation when the corresponding Status Byte Register bit is set; a 0 disables it.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No

- ***RST Condition:** unaffected
- **Power-On Condition:** no bits are enabled

Example **Enable service request on Message Available bit**

***SRE 16** *Enable request on MAV*

*STB?

***STB?** returns the value of the Status Byte Register. Bit 6 (decimal weight 64) is set if a service request is pending. STB? should not be used to read the Status Byte register if a service request is generated by a message available (MAV) condition.

Comments

- *STB? is a query. Thus, sending the command in response to a MAV condition will generate Error -410 "Query interrupted".
- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *SRE
- ***RST Condition:** none

*TRG

***TRG** is the command equivalent of the HP-IB Group Execute Trigger and the VXIbus Trigger word serial command and has exactly the same effect.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** ARM and TRIGger subsystem SOURce commands
- ***RST Condition:** none

*TST?

***TST?** causes the HP E1429 to execute its internal self-test and return a value indicating the results of the test.

A zero (0) response indicates that the self-test passed. A one (1) response indicates that the test failed. A failure also generates an error message with information on why the test failed. Additional information on the failure is provided by the DIAGnostic:TEST? command.

When the test completes, all commands are set to their *RST values.

Caution **Executing the self-test using *TST? erases all data in the digitizer's non-volatile memory.**

Comments

- **Executable when initiated:** No
- **Coupled command:** No
- ***RST Condition:** none

*WAI

***WAI** causes the HP E1429 to wait for all pending operations to complete before executing any further commands.

If STATus:OPC:INITiate OFF is set, command execution resumes when all commands received prior to *WAI have been executed. If ON is set, *WAI waits for the digitizer to return to the idle state before resuming command execution.

Comments

- **Executable when initiated:** Yes
- **Coupled command:** No
- **Related commands:** *OPC, *OPC?
- ***RST Condition:** none

Table 4-1. HP E1429A/B Command Quick Reference.

Subsystem	Commands	Description
ARM	ARM[:STARt]:COUNt <*count* >	Specifies the number of measurement cycles (bursts) to occur.
	ARM[:STARt]:DELay <*period* >	Delay from when the digitizer is armed to when it enters the wait-for- trigger state.
	ARM[:STARt][:IMMediate]	Places the digitizer in the wait-for-trigger state, independent of the selected ARM:STARt:SOURce.
	ARM[:STARt]:LEVel[<cha*n* >]:NEGative <*voltage* >	Selects the (negative-going) signal level which arms the digitizer.
	ARM[:STARt]:LEVel[<*chan* >]:POSitive <*voltage* >	Selects the (positive-going) signal level which arms the digitizer.
	ARM[:STARt]:SLOPe[<*n* >] <*edge* >	Selects the edge: positive, negative, or either, which will arm the digitizer.
	ARM[:STARt]:SOURce[<*n* >] <*source* >	Sets the digitizer arm source.
CALibration	CALibration[<*chan* >]:COUNt?	Returns a number that indicates how often the digitizer has been calibrated.
	CALibration[<*chan* >]:DATA <*block_data* >	Manually sets or queries the calibration constants.
	CALibration[<*chan* >]:DELay	Calibrates the A/D converter delay constant.
	CALibration[<*chan* >]:GAIN [<*readings* > [,<*period* >[,<*flag* >]]]	Performs a gain calibration using the specified number of readings and sample rate.
	CALibration[<*chan* >]:SECure:CODE <*code* >	Sets the code required to disable calibration security.
	CALibration[<*chan* >]:SECure:STATe <*mode* > [,<*code* >]	Enables/disables calibration security.
	CALibration[<*chan* >]:STORe	Stores the currently selected channel's calibration constants into non-volatile memory.
	CALibration[<*chan* >]:STORe:AUTO <*mode* >	Selects whether or not calibration constants will be automatically stored.
	CALibration[<*chan* >]:VALue <*number* >	Specifies the voltage level at the input.
	CALibration[<*chan* >]:ZERO [<*readings* > [,<*period* >,[<*mode* >]]]	Performs a calibration of the zero offset.
CONFigure	CONFigure[<*chan* >]:ARRay[:VOLTage][:DC] (<*size*>) [,<*expected value* >[,<*resolution* >]] [,(@<*input port*>)]	Configures the digitizer for <size> number of readings on the specified channel and input port.
DIAGnostic	DIAGnostic:CALibration[<*chan* >]:CONVerge?	Returns convergence data from the latest CAL:ZERO or CAL:GAIN.
	DIAGnostic:CALibration[<*chan* >]:GAIN:SENSitivity?	Returns the sensitivity constant used during the last CAL:GAIN command.
	DIAGnostic:CALibration[<*chan* >]:ZERO:SENSitivity?	Returns the sensitivity constant used during the last CAL:ZERO command.
	DIAGnostic:CHANnel[<*chan* >]:LABel <*label* >	Sets the bit pattern (label) specified on the four least significant bits of the reading.

Table 4-1. HP E1429A/B Command Quick Reference (Cont'd).

Subsystem	Commands	Description
DIAGnostic (cont'd)	DIAGnostic:FETCh[<chan >]? <*start_addr* >, <*count*>	Returns count number of readings beginning at start_addr.
	DIAGnostic:MEMory[<*chan* >]:FILL <*num_segments* > , <*count* >	Fill num_segments in memory with count readings.
	DIAGnostic:MEMory[<*chan* >]:ADDResses?	Returns the addresses of the next memory locations to be written to in each segment.
	DIAGnostic:PEEK? <*address* >, <*bits* >	Read the specified number of bits from the memory location specified.
	DIAGnostic:POKE <*address* >, <*bits* >, <*value* >	Write the value to the address specified.
	DIAGnostic:SGET? <*bit* >	Returns the state (0 or 1) of the specified bit in the serial control register.
	DIAGnostic:SPUT? <*bit* >, <*value* >	Sets the state (0 or 1) of the specified bit in the serial control register.
	DIAGnostic:TEST?	Returns information on a failed self-test.
FETCh	FETCh[<*chan* >]?	Returns readings from the specified channel.
	FETCh[<*chan* >]:COUNt?	Returns the number of readings stored in memory from the channel specified.
	FETCh[<*chan* >]:RECover?	Returns readings from the specified channel following a power-failure, digitizer configuration change, or reset.
FORMat	FORMat[:DATA] <*type* >[,<*length* >]	Specifies the output format for the measurement data.
INITiate	INITiate[:IMMediate]	Initiates the digitizer trigger system and places the digitizer in the wait-for-arm or wait-for-trigger state.
INPut	INPut[<*port* >]:FILTer[:LPASs][:STATe] <*mode* >	Enables/disables the 10 MHz input filter.
	IINPut[<*port* >]:IMPedance <*impedance* >	Sets the single ended port input impedance.
	INPut[<*port* >][:STATe] <*mode* >	Connects/disconnects the input ports from the signal path.
MEASure	MEASure[<*chan* >]:ARRay[:VOLTage][:DC]? (<*size*>) [,<*expected value* >[,<*resolution* >]] [,(@<*input por t*>)]	Configures the digitizer and takes <size> number of readings on the specified channel and input port.
MEMory	MEMory:BATTery:CHARge?	Checks the charge on the battery supporting non-volatile memory.
	MEMory:BATTery[:STATe] <*state* >	Enables/disables the battery supporting non-volatile memory.

Table 4-1. HP E1429A/B Command Quick Reference (Cont'd).

Subsystem	Commands	Description
OUTPut	OUTPut:ECLTrg<*n*>:FEED <*source* >	Specifies the source of the synchronization pulse routed to ECLTRG0 or ECLTRG1.
	OUTPut:ECLTrg<*n*>[:STATe] <*mode* >	Enables/disables the routing of the synchronization pulse.
	OUTPut:EXTernal[1]:FEED <*source* >	Specifies the source of the synchronization pulse routed to the "Ext 1" BNC port.
	OUTPut:EXTernal[1][:STATe] <mode>	Enables/disables the routing of the synchronization pulse.
	OUTPut:TTLTrg<*n* >:FEED <*source* >	Specifies the source of the synchronization pulse routed to a TTLTRG trigger line.
	OUTPut:TTLTrg<*n* >[:STATe] <*mode* >	Enables/disables the routing of the synchronization pulse.
READ	READ[<*chan* >]?	Returns readings from the specified channel.
SENSe	[SENSe[<*chan* >]]:FUNCtion <*function* >	Selects the channel's input port that will be used.
	[SENSe[<*chan* >]]:ROSCillator:EXTernal:FREQuency <*frequency* >	Indicates the frequency of the external reference source.
	[SENSe[<*chan* >]]:ROSCillator:SOURce <*source* >	Selects the reference frequency source.
	[SENSe[<*chan* >]]:SWEep:OFFSet:POINts <*count* >	Specifies the number of pre-arm readings.
	[SENSe[<*chan* >]]:SWEep:POINts <*count* >	Specifies the total number of (pre- and post-arm) readings taken during each arm cycle.
	[SENSe[<*chan* >]]:VOLTage[:DC]:RANGe <*range* >	Sets the digitizer measurement range.
	[SENSe[<*chan* >]]:VOLTage[:DC]:RESolution?	Queries the digitizer reading resolution.
STATus	STATus:OPC:INITiate <*state* >	Controls whether *OPC, *OPC?, and *WAI will complete immediately, or wait for the measurement or data transfer to complete.
	STATus:OPERation|:QUEStionable:CONDition?	Reads the settings of the condition register.
	STATus:OPERation|:QUEStionable:ENABle <*unmask* >	Specifies which bits in the event register are included in the summary bit.
	STATus:OPERation|:QUEStionable[:EVENt]?	Reads the settings of the event register.
	STATus:OPERation|:QUEStionable:NTRansition <*unmask* >	Sets the negative transition mask.
	STATus:OPERation|:QUEStionable:PTRansition <*unmask* >	Sets the positive transition mask.
	STATus:PRESet	Initializes the enable registers and transition masks, and sets STATus:OPC:INITiate ON.
SYSTem	SYSTem:ERRor?	Reads the error codes and messages in the digitizer error queue.
	SYSTem:VERsion?	Returns the SCPI conversion to which the digitizer complies.

Table 4-1. HP E1429A/B Command Quick Reference (Cont'd).

Subsystem	Commands	Description
TRIGger	ABORt	Removes the digitizer from the wait-for-trigger state.
	TRIGger[:STARt]:COUNt <*number* >	Specifies the total number of (pre- and post-arm) readings taken during each arm cycle.
	TRIGger[:STARt][:IMMediate]	Sends an immediate trigger to the digitizer. A reading is taken if the digitizer is in the wait-for-trigger state.
	TRIGger[:STARt]:SOURce <*source* >	Sets the trigger source.
	TRIGger[:STARt]:TIMer1 <*period* >	Sets the digitizer sample rate.
	TRIGger[:STARt]:TIMer2 <*period* >	Sets the post-arm sample rate for dual rate sampling.
VINStrument	VINStrument[:CONFigure]:LBUS:FEED <source>	Sets the Local bus data source.
	VINStrument[:CONFigure]:LBUS:MEMory:INITiate	Places the digitizer in the wait-for-trigger state and starts the data transfer from digitizer memory to the Local bus.
	VINStrument[:CONFigure]:LBUS[:MODE] <mode>	Sets the Local bus operating mode.
	VINStrument[:CONFigure]:LBUS:RESet	Resets the digitizer's (E1429B) Local bus chip.
	VINStrument[:CONFigure]:LBUS:SEND:POINts <count>	Sets the number of readings per Local bus transfer block.
	VINStrument[:CONFigure]:LBUS:SEND:POINts:AUTO <mode >	Sets the digitizers to the Local bus interleaved transfer mode.
	VINStrument[:CONFigure]:TEST:DATA <voltage_list>	Tests the Local bus by transmitting a list of data.
	VINStrument[:CONFigure]:VME:FEED <source>	Sets the VME (VXI data transfer) bus data source.
	VINStrument[:CONFigure]:VME:MEMory:INITiate	Places the digitizer in the wait-for-trigger state and starts the data transfer from digitizer memory to the VME (VXI data transfer) bus.
	VINStrument[:CONFigure]:VME[:MODE] <mode>	Sets the VME (VXI data transfer) bus mode.
	VINStrument[:CONFigure]:VME:SEND:ADDRess:DATA?	Returns the A24 offset address used to read measurement data from memory.
	VINStrument:IDENtity?	Indicates the virtual instrument capability of the digitizer.

SCPI Conformance Information

The HP E1429A/B 20 MSa/s 2-Channel Digitizer conforms to the SCPI-1992.0 standard.

The following tables list all the SCPI confirmed, approved, and non-SCPI commands that the HP E1429A/B can execute.

Table 4-2. SCPI Confirmed Commands.

```
ABORt

ARM
    [:START |:SEQuence[1]]
      :COUNt <count>
      :DELay <period>
      [:IMMediate]
      :LEVel[<chan>]
        :NEGative <voltage>
        :POSitive <voltage>
      :SLOPe[<n>] <edge>
      :SOURce[<n>] <source>

CALibration[<chan >]
    :DATA <block_data >
    :GAIN [<readings >[,<period>]]
    :VALue <number >
    :ZERO [<readings >
    [,<period>,[<mode >]]]

CONFigure[<chan >]
    :ARRay
        [:VOLTage]
          [:DC] (<size >)
[,expected value >[,<resolution >]]
[,(@ <input port >)]

FETCh[<chan>]?

FORMat
    [:DATA] <type > [,<length >]

INITiate
    [:IMMediate]

INPut[<port >]
    :FILTer
        [:LPASs]
          [:STATe] <mode >
    :IMPedance <impedance >
    [:STATe] <mode >

MEASure[<chan >]
    :ARRay
        [:VOLTage]
          [:DC]? (<size >)
[,expected value >[,<resolution >]]
[,(@ <input port >)]

OUTPut
    :ECLTrg<n >
        [:STATe] <mode >
    :TTLTrg<n >
        [:STATe] <mode>

READ[<chan >]?

[SENSe[chan]]
    :FUNCtion <function >
    :ROSCillator
      :EXTernal
          :FREQuency <frequency >
      :SOURce <source >
    :SWEep
      :OFFSet
          :POINts <count >
      :POINts <count >
    :VOLTage
      [:DC]
          :RANGe <range >
          :RESolution?

STATus
    :OPERation |:QUEStionable
        :CONDition?
        :ENABle <unmask>
        [:EVENt]?
        :NTRansition <unmask>
        :PTRansition <unmask>
    :PRESet

SYSTem
    :ERRor?
    :VERSion?

TRIGger
    [:STARt |:SEQuence[1]]
        :COUNt <number>
        [:IMMediate]
        :SOURce <source>
        :TIMer1 <period >
        :TIMer2 <period>
```

Table 4-3. Non-SCPI Commands.

```
CALibration[<chan >]
      :DELay
      :COUNt?
      :SECure
            :CODE <code >
            [:STATe] <mode >[,<code >]
      :STORe
            :AUTO <mode >

DIAGnostic
   :CALibration[<chan >]
      :CONVerge?
      :GAIN
         :SENSitivity?
      :ZERO
         :SENSitivity?
   :CHANnel[<chan >]
      :LABel <label >
   :FETCh[<chan >] <start_addr >,
            <count >
   :MEMory[<chan >]
      :FILL <num_segments >, <count >
      :ADDResses?
   :PEEK <address >, <bits >
   :POKE <address >, <bits >
   :SGET <bit >
   :SPUT <bit >, <value>
   :TEST?

FETCh[<chan >]
      :COUNt?
      :RECover?

MEMory
      :BATTery
            :CHARge?
            [:STATe] <state >

OUTPut
       :ECLTrg<n >
            :FEED  <source >
      :EXTernal[1]
            :FEED <source >
            [:STATe] <mode >
      :TTLTrg<n >
            :FEED  <source>

STATus
      :OPC
            :INITiate <state>

VINStrument
      [:CONFigure]
            :LBUS
               :FEED <source>
               :MEMory
                  :INITiate
               [:MODE] <mode >
               :RESet
               :SEND
                  :POINts <count >
                     :AUTO <mode >
            :TEST
               :DATA <voltage_list >
            :VME
               :FEED <source>
               :MEMory
                  :INITiate
               [:MODE] <mode >
               :SEND
                  :ADDRess
                     :DATA?
      :IDENtity?
```

Notes

Appendix A
Specifications

Appendix Contents

This appendix contains the HP E1429A/B 20 MSa/s 2-Channel Digitizer operating specifications. Except as noted, the specifications apply under the following conditions:

- **Period:** 1 year
- **Temperature:** 0° - 55° C
- **Relative humidity:** ≤ 65% @ 0° - 40° C
- **Warm up time:** 1 hour

References to the Local Bus Interface apply ONLY to E1429B; otherwise, E1429A and E1429B are identical except as noted. Characteristics given as "typical", "nominal", or "supplemental" are non-warranted; they provide additional information for application assistance.

NOTE (4.1.2.1) and similar notation refer to sections of "IEEE Standard 1057: Trial-Use Standard for Digitizing Waveform Recorders", published July 21, 1989. This document prescribes standard measurement procedures for several performance characteristics.

NOTE "Full-scale" refers to the entire two-sided range (+ and -) of the A/D, NOT to the one-sided interpretation customarily used by HP. The usage here conforms to that defined by IEEE 1057.

Memory Characteristics

Architecture

Equal amounts of memory are dedicated to channel 1 and channel 2. During measurements, both channels sample simultaneously and send data to their respective portions of memory. Memory is not dual-ported, and may not be read while it is being filled. However, data being routed to the memory may simultaneously be routed to the Local Bus or the VME Bus, according to their respective speed capabilities.

Read-Out

To VME Bus

After measurement completion, memory read-out may be requested for either channel separately or for both channels interleaved. Memory access is via repeated reading from a single VME register address; 16- or 32-bit accesses are permitted. This is the "channel I/O" model; i.e. memory can not be mapped or shared as can VME memory. Speed to VME Bus should be up to 2M transfers/s (16- or 32-bit transfers, 2M readings/s or 4M readings/s respectively); however, rates this high have not been tested with existing controllers.

To Local Bus (E1429B only)

After measurement completion, memory read-out may be requested for either channel separately or for both channels interleaved. (Data may also be routed to Local Bus in real-time, while digitizing is occurring.) Speed is up to 40 MBytes/s (20 Mreadings/s) for either channel separately, or 80 MBytes/s (40 Mreadings/s) for both channels interleaved.

Memory Size 524,288 readings per channel (512K, 1K=1024)

Nonvolatile memory A battery and associated support functions are provided to make the memory non-volatile for 4 years (nominal). Battery life can be extended to 7 years (nominal) shelf life by electronically disconnecting it. A register indicates low battery.

Partitions (Segments) When pre-arm readings are to be taken, and ARM:COUNt is from 1 to 128, memory is partitioned. This permits each partition to be used as a circular buffer, without disturbing data already recorded in other partitions. Each partition records data (both pre-arm and post-arm data) from one arm event.

Number of Partitions

1, 2, 4, 8, 16, 32, 64, or 128

Partitions are always equal-sized. The number of partitions is sufficient for the value programmed for ARM:COUNt. For example, an ARM:COUNt value of 5,6,7, or 8 will cause 8 partitions to be created.

Total Readings per Partition

(total memory size)/(number of partitions)

When the non-volatile mode of memory is enabled, this number is reduced by four readings.

Pre-Arm Data before the arm event can be stored in a circular-overwrite mode (per partition) until the event occurs.

Amplitude Characteristics and Signal Conditioning

A/D Converter 50Ω input, -1.0225V to 1.0230V range

Each channel contains a high-performance 12-bit, 20MSa/s A/D converter. For best A/D performance (highest linearity and lowest noise), choose the 50Ω input port and select its ± 1 volt input range. Except as noted, this port is used for A/D performance characteristics listed below. Due to amplitude and frequency limitations of available test sources, and other practical considerations, some parameters are specified only for this port and range.

Resolution

12 bits (including sign)

Codes from -2048 to +2047 indicate results as follows:

-2048 : amplifier overload (single-ended or differential inputs)
-2047 : not used
-2046 : normal-mode overload (negative)
-2045 : minimum on-scale reading
.
.
+2046 : maximum on-scale reading
+2047 : normal-mode overload (positive)

Output Formats

ASCII (9 significant digits) or REAL 64 (IEEE 64-bit binary) formats represent input voltage in volts, scaled appropriately according to voltage range setting used.

PACKED denotes 2's complement binary integers, with the raw A/D code (including sign) occupying the leftmost 12 bits of a 16-bit word, padded with four zero bits on the right.

Any of the above formats can be returned under the Word Serial protocol. The Packed format is also the format returned by direct VME register read operations and transmitted onto the Local Bus.

Gain and Offset (4.3.1, note 1057-1)

nominal gain : 2000 codes per volt
nominal offset : zero

Filtering

2-pole Bessel (10 MHz nominal) or none. For no filter, the analog bandwidth depends on the input port used (see below).

Effective Bits (4.5.2; 4.1.3) sampling at 20 MSa/s

Input signal 500 kHz : 10.0 (10.3 typical)
10 MHz : 9.5 (9.8 typical)

Harmonic Distortion (4.4.2.1)

Sample rate 20 MSa/s
-64 dB THD at 500 kHz input
-61 dB THD at 10 MHz input
(THD includes 2nd through 6th harmonics)

Signal-to-Noise Ratio (4.5.1)

62 dB (500 kHz)
59 dB (10 MHz)

("Noise" includes noise, distortion, and all other undesired effects, as defined in IEEE 1057.)

Differential Nonlinearity (4.4.1.2, 3680 Hz sine wave, codes -2045 to +2046)

1 LSB (no missing codes)

A/D Converter Supplemental Characteristics

Integral Nonlinearity (4.4.3, Note 1057-1)

2 LSB

Maximum Static Error (4.4.4.1, Note 1057-1)

2%

Word Error Rate (4.15)

qualified error level word error rate

>16 LSB	<2.5E-7
>32 LSB	<6E-8
>64 LSB	<5E-9

Single-ended inputs

Connector : BNC

Coupling : DC

Impedance : 50Ω or 75Ω ±0.5% (nominal) selection is programmable

Disconnect : via internal relay, impedance -> high

Ranges :- 0.10225V to 0.10230V
- 0.2045V to 0.2046V
- 0.51125V to 0.5115V
-1.0225V to 1.0230V

Overload : Flagged on-the-fly as ±FS in binary data. Input impedance remains nominally constant up to ±5V transient and continuous. Input voltages substantially exceeding this level cause a protection relay to trip, which resets itself when the overload is removed. Never exceed ±42 Vpk.

DC Accuracy : ±0.4% of reading ±0.25% of peak-to-peak full-scale

Accuracy is specified for the average of 100 readings (with CAL:ZERO performed within 24 hours prior to reading in a stable environment).

For temperatures outside 18-28 degrees C, add the following temperature coefficients for each degree below 18 C or above 28 C:

Range	% of peak-to-peak full-scale, per degree
0.1	0.055
0.2	0.035
others	0.025

Single-ended inputs, supplemental characteristics

Analog Bandwidth (4.6.1) (filter off)

>50 MHz (1V range)
>40 MHz (other ranges)

Effective bits on different ranges

(For performance on 1V single-ended range, see the previous A/D section.)

Typical effective bits relative to 1 v range:

Range	500 kHz	10 MHz
0.1	-0.4	0
0.2	-0.2	0
0.5	0	0

Crosstalk between channels (4.11)

Relative to full-scale input, DC - 10 MHz: -80 dB

Differential Inputs

Connectors: Two BNCs, one with positive (+) gain to A/D, one with negative (-) gain to A/D. The BNCs' outer shells (shield), though not grounded at the front panel, ARE connected internally to ground; they are NOT floating.

Coupling : DC

Impedance : 1 MΩ in parallel with 25 pF (nominal)

Ranges : A/D responds to the difference of the two input voltages on the(+) and (-) connectors.

-0.10225V	to	0.10230V
-0.2045V	to	0.2046V
-0.51125V	to	0.5115V
-1.0225V	to	1.0230V
-2.045V	to	2.046V
-5.1125V	to	5.115V
-10.225V	to	10.230V
-20.45V	to	20.46V
-51.125V	to	51.15V
-102.25V	to	102.30V

Overload: On-the-fly flagging includes amplifier overload. Ranges 0.1V to 5V: input impedance remains nominally constant up to ±10V.After removal of ±20 Vpk input, recovery is typically to within 1% of peak-to-peak full-scale in 250 µsec. Ranges 10V to 100V: input impedance remains nominally constant upto ±102.3V. After removal of ±100Vpk input, recovery is typically to within 1 % of peak-to-peak full-scale in 30 µsec. Never exceed ±102.3V input.

CMRR : Ranges 0.1023V to 5.115V:

for |Vcm|<= 10 volts peak and slew rate <150 v/µs:

DC: >68 dB
AC: >60 dB (1 MHz)

Ranges 10.23V to 102.3V:

for |Vcm|<=102.3 volts peak and slew rate <1500 v/µs:

DC: >45 dB
AC: >40 dB

CMRR is measured by applying a signal from a 50Ω through a "Tee" connector to both inputs.

DC Accuracy:

±0.5% of reading ±1% of peak-to-peak full-scale

Accuracy is specified for the average of 100 readings with inputs terminated in <1 kΩ and CAL:ZERO performed within 24 hours prior to reading in a stable ambient. For temperatures outside18-28 degrees C, add the following temperature coefficients for eachdegree below 18 C or above 28 C:

Range	% of peak-to-peak full scale, per degree
0.1024	0.055
0.2048	0.035
others	0.025

Differential Inputs, supplemental characteristics

Analog Bandwidth (4.6.1) (filter off)

15 MHz on 0.1023V through 1.023V ranges, plus 10.23V range
10 MHz on 2.046V and 20.46V ranges
4 MHz on 5.115V and 51.15V ranges
2 MHz on 102.3V range

Effective bits, relative to 1V single-ended range

(For 1 volt single-ended performance, see the previous A/D section.)
Typical effective bits relative to 1V single-ended:

Range	500 kHz	10 MHz
0.1	-1.4	-0.9
0.2	-0.7	-0.3
0.5	-0.2	-0.1
1.0	-0.1	-0.3

Crosstalk (4.11)

On 1V range, relative to full-scale input:

DC - 1 MHz	-75 dB
1 - 10 MHz	-70 dB

Frequency and Sample Rate Characteristics

Both channels always sample simultaneously at the indicated rates and times.

Tolerances: All internally-generated frequencies and rates are ±0.0075% initial tolerance.

Internal Timer

The Internal Timer generates time intervals useful in controlling the sample rate. It divides a reference frequency by $1x10^n$, $2x10^n$, or $4x10^n$, n = 0 to 8. When the reference frequency is the internal 20 MHz oscillator, the resulting rates are from 0.05 Sa/s to 20 MSa/s in a 1,2,5 sequence. Other programmable choices for the reference source are VXI CLK10, the ECL Trigger lines, and the Ext2 BNC.

Trigger (Sample Clock) Subsystem

Each event in this subsystem causes one A/D conversion in both channels. (In Standard Commands for Programmable Instruments (SCPI), this is the meaning of "Trigger". The SCPI term for the commencement of a series of one or more triggers is "Arm".)

Rate: (Internal Timer using built-in reference oscillator)
0.05 Sa/s to 20 MSa/s in 5,1,2 sequence

External: VXI Trigger Busses (TTL and ECL), External BNCs, software, or VME Read cycle

Post-Count: 1, or 7 to 2^{24-1}, or continuous. (This specifies the desired number of sample triggers after the Arm event.)
Additional limitations apply when the "Pre-Count"feature is used; see below.

Pre-Count: 0, or 3 to 65535. When this mode is used (pre-count0) the Digitizer samples continuously until the pre-count is satisfied and then an Arm event occurs. Memory is used as a circular buffer with older readings overwritten by newer readings. The programmed pre-count value specifies the number of pre-arm samples to be protected from overwriting by post-arm sample data. This mode limits the number of Post-Count read-ings: The total of (pre-count + post-count) must be no larger than the memory partition size.

Dual-Rate: The Pre-Count readings can be measured at one sample rate; then the Post-Count readings can be measured at another sample rate. The possible sample rate sources occur in pairs. There are four possible pairs:

{ EXT1 BNC (pre), EXT2 BNC (post); or vice-versa }
{ ECLTRG0 (pre), ECLTRG1 (post); or vice-versa }
{ REFERENCE (pre), REFERENCE/N (post)}
{ REFERENCE/N (pre), REFERENCE (post)}
(N=2,4,10,20,40...4E8).

Timebase and Trigger additional supplemental characteristics

Fixed error in sample time (4.9.1)

40 psec (record size 32K)

Arm Subsystem

Each event in this subsystem allows acquisition of one waveform record (i.e., a burst of one or more dual-channel A/D conversions). (Note that in SCPI, each A/D conversion event is a "Trigger".)

Sources: VXI Trigger Busses (TTL and ECL), External BNC, plus input channel voltage (*). The logical "OR" condition of any two of these sources may be used.

(*) Each channel's detection circuitry uses two programmed voltage levels, so that it is possible to generate the arm event when the signal either enters or leaves a defined voltage window. Voltage level set points typically are accurate to 3% of full-scale and have hysteresis 0.5% of full-scale.

Rate: when not taking Pre-[Arm]count readings: up to 2M/sec.
when taking Pre-[Arm]count readings: 650 µsec typical

Count: 1 to 65535 or continuous (no pre-arm readings)
1 to 128 (with pre-arm readings)

Delay: Specifies additional programmed time delay from an arm's causative event to when the arming actually occurs. This is in addition to irreducible internal delays.

Let the Reference period be T. Added delay can be:

0T to 65534T in steps of T
65540T to 655350T in steps of 10T

Bus Access and Connectors

Front Panel Connectors (BNC) :

Channel 1 Inputs : 50Ω/75Ω single-ended
1 MΩ (+)
1 MΩ (-)

Channel 2 Inputs : as for channel 1

Ext 1 : In : Arm, Trigger
Out: Arm, Trigger, Reference, Ready for Trigger, Pre-Arm Count Complete

Ext 2 : In : Trigger, Reference

VXI ECLTrig: In: Arm, Trigger, Reference
Out: Arm, Trigger, Reference, Ext 1 BNC signal

VXI TTLTrig: In: Arm, Trigger
Out: Arm, Ready

VME Bus: The HP E1429A can be used as either a VXI Message-Based or register-based instrument.

Message-Based operation: Uses Standard Commands for Programmable Instruments (SCPI). This provides easy operation with an industry-standard programming interface.

Register-Based operation: Registers controlling the hardware are directly accessible, providing the highest possible throughput, at the cost of programming effort. Documentation provided includes descriptive material and "C" source code. Some functions, such as calibration, can only be performed with the assistance of the on-board 68000 processor.

P2 Local Bus : (E1429B only)

Modes per HP Virtual Instrument Protocol:

Append
Generate
Off
Pipeline

Data Source

Data can be sourced directly from the A/Ds or from memory -- either channel or both channels interleaved.

Pacing

Real-time (during measurements) paced by arm/trigger system
From memory (after measurements) paced by receiving module on local bus

Maximum speed: 80 MByte/sec (i.e., can do full speed on both channels)

General Characteristics

Size :	C
Slots :	1
Connectors :	P1, P2
Weight (kg) :	1.9
Device Type :	Message-Based Servant
VXIBus Revision Compliance :	1.4
Register Level Documentation :	Subset
SCPI Revision :	1992.0
Manufacturer Code :	4095 Decimal
Model Code :	448 Decimal
Slave :	A16/A24 D08/D16/D32

Currents in Amps (typical)		E1429A	E1429B
+5V	I(pm)	2.9	3.1
	I(dm)	0.5	0.5
+12V	I(pm)	0.2	0.2
	I(dm)	0.04	0.04
- 12V	I(pm)	0.2	0.2
	I(dm)	0.04	0.04
+24V	I(pm)	0.1	0.1
	I(dm)	0.05	0.05
-24V	I(pm)	0.1	0.1
	I(dm)	0.05	0.05
-5.2V	I(pm)	3.6	4.1
	I(dm)	0.36	0.36
-2V	I(pm)	1.2	1.3
	I(dm)	0.12	0.12
+5VS	I(pm)	0	0
	I(dm)	0	0

Typical Watts/Slot: 41.5 45.3
dPressure (H2O): 0.8 mm
AirFlow (liters/s):3.8
EMC: To meet EMC requirements in Europe, a backplane connector shield kit is included.

Built-In Test : Extensive built-in test checks memory, timebase, much of the trigger system, and part of the analog signal path.

Notes pertaining to IEEE 1057:

1057-1:

Based on Code Transition Levels per 4.1.2 except that levels -2047 and -2046 are not included in this characterization. The minimum code transition level characterized is (from -2046 to) -2045; the maximum is (from +2046 to) +2047.

Notes

Appendix B
Useful Tables

Appendix Contents

The tables in this appendix contain information often referred to during HP E1429A/B programming. The tables in this appendix include:

Table B-1. HP E1429A/B Example Program Listing

Location	Program Name	Language	Description
Chapter 1	IDN.C	HP BASIC, C	Program to test communication between the PC and the digitizer.
	SLFTST.C	"	E1429A/B Self Test.
	RSTCLS.C	"	Resetting and clearing the digitizer.
	LRN.C	"	Power-on/reset configuration.
	MEAS.C	"	Making a measurement with the digitizer.
	CONF.C	"	Configuring the digitizer.
	QUERY.C	"	Queries SCPI command settings.
	ERRORCHK.C	"	Error checking program.
Chapter 2	INPUT.C	C	Configures the digitizer input.
	ARMCNT.C	"	Takes a burst of readings.
	ARMLEVEL.C	"	Arm on a specified input signal level.
	PREPOST.C	"	Taking pre- and post-arm readings.
	SAMPLE.C	"	Specifying a sample rate.
	DUALSAMP.C	"	Pre- and post-arm dual rate sampling.
	MULT_AD.C	"	Uses multiple digitizers.
	PACKED.C	"	Uses the packed data format.
	VME_REAL.C	"	VME bus data transfers.
	VME_SEG1.C	"	Transfers segmented readings.
	SEGTST16.CPP	C++	VME bus data (16-bit) transfers using embedded controller.
	SEGTST32.CPP	"	VME bus data (32-bit) transfers using embedded controller.
	INST.H	"	Used with SEGTST.16 and SEGTST.32.
	INST.CPP	"	Used with SEGTST.16 and SEGTST.32.
	E1429.H	"	Used with SEGTST.16 and SEGTST.32.
	E1429.CPP	"	Used with SEGTST.16 and SEGTST.32.

Table B-1. HP E1429A/B Example Program Listing (Cont'd)

Location	Program Name	**Language**	**Description**
Chapter 2 (Cont'd)	LOCAL_AD.C	C	Local bus data transfer using a single digitizer.
	LBUS2PST.C	"	Local bus data transfer from digitizer memory using multiple digitizers.
	LBUSAUTO.C	"	Local bus data transfer from the digitizer A/D using multiple digitizers.
	STATUS.C	"	Demonstrates the use of the digitizer status registers.
Appendix C	REG_PROG.C	C	Sets the measurement range, trigger source, sample rate, reading count, and re-initiates the digitizer using register reads and writes.
Appendix D	LBUSINTR.C	C	Transfers data using the Local bus interleaved transfer mode.

Table B-2. HP E1429A/B Power-On/Reset Configuration (returned by *LRN?)

Parameter	Command	Power-on/Reset Setting
Macro useage	*EMC	+0
Automatic cal constant storage	CAL:STOR:AUTO	1 (enabled)
Channel 1 calibration value	CAL1:VAL	+1.01850000E+000
Channel 2 calibration value	CAL2:VAL	+1.01850000E+000
Reading format	:FORM	ASC,+9
Channel 1 S/E input filter	:INP1:FILT	0 (disabled)
Channel 1 S/E input impedance	IMP	+5.00000000E+001
Channel 1 S/E input state	STAT	1 (enabled)
Channel 2 S/E input filter	:INP2:FILT	0 (disabled)
Channel 2 S/E input impedance	IMP	+5.00000000E+001
Channel 2 S/E input state	STAT	1 (enabled)
Channel 1 differential input filter	:INP3:FILT	0 (disabled)
Channel 1 differential input state	STAT	1 (enabled)
Channel 2 differential input filter	:INP4:FILT	0 (disabled)
Channel 2 differential input state	STAT	1 (enabled)
Input port select (channel 1)	:FUNC	"VOLT1"
Measurement range (channel 1)	:VOLT:RANG	+1.02350000E+000
Input port select (channel 2)	:SENS2:FUNC	"VOLT2"
Measurement range (channel 1)	:SENS2:VOLT:RANG	+1.02350000E+000
Reference oscillator freqency	:ROSC:EXT:FREQ	+2.00000000E+007
Reference oscillator source	:ROSC:SOUR	INTernal
VME bus data source	:VINS:VME:FEED	"MEM:BOTH32"
VME bus data transfer mode	MODE	OFF
Local bus data source	:VINS:LBUS:FEED	"MEM:BOTH"
Local bus data transfer mode	MODE	OFF
Local bus readings per block	:VINS:LBUS:SEND:POIN	+2.00000000E+000
Interleaved transfer mode	POIN:AUTO	ON

Table B-2. HP E1429A/B Power-On/Reset Configuration (Cont'd)

Parameter	Command	Power-on/Reset Setting
Arm count	:ARM:COUN	+1.00000000E+000
Arm delay	DEL	+0.00000000E+000
Arm source1	SOUR	IMMediate
Arm slope1	SLOP1	POSitive
Arm level1 (negative)	:ARM:LEV:NEG	-1.02241848E+000
Arm level1 (positive)	POS	+1.02241848E+000
Arm source2	:ARM:SOUR2	HOLD
Arm slope2	SLOP2	POSitive
Arm level2 (negative)	:ARM:LEV2:NEG	-1.02241848E+000
Arm level2 (positive)	POS	+1.02241848E+000
Trigger source	:TRIG:SOUR	TIMer
Sample rate1	TIM1	+5.00000000E-008
Sample rate2	TIM2	+1.00000000E-007
Reading count	:SWE:POIN	+1.000000000E+000
Pre-arm reading count	OFFS:POIN	+0
ECLTrg0 synchronization pulse source	:OUTP:ECLT0:FEED	"TRIG"
ECLTrg0 synchronization state	STAT	0 (disabled)
ECLTrg1 synchronization pulse source	:OUTP:ECLT1:FEED	"EXT"
ECLTrg0 synchronization state	STAT	0 (disabled)
External 1 BNC synchronization source	:OUTP:EXT:FEED	"TRIG"
External 1 BNC synchronization state	STAT	0 (disabled)
TTLTrg0 synchronization pulse source	:OUTP:TTLT0:FEED	"ARM"
TTLTrg0 synchronization state	STAT	0 (disabled)
TTLTrg1 - TTLTrg7 synchronization states	STAT	0 (disabled)

Table B-3. HP E1429A/B Error Messages

Code	Message	Description
-101	Invalid character	Unrecognized character in parameter.
-102	Syntax error	Command is missing a space or comma between parameters.
-103	Invalid separator	Parameter is separated by a character other than a comma.
-104	Data type error	The wrong data type (number, character, string, expression) was used when specifying the parameter.
-105	GET not allowed	An HP-IB Group Execute Trigger was included in a command string sent to the digitizer.
-108	Parameter not allowed	More parameters were received than expected for the command header.
-109	Missing parameter	Command requires a parameter or parameters.
-112	Program mnemonic too long	Command keyword ≥12 characters
-113	Undefined header	Command header (keyword) was incorrectly specified.
-121	Invalid character in number	A character other than a comma or number is in the middle of a number.
-123	Exponent too large	The magnitude of the exponent was larger than 32000.
-124	Too many digits	More than 255 digits were used to specify a number.
-128	Numeric data not allowed	A number was specified when a letter was required.
-131	Invalid suffix	Parameter suffix incorrectly specified (e.g. 10 MZ rather than 10 MHZ).
-138	Suffix not allowed	Parameter suffix is specifed when one is not allowed.
-141	Invalid character data	Discrete parameter specified is not a valid choice.
-144	Character data too long	A character data type parameter is ≥12 characters.
-148	Character data not allowed	Discrete parameter was specified when another type (e.g. numeric, boolean) is required.
-151	Invalid string data	The string data specified (such as for the OUTPut:ECLTrg:FEED <source>command) is not a valid choice.
-158	String data not allowed	A string was specified when another parameter type (i.e. discrete, numeric, boolean) is required.

Table B-3. HP E1429A/B Error Messages (Cont'd)

Code	Message	Description
-161	Invalid block data	The number of bytes in a definite length data block does not equal the number of bytes indicated by the block header.
-168	Block data not allowed	Block data was specified when another parameter type (i.e. discrete, numeric, boolean) is required.
171	Invalid Expression	The expression used to calculate a parameter value is invalid.
-178	Expression data not allowed	An expression cannot be used to calculate a parameter value .
181	Invalid outside macro definition	A macro parameter placeholder ($<number) was encountered outside of a macro definition.
-183	Invalid inside macro definition	A command was encountered that is not allowed inside a macro.
184	Macro parameter error	A command inside the macro definition had the wrong number or wrong type of parameters.
-211	Trigger ignored	A trigger was received and the digitizer was not in the wait-for-trigger state. Or, a trigger was received from a source other than the specified source.
-212	Arm ignored	An arm was received and the digitizer was not in the wait-for-arm state. Or, an arm was received from a source other than the specified source.
-213	Init ignored	INITiate:IMMediate received while the digitizer was initiated.
-214	Trigger deadlock	Readings cannot be retrieved using FETCh? or READ? because TRIGer:STARt:COUNt INFinite is set. Also occurs with READ? and TRIGger:STARt:SOURce HOLD or TRIGger:STARt:SOURce BUS set.
-215	Arm deadlock	Readings cannot be retrieved using FETCh? or READ? because ARM:STARt:COUNt INFinite is set. Also occurs with READ? and ARM:STARt:SOURce HOLD, ARM:STARt:SOURce BUS, or ARM:STARt:SOURce OFF set.
-221	Settings conflict	Refer to the statement appended to the "Settings conflict" message for a description of the conflict and how it was resolved.
-222	Data out of range	Parameter value is out of range for any digitizer configuration.
-224	Illegal parameter value	An exact value, from a list of possible choices, was expected.

Table B-3. HP E1429A/B Error Messages (Cont'd)

Code	Message	Description
-230	Data corrupt or stale	Attempting to FETCh? data from the digitizer following a reset or other digitizer configuration change.
-231	Data questionable	Reading accuracy is questionable. An example is when the expected value and resolution parameters of the CONFigure or MEASure command are specified. If the resolution is too fine for the expected value, this error occurs.
-240	Hardware error	The command could not be executed because of a hardware failure.
-270	Macro error	*RMC <name>was executed and name is not defined.
-271	Macro syntax error	A syntax error occurred among the commands within the macro.
-272	Macro execution error	Macro program data sequence could not be executed due to a syntax error within the macro definition.
-273	Illegal macro label	The macro label defined in the *DMC command was too long, the same as a common command keyword, or contained invalid header syntax.
-274	Macro parameter error	The macro definition improperly used a macro parameter placeholder.
-275	Macro definition too long	The commands within the macro could not be executed because the string or block contents were too long.
-276	Macro recursion error	A macro program data sequence could not be executed because the sequence leads to the execution of a macro being defined.
-277	Macro redefinition not allowed	A macro label in the *DMC command could not be executed because the macro label was already defined.
-278	Macro header not found	A legal macro label in the *GMC? query could not be executed because the header was not previously defined.
-312	PUD memory lost	The protected user data saved by the *PUD command has been lost.
-313	Calibration memory lost	The nonvolatile calibration data used by the *CAL command has been lost.
-330	Self-test failed	Note the information associated with the message for a description of the failure.
-350	Queue overflow	The digitizer error queue is full and additional errors have occurred.

Table B-3. HP E1429A/B Error Messages (Cont'd)

Code	Message	Description
-410	Query INTERRUPTED	The digitizer was sent a command before it was finished responding to a query command.
-420	Query UNTERMINATED	The controller (computer) attempts to read a query response from the digitizer without having first sent a complete query command.
-430	Query DEADLOCKED	The digitizer's input and output buffers are full and the digitizer cannot continue.
-440	Query UNTERMINATED after indefinite response	Occurs when the *IDN? query is not the last query executed in a command string.
1002	Cal security enabled	Calibration security must be disabled to calibrate the digitizer, to read or write calibration data, to change the security code, or to change the protected user data.
1004	Cal write fail	Writing calibration or protected user data (*PUD) to nonvolatile memory failed.
1005	Error during CAL	An error occurred during calibration. Refer to the statement appended to this message for a description of the error.
1007	Calibration security defeated	A jumper was moved to defeat calibration security.
1008	Error during zero cal	An error occurred during calibration of the zero offset. Refer to the statement appended to this message for a description of the error.
1009	Error during gain cal	An error occurred during gain calibration. Refer to the statement appended to this message for a description of the error.
1010	Error during linearity cal	An error occurred during linearity calibration. Refer to the statement appended to this message for a description of the error.
1015	A/D control register not responding	The serial interface register was not working properly at power-on.
1016	Illegal during LBUS or VME memory transfer	The command can not be executed while a VME bus or Local bus data transfer is in progress.
1017	Battery too low, data may be lost	The battery does not contain sufficient charge to maintain memory over an extended period. This error occurs when readings are taken, when the battery is enabled/disabled, or during the self-test.
1018	Battery backed data corrupt	This error is due to a low battery charge, or if the battery is enabled after readings are in memory.

Table B-3. HP E1429A/B Error Messages (Cont'd)

Code	Message	Description
1019	Data loss detected during LBUS transfer	Readings from the digitizer A/D were lost during a Local bus transfer. This error usually occurs when multiple digitizers are used and arming/triggering signals between them cause readings to be missed.
1020	Indefinite block not allowed	When executing the VINStrument:TEST:DATA command, the data must be in the IEEE-488.2 definite length block format.
1021	LBUS still running, ABOR or VINS:LBUS:RES needed	The HP E1429B digitizer is in the interleaved transfer mode and the LBUS chip is still active. The chip must be reset before the next INITiate command.
1022	Local bus test data size not multiple of 4 bytes	When executing the VINStrument:TEST:DATA command, the amount of data sent must be a multiple of 4 (bytes).
1213	Illegal when initiated	The command can not be executed while the digitizer is INITiated.
2003	Memory address incorrect	Address specified by DIAGnostic:POKE or DIAGnostic:PEEK? is not valid.
2004	Invalid address for 32-bit access	Attemping a 32-bit read from an odd numbered address.
2007	Bus error	Error during DIAGnostic:PEEK or DIAGnostic:POKE?

n

Appendix C
Register Programming

Appendix Contents

The HP E1429A/B 20 MSa/s 2-Channel Digitizer is a message-based device. As such, it supports the VXI word-serial protocol used to transfer ASCII command strings and is capable of converting the SCPI commands it receives to reads and writes of its hardware registers.

Register-based programming allows direct access to the hardware registers. This increases the speed at which events in the digitizer occur since the parsing (converting to register reads and writes) of SCPI commands is eliminated.

This appendix **is not** a 1-to-1 correlation between each digitizer SCPI command and an equivalent register read or write. However, basic digitizer configuration is covered, together with methods of re-initiating the digitizer and retrieving data from memory through direct register access. The sections of the appendix include:

System Configuration

The example programs and programming techniques shown in this appendix are based on the following system configuration:

Controller:	HP Vectra 386/25 personal computer
HP-IB Interface Card:	HP 82335 HP-IB Interface with Command Library
Mainframe:	HP 75000 Series C
Slot0/Resource Manager:	HP E1406 Command Module
HP E1429A/B Logical Address:	40

Each program uses a combination of SCPI commands and register reads/writes. In most cases SCPI commands set the initial digitizer configuration. Register reads/writes are used to modify the configuration, re-initialize the digitizer, and retrieve readings.

Reading and Writing to the Registers

The examples in this appendix are based on the system configuration listed previously. With this configuration, the digitizer's A24 registers are read, and written to, using the **HP E1406 Command Module's** DIAGnostic:PEEK?, DIAGnostic:POKE, and DIAGnostic:UPLoad:SADDress? commands:

DIAGnostic:PEEK? <*address* >, <*width* >

DIAGnostic:POKE <*address* >, <*width* >, <*data* >

DIAGnostic:UPLoad:SADDress? <*address* >, <*byte_count* >

<*address* > - the address (A24 base address + register offset) of the register.

<*width* > - the number of bits read (DIAG:PEEK?), or the number of data bits written to the register (DIAG:POKE). **Unless otherwise noted, register reads and writes are 8-bits.**

<*data* >- the integer data written to the register.

<*byte_count* >- the number of reading bytes uploaded (read) from digitizer memory. Since each reading is two bytes, *byte_count* is equal to 2 times the number of readings to upload. In the example program at the end of this appendix, DIAGnostic:UPLoad:SADDress? is used to retrieve all the readings from memory once the digitizer has been re-initiated.

Note With an embedded controller, the Standard Instrument Control Library (SICL), and the programming procedures found in this appendix, higher throughput rates can be achieved than the rates available with the system configuration listed. To use these procedures, note the A24 address mapping of the embedded controller, and replace DIAG:PEEK?, DIAG:POKE, and DIAG:UPL:SADD? with the appropriate SICL reads and writes.

Addressing the Registers

Access to the digitizer's registers is through addresses mapped into A24 address space. At power-on, the system resource manager (HP E1406) reads the digitizer's Device Type register (in A16 address space) to determine the amount of A24 memory the digitizer needs (4096 bytes). The resource manager allocates a block of A24 memory to the digitizer and writes the base (starting) address into the digitizer's Offset register (also in A16 space).

Figure C-1 is an example of how the HP E1406 resource manager maps the digitizer registers into A16 and A24 address space.

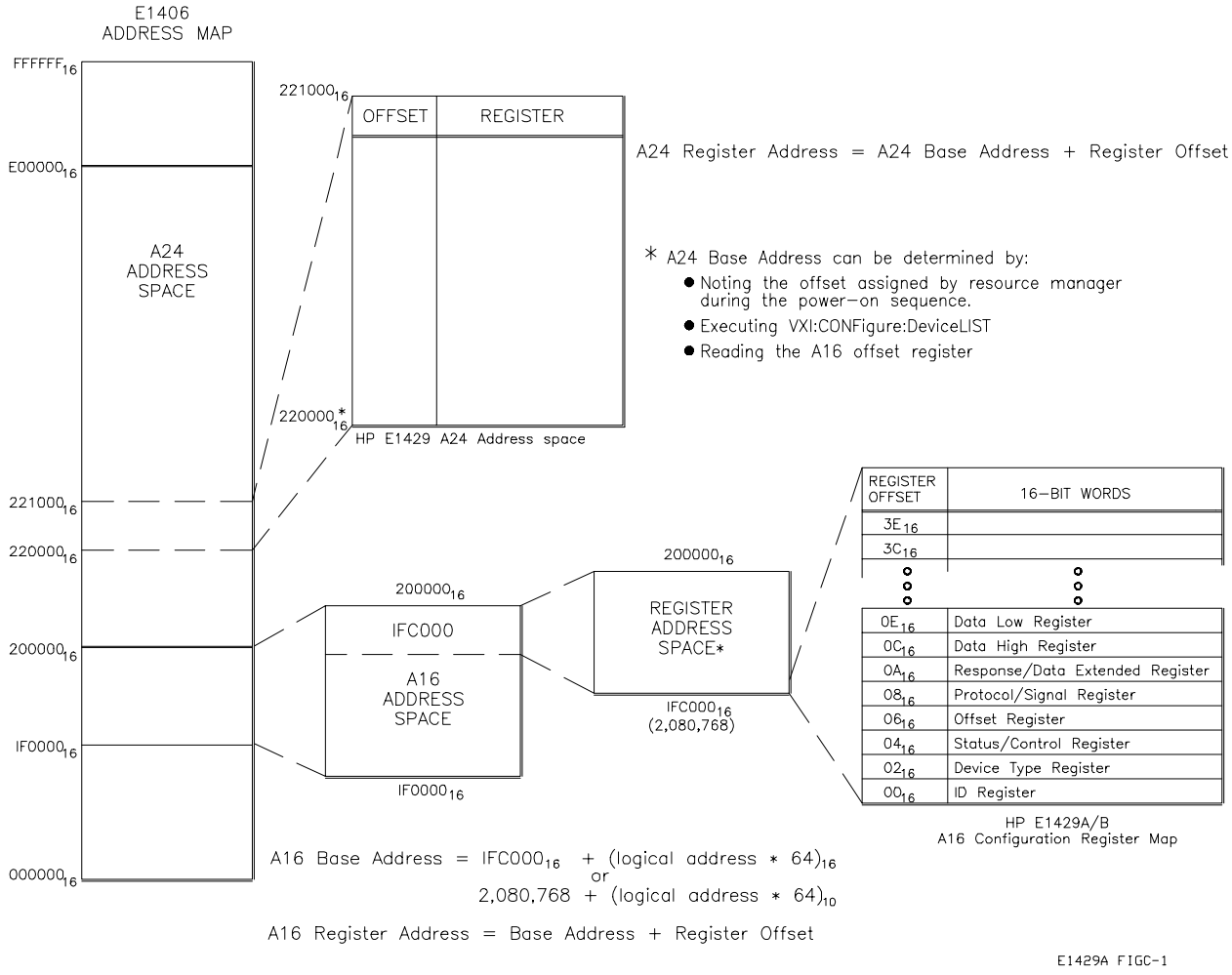


Figure C-1. HP E1429A/B A24 Address Space

Note The following information on determining register addresses is based on the computer configuration shown in Figure C-1, and on address mapping as performed by the HP E1406 Command Module's resource manager. For configurations with embedded controllers or configurations with a resource manager other than the HP E1406 Command Module, refer to those manuals containing information on A24 address mapping.

Determining the A24 Base Address

When you are reading or writing to a digitizer register, a hexadecimal or decimal register address is specified. An A24 register address is:

A24 register address = A24 base address + register offset

There are three ways to determine the digitizer's **A24 base address**:

1. Note the base address assigned by the resource manager at power-on. The HP E1406 resource manager configuration sequence can be monitored using an RS-232 terminal or printer. The "C-Size VXIbus Systems Installation and Getting Started Guide" contains information on connecting a terminal.
2. Execute the following HP E1406 Command Module command:

   VXI:CONFigure:DeviceLIST? <logical_address>

   The C language example programs disk contains the program Query.C. By changing the line:

   #define ADDR 70905L (E1429 digitizer address)

   to:

   #define ADDR 70900L (E1406 address)

   and entering the command:

   VXI:CONF:DLIS? 40(or the current E1429 logical address)

   a program string similar to the following is returned when the program executes:

   vxi:conf:dlis? 40 = +40,+0,+4095,+448,+1,+0,MSG,A24, **#H00220000**, #H00001000,Ready,"","","",MBinstr INSTALLED AT SECONDARY ADDR 5"

3. The hexadecimal number in **bold** is the digitizer's A24 base address.

4. Read the digitizer's offset register in A16 address space. As shown in Figure C-1, the Offset register is one of the digitizer's configuration registers.

   In a system where the HP E1406 Command Module allocates address space, the A16 base address of the configuration registers is computed as:

   $1FC000_{16} + (LADDR * 64)_{16}$

   2,080,768 + (LADDR * 64)

   where $1FC000_{16}$ is the starting location of the configuration register addresses, LADDR is the digitizer's logical address, and 64 is the number of address bytes in A16 per VXI device.

   The digitizer's factory set logical address is 40. If this address is not changed, the base address of the digitizer's configuration registers in A16 is:

   $1FC000_{16} + (40 * 64)_{16}$

   $1FC000_{16} + A00_{16} = 1FCA00_{16}$

   or decimal

   2,080,768 + (40 * 64)

   2,080,768 + 2560 = 2,083,328

   Given the A16 base address and the "offset" of the Offset register (06 from Figure C-1), the digitizer's A24 base address can be determined as shown in the program A24_REAL.C.

A24_READ.C

```
/* A24_READ.C - This program reads the digitizer's A24 base address. */

/* Include the following header files */

#include   <stdio.h>
#include   <cfunc.h>              /* This file is from the HP-IB Command Library */

#define CMD_MOD 70900L     /* I/O path between the digitizer and the Command Module */

/* Function prototypes */

long get_base_addr(void);

/*****************************************************************************/
void main(void)
{
long base_addr;/* variable for digitizer A24 base address */


base_addr = get_base_addr();          /* function call to calculate and */
                                      /* return digitizer A24 base address */

printf("\nA24 base address = %ld", base_addr);
}

/*****************************************************************************/
long get_base_addr(void)
{
/* base address of (A24) offset register in A16 address space */
long base_addr = (0x1FC000 + (40 * 64)) + 6;    /* digitizer logical address is 40 */

float a24offst;         /* A24 offset from A16 offset register */

char rd_addr[80];    /* command string variable */

/* Create the command string which reads the A24 base address from the offset register*/
sprintf(rd_addr, "DIAG:PEEK? %ld, %d", base_addr,16);

/* Send DIAG:PEEK? command */
IOOUTPUTS(CMD_MOD, rd_addr, strlen(rd_addr));

/* Read value from offset register */
IOENTER(CMD_MOD, &a24offst);
```

Continued on Next Page

```
/* Multiply offset value by 256 for 24-bit address value */
a24offst *= 256.;

return (long)a24offst;
}
```

Note Multiplying the value of the offset register (a24offst) by 256 (100_{16}) converts the 16-bit register value to a 24-bit address.

Register Descriptions

The registers used to configure the digitizer are shown on the following pages. The registers are listed by functional group. Listed with each register are its bit definitions and, where applicable, the bit settings at power-on or following a reset. Note that "base" is the A24 base address.

A24 Register Table

Category	Address	Read Registers	Write Registers	See Page...
Input Configuration Registers	base + 03_{16}	A/D Status Register		C-350
	base + 05_{16}	A/D Serial Read Register	A/D Serial Write Register	C-350
	base + $0B_{16}$		A/D Parallel Strobe Register	C-351
		A/D Shift Register		C-351
Arm and Trigger (timebase) Configuration Registers	base + 41_{16}	Arm Immediate Register	Abort Register	C-353
	base + 43_{16}	Arm Status Register		C-353
	base + 45_{16}	Sample Trigger Register	Timebase Initiation Register	C-354
	base + 47_{16}		Arm Trigger Register	C-355
	base + 49_{16}		Arm Source Register	C-355
	base + $4B_{16}$		Arm Control Register	C-356
	base +$4D_{16}$		Trigger Source Register	C-357
	base +$4F_{16}$		Reference Oscillator Register	C-359
	base +51_{16}		MSByte Arm Delay Register	C-360
	base + 53_{16}		LSByte Arm Delay Register	C-360
	base +55_{16}		MSByte Arm Count Register	C-360
	base +57_{16}		LSByte Arm Count Register	C-360
	base +$5D_{16}$		Trigger Immediate Register	C-361
	base + $5F_{16}$		Timebase Reset Register	C-380
	base +59_{16}		Arm Count Latch Register	C-360
	base +61_{16}		Decade Division Register	C-361
	base +63_{16}		Binary Division Register	C-361
	base + 65_{16}		Interpolator Control Register	C-380
	base +67_{16}		Stop Data Register	C-380
	base + 69_{16}		Interpolator Calibration Register	C-380
	base + $6B_{16}$		Self-test Register	C-380
	base +73_{16}		LSByte Pre-arm Count Register	C-362
	base +75_{16}		MSByte Pre-arm Count Register	C-362
	base +77_{16}		LSByte Post-arm Count Register	C-362
	base +79_{16}		MIDByte Post-arm Count Register	C-362
	base +$7B_{16}$		MSByte Post-arm Count Register	C-362
	base +$7D_{16}$		Timebase Initialization Register	C-380
	base +$7F_{16}$		Timebase Initialization Register	C-380

Category	Address	Read Registers	Write Registers	See Page...
Memory Control Registers	base + 02_{16}	Traffic Register		C-363
	base + 08_{16}	Pulse Register		C-364
	base + $0A_{16}$	Channel ID Register		C-364
	base + $0C_{16}$	Data Register		C-364
	base + 21_{16}	Memory Control Register		C-365
	base + 23_{16}	Memory address register 0		C-366
	base + 25_{16}	Memory address register 1		C-366
	base + 27_{16}	Memory address register 2		C-366
	base + $2B_{16}$	Terminal Address Register		C-367
	base + $2D_{16}$	Base Address 0 Register		C-367
	base + $2F_{16}$	Base Address 1 Register		C-367

The Input Configuration Registers

The input configuration registers are used to set the following digitizer input parameters:

- Input enable
- Input impedance
- Input filter
- Measurement range

The A/D Status Register — base +03_{16}

The A/D status register is a read only register that returns the status of the digitizer's input section. The register bits are defined below. Only the use of bit 0 is documented.

Address	7	6	5	4	3	2	1	0
base + 03_{16}	Ch. 2 Diff. ovld	Ch. 1 Diff. ovld	Ch2. S/E ovld	Ch1. S/E ovld	unused	Ovld clr	Error LED	Bit State
	0 - no ovld 1 - ovld	0 - no ovld 1 - ovld	0 - no ovld 1 - ovld	0 - no ovld 1 - ovld	---		0 - OFF 1 - ON	out data

The state ('1' or '0') of bit 0 represents the state of the A/D shift register's output bit (bit 55). A read of the status register (and the output bit) does not cause a shift of the A/D shift register.

The A/D Serial Register — base +05_{16}

The A/D serial register is a read/write register that receives and sends configuration data from/to the A/D shift register. The shift register is used to enable the inputs, and to set the input impedance, filter, and measurement range.

Address	7	6	5	4	3	2	1	0
base + 05_{16}	unused							register write: shifts one bit into the shift register register read: reads one bit out of the shift register

Each time a '1' or '0' is written to the serial register, one bit is loaded into the A/D shift register at bit position 0.

Each time the serial register is read, one bit is shifted out of the A/D shift register.

The A/D Parallel Strobe Register — base +0B$_{16}$

The A/D parallel strobe register is a write only register that latches the A/D configuration held by the A/D shift register to the analog-to-digital converter.

Address	7	6	5	4	3	2	1	0
base + 0B$_{16}$	not used					0 - idle 1 - strobe	not used	

Setting bit 2 to '1' latches the configuration represented by the bits in the A/D shift register to the A/D. Only one strobe is required.

The A/D Shift Register

The A/D shift register is a 56-bit serial register used to configure various parameters of the digitizer. The register does not have an A24 address since it is accessed using the A/D serial register and the A/D strobe register as shown in Figure C-2.

Each time the configuration is changed, all 56 bits must be written to the shift register from the A/D serial register (base +05$_{16}$). The configuration is then latched to the A/D with a single write to the A/D strobe register (base + 0B$_{16}$). The bits of the shift register are defined as follows. Only the unshaded bits are covered in this appendix. Bits 55 - 52 are undefined and are not shown.

Bit	Name	Function	Bit	Name	Function
0	ENSYNC1	Enable Ext1 output	26	OPENINH	Input protection inhibit
1	ENSYNC2	Enable Ext2 output	27	TESTLEDS	LED test
2	ENEXT1	Enable Ext1 input	28	not used	
3	ENEXT2	Enable Ext2 input	29	not used	
4	CALSELA	Lower bit of cal source select	30	not used	
5	CALSELB	Upper bit of cal source select	31	not used	
6	TERM75 1	0 - Ch1 input impedance 50Ω 1 - Ch1 input impedance 75Ω	32	CH1HCAL	0 - switches H_CAL to ch1 +input 1 - switches HI to ch1 +input
7	SINGEND1	0 - Ch1 S/E input disabled 1 - Ch1 S/E input enabled	33	CH1LCAL	0 - switches L_CAL to ch1 -input 1 - switches LO to ch1 -input
8	DUMMY1	Ch1 dummy load select	34	ATT20DB	0 - Ch1 20dB input attenuator ON 1 - Ch1 20dB input attenuator OFF
9	FILTER1	0 - Ch1 filter disabled 1 - Ch1 filter enabled	35	CH1POST	0 - Ch1 20dB post attenuator ON 1 - Ch1 20dB post attenuator OFF
10	PIGGY1	0 - Ch1 attenuators disabled 1- Ch1 attenuators enabled	36	CH1INPT	0 - Ch1 differential input enabled 1 - Ch1 differential input disabled
11	TERM75 2	0 - Ch2 input impedance 50Ω 1 - Ch2 input impedance 75Ω	37	CH1INT	0 - Ch1 6dB int attenuator ON 1 - Ch1 6dB int attenuator OFF
12	SINGEND2	0 - Ch2 S/E input disabled 1 - Ch2 S/E input enabled	38	CH1INT	0 - Ch1 14dB int attenuator ON 1 - Ch1 14dB int attenuator OFF
13	DUMMY2	Ch2 dummy load select	39	not used	
14	FILTER2	0 - Ch2 filter disabled 1 - Ch2 filter enabled	40	CH2HCAL	Ch2 - see bit 32
15	PIGGY2	0 - Ch2 attenuators disabled 1- Ch2 attenuators enabled	41	CH2LCAL	Ch2 - see bit 33
16	CAL1A(0)	Ch1 cal address line 0	42	ATT20DB	0 - Ch2 20dB input attenuator ON 1 - Ch2 20dB input attenuator OFF
17	CAL1A(1)	Ch1 cal address line 1	43	CH2POST	0 - Ch2 20dB post attenuator ON 1 - Ch2 20dB post attenuator OFF
18	CAL1A(2)	Ch1 cal address line 2	44	CH2INPT	0 - Ch2 differential input enabled 1 - Ch2 differential input disabled
19	CAL1A(3)	Ch2 cal address line 3	45	CH2INT	0 - Ch2 6dB int attenuator ON 1 - Ch2 6dB int attenuator OFF
20	CAL1EN	Ch1 cal enable	46	CH2INT	0 - Ch2 14dB int attenuator ON 1 - Ch2 14dB int attenuator OFF
21	CAL2A(0)	Ch2 cal address line 0	47	not used	
22	CAL2A(1)	Ch2 cal address line 1	48 - 49	HCALMUX Output	Bits: 49/51 48/50 0 0 GND 0 1 +10V REF 1 0 +1V REF 1 1 CALSIG
23	CAL2A(2)	Ch2 cal address line 2	50 - 51	LCALMUX Output	

The Arm and Trigger Configuration Registers

The following registers are used to set the digitizer's arm and trigger parameters.

The Abort and Arm Immediate Register — base + 41_{16}

The function of the Abort and Arm Immediate register depends on whether you are writing to the register, or reading the register. Its useage is defined as follows

Address	7	6	5	4	3	2	1	0
base + 41_{16}	register write: measurements aborted register read: arm immediate							

Writing any 8-bit value to this register aborts the current measurements.

Reading this register arms the digitizer if the digitizer is initiated (wait-for-arm state). Once armed, the digitizer moves to the wait-for-trigger state.

The Arm Status Register — base + 43_{16}

The arm status register monitors states and conditions associated with the digitizer's arming hardware. The register bits are defined below.

Address	7	6	5	4	3	2	1	0
base + 43_{16}	128	64	32	16	8	4	2	1
Purpose	Pre-delay	Stage2Q	No arm	Last TRG*	Begin samp	Delayed	Initiated	Initialized

Pre-delay: Bit 7 is set to '1' when an arm signal is received, but the arm delay (as set by the arm delay register) must elapse before the digitizer is armed. When arm immediate is used with the dual rate sampling mode (bit 5: base + $4B_{16}$), bit 7 is set to '1', one reference period before the digitizer is actually armed.

Stage2Q: Bit 6 is set to '1' when an arm signal other than an arm immediate is received.

No arm: Bit 5 is set to '1' while the digitizer is taking pre-arm readings. The bit is set to '0' when the pre-arm count is reached. This bit is checked before an arm immediate is sent (a write to base + 41_{16}).

Last TRG*: Bit 4 is set to ’0’ when the last programmed arm count is reached (base + 55_{16} and base + 57_{16}). The bit is set to ’1’ when the burst of readings associated with the arm are complete.

Begin samp: Bit 3 is set to ’0’ with the first reading in each arm burst and is set to ’1’ after the last reading in each arm burst.

Delayed: Bit 2 is set to ’1’ after the programmed arm delay (base + 51_{16} and base + 53_{16}) has elapsed.

Initiated: Bit 1 is set to ’1’ when the digitizer is initiated and can accept an arm trigger. This bit is monitored when taking multiple bursts of pre- and post-arm readings and transferring the readings over the VME bus. Multiple bursts of pre- and post-arm readings segment memory (Figure 3-13). There is a period (partition window) between each segment that is used by the processor to set up the next segment. When bit 1 is set to ’1’, the next segment is ready for data storage and transfer. See "VMEbus Data Transfers" in Chapter 3 for more information.

Initialized: Bit 0 is set to ’1’ when the digitizer is initialized and is ready to accept an initiate pulse.

The Timebase Initiation Register — base + 45_{16}

The function of the timebase initiation register is defined below.

Address	7	6	5	4	3	2	1	0
base + 45_{16}	register write: initiates the timebase processor register read: sample trigger							

Writing any 8-bit value to the register initiates the timebase processor.

Reading this register generates a sample trigger when the trigger source is an HP-IB Group Execute Trigger or the IEEE-488.2 *TRG command.

The Arm Internal Bus Register — base + 47_{16}

The function of the arm internal bus register is defined below.

Address	7	6	5	4	3	2	1	0
base + 47_{16}	register write: arm trigger							

Writing any 8-bit value to the register generates an arm trigger when the arm source is an HP-IB Group Execute Trigger or the IEEE-488.2 *TRG command.

The Arm Source Register — base + 49_{16}

The Arm source register is used to set the source and slope of the signal which arms the digitizer. The register bits are described below.

Address	7	6	5	4	3	2	1	0
base + 49_{16}	128	64	32	16	8	4	2	1
Purpose	Source 2 slope	Arm source 2			Source 1 slope	Arm source 1		
Setting	0 - positive 1 - negative	0 0 0 - 1 1 1			0 - positive 1 - negative	0 0 0 - 1 1 1		

Arm Source Register Power-on/Reset Settings

At power-on or following a reset, the arm source register is set to 0111 1111 or $7F_{16}$.

Bit Descriptions

Source 2 slope: Bit 7 sets the slope of arm source 2. For all arm sources except a TTLTrg trigger line (bits 6 - 4 =001) and the HP-IB GET command or *TRG command (bits 6 - 4 =010), the slope should be set to positive (0).

Arm source 2: Bits 6 - 4 set arm trigger source 2. Arm source 2 and arm source 1 are ORed together so that an arm from either source arms the digitizer. Setting bits 6 - 4 as follows sets the arm source indicated.

0 0 0 - "Ext 1" BNC connector.
0 0 1* - TTLTrg trigger line (negative-edge triggered)
0 1 0* - HP-IB GET command or IEEE-488.2 *TRG command(negative edge triggered)
0 1 1 - arm when a specified input level on channel 1 is reached
1 0 0 - arm when a specified input level on channel 2 is reached
1 0 1 - ECLTrg0 trigger line
1 1 0 - ECLTrg1 trigger line
1 1 1 - OFF (arm source 2 is disabled)

Source 1 slope: Bit 3 sets the slope of arm source 1. For all arm sources except a TTLTrg trigger line (bits 2 - 0 =001) and the HP-IB GET command or *TRG command (bits 2 - 0 =010), the slope should be set to positive (0).

Arm source 1: Bits 2 - 0 set arm trigger source 1. Arm source 2 and arm source 1 are ORed together so that an arm from either source arms the digitizer. Setting bits 2 - 0 as follows sets the arm source indicated.

0 0 0 - "Ext 1" BNC connector.
0 0 1* - TTLTrg trigger line (negative-edge triggered)
0 1 0* - HP-IB GET command or IEEE-488.2 *TRG command(negative edge triggered)
0 1 1 - arm when a specified input level on channel 1 is reached
1 0 0 - arm when a specified input level on channel 2 is reached
1 0 1 - ECLTrg0 trigger line
1 1 0 - ECLTrg1 trigger line
1 1 1 - arm immediate (arm source 2 must be OFF when selecting this source)

The Arm Control Register — base + $4B_{16}$

The arm control register controls various digitizer arming parameters. The register bits are defined below.

Address	7	6	5	4	3	2	1	0
base + $4B_{16}$	128	64	32	16	8	4	2	1
Purpose	not used	enintr0	2 speed	reclk/ 10	pre-trig	thold	triginf	delay ref
Setting	---	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on

Arm Control Register Power-on/Reset Settings

At power-on or following a reset, the arm control register is set to 0000 0001 or 01_{16}.

Bit Descriptions

enintr0: Enable local interrupt 0. Setting bit 6 to '1' enables local interrupt 0 to go 'high' if an arm trigger is received while arm source hold is set, or if the digitizer is already armed. The bit is cleared ('0') when the digitizer is initiated.

2 speed: Setting bit 5 to '1' enables the dual rate sampling mode.

reclk/10: Setting bit 4 to '1' causes the reference divider to be reclocked by the reference clock / 10. Setting bit 4 to '0' causes the reference divider to be reclocked by the reference clock.

pre-trig: Setting bit 3 to '1' enables pre- and post-arm readings.

thold: Setting bit 2 to '1' sets arm trigger hold which prevents the digitizer from accepting arm signals from any source except an arm immediate (writing any value to base + 41_{16}). This bit is used to suspend arming while changing the arm source when the digitizer is initiated.

triginf: Setting bit 1 to '1' sets the digitizer to accept an infinite number of arm triggers. The bit overrides the arm count registers, however; the arm count remains active. Thus, if a number of arms less than the arm count have occurred when bit 1 is set, the counter will keep track of the number of arms which have occurred. When bit 1 is cleared ('0'), the digitizer returns to the idle state if the arm count was reached. Otherwise, arms are accepted until the arm count is reached.

delay ref: When bit 0 is set to '1' , the arm delay is derived from the reference clock. When bit 0 is cleared ('0'), the arm delay is derived from the reference clock / 10. See "Setting the Arm Delay" for more information.

The Trigger Source Register — base + $4D_{16}$

The trigger source register is used to set the digitizer's trigger (sample) source. The register bits are defined below.

Address	7	6	5	4	3	2	1	0
base + $4D_{16}$	128	64	32	16	8	4	2	1
Purpose	Sample/ Hold	Sample Infinite	Sample Once	Trigger Source			Internal TTL Sources	
Setting	0 - OFF 1 - ON	0 - OFF 1 - ON	0 - OFF 1 - ON	0 0 0 - 1 1 1			0 0 - 1 1	

Trigger Source Register Power-on/Reset Settings

At power-on or following a reset, the trigger source register is set to 0010 0000 or 20_{16}.

Bit Descriptions

Sample/Hold: Setting bit 7 to ’1’ sets sample trigger hold which prevents the digitizer from accepting sample trigger signals.

Sample Infinite: Setting bit 6 to ’1’ sets infinite sample triggers. Triggering continues until aborted (base + 41_{16}) or until the bit is set to ’0’ and the post arm reading count is reached. One sample trigger occurs after the bit is set to ’0’ even if the post-arm trigger count is reached. Sample Infinite overrides Sample Once (bit 5).

Sample Once: Setting bit 5 to ’1’ causes the digitizer to take one sample and return to the idle state, regardless of the pre-arm and post-arm reading counts. This bit should not be set if the pre-arm and post-arm reading mode is set (arm control register bit 3: base + $4B_{16}$). The bit is overridden by bit 6 (Sample Infinite).

Trigger Source: Bits 4 - 2 set the digitizer trigger (sample) source. Setting bits 4 - 2 as follows sets the trigger source indicated.

0 0 0 - reference oscillator output.
0 0 1 - ECLTrg0 trigger line.
0 1 0 - "Ext 1" BNC connector.
0 1 1 - internal TTL source as specified by bits 1 - 0.
1 0 0 - reference period / n.
1 0 1 - ECLTrg1 trigger line.
1 1 0 - "Ext 2" BNC connector.
1 1 1 - not used.

Internal TTL Sources: Bits 1 - 0 are additional sample sources which are selected when bits 4 - 2 are set to 011. The sources set by bits 1 - 0 are:

0 0 - VME (VXI data transfer) bus. Trigger when data register (base + 12_{16}) is read.
0 1 - HP-IB Group Execute Trigger or IEEE-488.2 *TRG command
1 0 - TTLTrg trigger line
1 1 - user during local bus data transfer (does not take data)

The Reference Oscillator Register — base + $4F_{16}$

The reference oscillator register sets the reference source from which the sample rate is derived. The register is also used to output synchronization signals.

Address	7	6	5	4	3	2	1	0
base + $4F_{16}$	128	64	32	16	8	4	2	1
Purpose	Arm source 1 enable	ECLTrg1 source		ECLTrg0 source		Reference oscillator source		
Setting	0 - enabled 1 - disabled	0 0 - 1 1		0 0 - 1 1		0 0 0 - 1 0 0		

Reference Oscillator Register Power-on/Reset Settings

At power-on or following a reset, the reference oscillator register is set to 0111 1000 or 78_{16}.

Bit Descriptions

Arm source 1 enable: Setting bit 7 to '0' enables the arm source 1 trigger source (arm source register: base + 49_{16}) to arm the digitizer. Setting bit 7 to '1' disables arm source 1.

ECLTrg1 source: Bits 6 - 5 set the signal source that is output on the ECLTrg1 trigger line. The sources include:

0 0 - a 25 ns wide negative-going pulse each time a convert pulse is sent tothe A/D converter.
0 1 - reference oscillator as selected by bits 2 - 0. The falling edge is synchronous with the rising edge of the internal 20 MHz oscillator, the ECLTrg lines, CLK10, and is synchronous with the falling edge of an external reference oscillator.
1 0 - reserved.
1 1 - off. Outputs an ECL high level which then allows ECLTrg1 to be used as an input.

ECLTrg0 source: Bits 4 - 3 set the signal source that is output on the ECLTrg0 trigger line. The sources include:

0 0 - a 25 ns wide negative-going pulse each time a convert pulse is sent tothe A/D converter.
0 1 - reference oscillator as selected by bits 2 - 0. The falling edge is synchronous with the rising edge of the internal 20 MHz oscillator, the ECLTrg lines, CLK10, and is synchronous with the falling edge of an external reference oscillator.
1 0 - reserved.
1 1 - off. Outputs an ECL high level which then allows ECLTrg1 to be used as an input.

Reference oscillator source: Bits 2 - 0 set the reference oscillator source from which the sample rate is derived. The sources include:

0 0 0 - the digitizer's internal 20 MHz oscillator.
0 0 1 - backplane trigger line ECLTrg0.
0 1 0 - backplane trigger line ECLTrg1.
0 1 1 - the EXT2 front panel BNC connector.
1 0 0 - backplane 10 MHz (CLK10) signal.

The Arm delay Register — base + 51_{16} and base +53_{16}

The arm delay is set using the arm delay registers defined below.

Address	7	6	5	4	3	2	1	0
base + 51_{16} base + 53_{16}	base + 51_{16} = arm delay most significant byte base + 53_{16} = arm delay least significant byte							

Register 51: Contains the most significant byte of the arm delay.
Register 53: Contains the least significant byte of the arm delay.

The Arm Count Register — base + 55_{16} and base + 57_{16}

The arm count is set using the arm count registers defined below.

Address	7	6	5	4	3	2	1	0
base + 55_{16} base + 57_{16}	base + 55_{16} = arm count most significant byte base + 57_{16} = arm count least significant byte							

Register 55: Contains the most significant byte of the arm count.
Register 57: Contains the least significant byte of the arm count.

The Arm Count Latch Register — base + 59_{16}

The arm count latch register is used to load the arm count and initialize the trigger counters. It is written to three times prior to sending the timebase processor an initiate pulse.

Address	7	6	5	4	3	2	1	0
base + 59_{16}	register write: loads the arm count and initializes the trigger counters							

The Trigger Immediate Register — base + $5D_{16}$

A trigger immediate occurs when any 8-bit value is written to the trigger immediate register shown below.

Address	7	6	5	4	3	2	1	0
base + $5D_{16}$	register write: sends trigger (sample) immediate							

The Decade Division Register — base + 61_{16}

The decade division register is used with the binary division register to divide the reference frequency in order to obtain the desired sample rate. The register bits are defined below.

Address	7	6	5	4	3	2	1	0
base + 61_{16}	128	64	32	16	8	4	2	1
Purpose	divide by 10E7	divide by 10E6	divide by 10E5	divide by 10E4	divide by 1000	divide by 100	divide by 10	divide by 1
Setting	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on

Only one bit at a time can be set in this register. The decade division selected (bits 7 - 0 or bit 0 of base + 63_{16}) is combined with the binary division selected (bits 3 - 1 of base + 63_{16}) to derive the sample rate from the reference source.

The Binary Division Register — base + 63_{16}

The binary division register is used with the decade division register to divide the reference frequency in order to obtain the desired sample rate. The register bits are defined below.

Address	7	6	5	4	3	2	1	0
base + 63_{16}	128	64	32	16	8	4	2	1
Purpose	reserved (bits 7 - 4 must be set to 1000)				divide by 4	divide by 2	divide by 1	divide by 10E8
Setting	1	0	0	0	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on	0 - off 1 - on

The Pre-Arm Reading Count Registers base + 73_{16} and base + 75_{16}

The pre-arm reading count registers are defined below.

Address	7	6	5	4	3	2	1	0
base + 73_{16} base + 75_{16}	base + 73_{16} = pre-arm reading count least significant byte base + 75_{16} = pre-arm reading count most significant byte							

Register 73: Contains the least significant byte of the pre-arm reading count.

Register 75: Contains the most significant byte of the pre-arm reading count.

Pre-Arm Reading Count Register power-On/Reset Settings

At power-on or following a reset, the digitizer is set for 0 pre-arm readings. When the pre-arm reading mode is set (base +$4B_{16}$ bit 3 set to '1'), the digitizer is set for 3 pre-arm readings.

The Post-Arm Reading Count Registers base + 77_{16} to base + $7B_{16}$

The post-arm reading count registers are defined below.

Address	7	6	5	4	3	2	1	0
base + 77_{16} base + $7B_{16}$	base + 77_{16} = post-arm reading count least significant byte base + 79_{16} = post-arm reading count middle significant byte base + $7B_{16}$ = post-arm reading count most significant byte							

Register 77: Contains the least significant byte of the pos-arm reading count.

Register 79: Contains the middle significant byte of the post-arm reading count.

Register 7B: Contains the most significant byte of the post-arm reading count.

Post-Arm Reading Count Register power-On/Reset Settings

At power-on or following a reset, the digitizer is set for 7 post-arm readings. The digitizer is also set for 7 post-arm readings when the pre-arm reading mode is set (base +$4B_{16}$ bit 3 set to '1').

The Memory Control Registers

The following memory control registers are used to initialize digitizer memory.

The Traffic Register — base +02_{16}

The traffic register is a read/write register that selects data and clock sources for the high-speed data bus. The bits are defined as follows.

Address	7	6	5	4	3	2	1	0
base + 02_{16}	128	64	32	16	8	4	2	1
Purpose	Data Register Mode		not used	Read Data	High-Speed Clock Source		High-Speed Data Source	
Setting	0 0 - invalid 0 1 - channel 1 1 0 - channel 2 1 1 - alternate channels			0 - data not ready 1 - data ready	0 0 - pulse register 0 1 - A/D 1 0 - data register 1 1 - Local bus		0 0 - A/D 0 1 - data register 1 0 - not used 1 1 - memory data	

Traffic Register Power-on/Reset Settings

At power-on or following a reset, the traffic register is set to 1100 0100 or $C4_{16}$.

Bit Descriptions

Data Register Mode. Bits 7 - 6 specify how data will be presented when read by the data register, and how it will be written into memory as written by the data register. Writing data to only one channel's memory is not recommended since invalid data is placed in the unselected channel. When 'alternate channels' is selected, the two channels alternate, beginning with the channel previously selected. For example, alternate channels beginning with channel 2 can be achieved by setting the Data Register Mode field to 1 0, and then setting it to 1 1.

Read Data. Bit 4 is set to 1 when data can be read from digitizer memory. See "Segmented Reading Transfers" in Chapter 3 for information on how the bit is used.

High-Speed Clock Source. Bits 3 - 2 set the source which clocks data transfers over the internal high-speed bus.

High-Speed Bus Source. Bits 1 - 0 select the data source which drives the internal high-speed bus.

The high-speed internal (data) bus routes data between the A/D, digitizer memory, the local bus, and the VME (VXI data transfer) bus (Figure 3-7). There is no user-access to the high-speed internal bus.

The Pulse Register — base +08_{16}

The pulse register is a read/write register that generates high-speed clock signals when the traffic register's high-speed clock source is set to 'pulse register' .

Address	7	6	5	4	3	2	1	0
base + 08_{16}	register read/write: generates high-speed bus clock pulse							

Reading, or writing to this register generates a clock pulse for the internal high-speed bus.

The Channel ID Register — base +$0A_{16}$

The channel ID register is a read/write register that allows user-defined identifiers to be appended to each channel's readings.

Address	7	6	5	4	3	2	1	0
base + $0A_{16}$	channel 1 ID LSB			MSB	channel 2 ID LSB			MSB

The ID assigned is represented by the 4 least significant bits of each reading. The ID bits are not stored in memory with the readings, but are appended to each reading as it is read over the VME (VXI data transfer) bus or Local bus.

The Data Register — base +$0C_{16}$

The data register is a read/write register used to retrieve readings from digitizer memory or to retrieve them from the digitizer's A/D converter.

Address	7	6	5	4	3	2	1	0
base + $0C_{16}$	register read/write: retrieves a reading from digitizer memory							

Each digitizer reading is stored in memory as a 12-bit, two's complement number. When a reading is retrieved, it is expanded to 16-bits with the reading left-justified in the 16-bit field. The four least significant bits are normally zeros, but can be set as indicated by the channel ID register (base +$0A_{16}$). The channel from which readings are retrieved is set with the "Data register Mode" field of the traffic register (base +02_{16}).

The Memory Control Register — base +21_{16}

The memory control register is a read/write register that controls the operation of digitizer memory. The register bits are defined below.

Address	7	6	5	4	3	2	1	0
base + 02_{16}	128	64	32	16	8	4	2	1
Purpose	Backup Enable	TTL Mux	BNC Mux			Address Count Enable	Memory Read Enable	Memory Write Enable
Setting	0 - OFF 1 - ON					0 - OFF 1 - ON	0 - reset 1 - enable	0 - reset 1 - enable

Memory Control Register Power-on/Reset Settings

At power-on or following a reset, the memory control register is set to 0011 1000 or 38_{16}.

Bit Descriptions

Backup Enable. Bit 7 is used to enable/disable the battery which maintains memory at power-down.

TTL Mux and BNC Mux. Bit 6 and bits 5 - 3 are used by the digitizer's timebase processor. Their usage is not covered in this appendix.

Address Count Enable. Bit 2 is used to initialize the memory address counter. Setting bit 2 to '0' disables the memory address counter but sets it to receive the next memory address from the base address registers. Setting bit 2 to '1' enables the address counter to receive addresses from the terminal address register, thus allowing the counter to wrap around and make repeated passes through memory.

Memory Read Enable. Setting bit 1 to '1' places digitizer memory in the read mode. Data is placed on the internal high-speed bus if enabled by bits 1 - 0 of the traffic register.

Memory Write Enable. Bit 0 enables A/D readings to be written to digitizer memory when bit 1 (memory read enable) is set to '0'.

The Memory Address Registers — base +23_{16} to base +27_{16}

The memory address registers are read only registers that return the address where the last reading in the set will be stored. These registers are useful for determining if the digitizer has been re-initiated by indicating the number of readings that have been taken. For example, if 100 readings are to be taken when the digitizer is re-initiated, the memory address registers point to the location where the 100th reading is to be stored.

Address	7	6	5	4	3	2	1	0
base + 23_{16}	128	64	32	16	8	4	2	1
Purpose	Wrapped	Memory Size	Address Valid	Local Interrupt	Reserved	current address		
Setting	0 - no 1 - yes	0 - 128K 1 - 512K	0 - no 1 - yes			A18 - A16		

Address	7	6	5	4	3	2	1	0
base + 25_{16}	Memory address register 1: current address A15 - A08							

Address	7	6	5	4	3	2	1	0
base + 27_{16}	Memory address register 2: current address A07 - A00							

Bit Descriptions

Wrapped. Bit 7 indicates if the data memory address counter has wrapped around.

Memory Size. Bit 6 indicates the number of 2-channel readings digitizer memory can hold. For all digitizers this is 512K.

The Terminal Address Register — base +2B$_{16}$

The terminal address register is a read/write register that sets the last address of a memory segment. It is used with the base address registers to define the memory segment.

Address	7	6	5	4	3	2	1	0
base + 2B$_{16}$	Terminal address							Base address

Terminal Address. The terminal address field specifies the last address in a memory segment before returning to the base address.

Base Address. The Base Address (bit 0) is the most significant bit (bit 18) of the memory segment's base address.

The Base Address Registers — base +2D$_{16}$ and base +2F$_{16}$

The base address registers are read/write registers that specify the beginning of a memory segment. This is the address where the segment starts after either being reset by the address count enable bit (bit 2) of the memory control register, or after reaching the terminal address.

Address	17	16	15	14	13	12	11	10
base + 2D$_{16}$	Base address 0: start address MSByte							

Address	9	8	7	6	5	4	3	2
base + 2F$_{16}$	Base address 1: start address LSByte							

The most significant byte of the segment's base address is store in the least significant byte of register base +2D$_{16}$. The least significant byte of the base address is stored in the least significant byte of base +2F$_{16}$. The data written to these registers is held in a temporary register until it is loaded into the actual address counter by clearing and then setting the address count enable bit (bit 2) of the memory control register (base +21$_{16}$).

Configuring the Digitizer Input

This section contains the procedures used to configure the digitizer's input section. The configuration includes:

- Enabling the Single Ended and Differential Inputs
- Setting the input impedance
- Enabling the 10 MHz filter
- Setting the measurement range

The digitizer must be in the idle state when configuring the input section.

Using the A/D Shift Register

Each input parameter listed above is set by the digitizer's A/D shift register. This register is accessed through the A/D status register (base + 03_{16}), the A/D serial register (base + 05_{16}), and by the A/D parallel strobe register (base + $0B_{16}$) as shown in Figure C-2. The procedure for setting the configuration using the register is described in the following section.

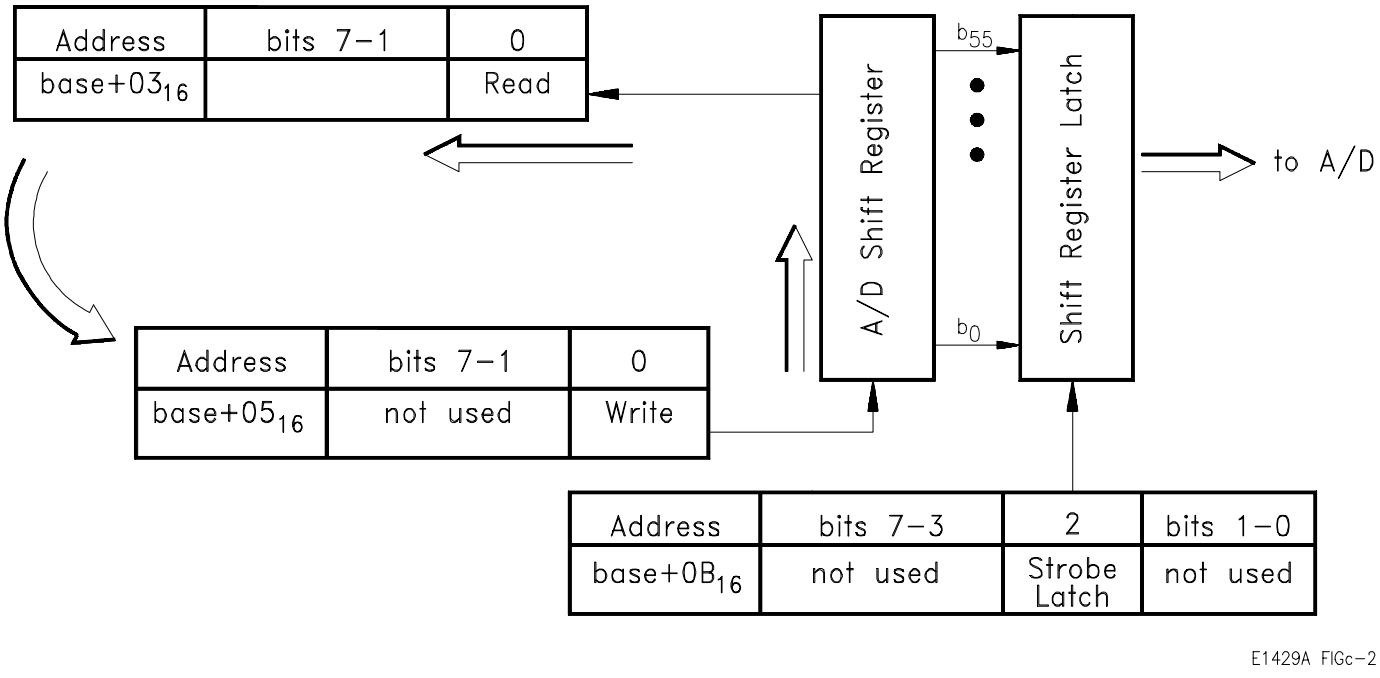


Figure C-2. Accessing the A/D Shift Register

Reading and Writing to the Shift Register

1. Define a programming loop which counts from 55 to 0 and which contains the following.

A. Read and save the current shift register bit at the output (bit 55) position using the A/D status register (base + 03_{16}).

B. If the loop count **does not** equal the bit position to be changed, write back the bit using the A/D serial register (base + 05_{16}). This restores the bit and shifts it to the bit 0 position, which shifts a new shift register bit to the output (bit 55) position.

C. If the loop count **does** equal the bit position to be changed, write the new bit setting using the A/D serial register (base + 05_{16}). This sets the bit and shifts it to the bit 0 position, which shifts a new shift register bit to the output (bit 55) position.

D. Continue through the loop until the loop count is 0. This shifts each bit back to its original position.

2. After the configuration is set, write a value of 4 to the A/D parallel strobe register (base + $0B_{16}$). This copies each shift register bit to the shift register latch at which point the configuration is set.

Enabling the Inputs

At power-on or following a reset, the single-ended and differential inputs of channels 1 and 2 are enabled such that input signals can be applied. An input can be disabled (and later enabled) using the following bits of the A/D shift register.

Bit Position	Name	Setting
7	SINGEND1	0 - Ch1 S/E input disabled 1 - Ch1 S/E input enabled
12	SINGEND2	0 - Ch2 S/E input disabled 1 - Ch2 S/E input enabled
36	CH1INPT	0 - Ch1 differential input enabled 1 - Ch1 differential input disabled
44	CH2INPT	0 - Ch1 differential input enabled 1 - Ch1 differential input disabled

Procedure

With the bit positions known, disable/enable the inputs as required using the procedure for reading and writing to the shift register.

Setting the Input Impedance

The impedance of the single-ended inputs can be set to 50Ω or 75Ω. At power-on or following a reset, the impedance is set to 50Ω. The impedance is changed using the following bits of the A/D shift register.

Bit Position	Name	Setting
6	TERM75 1	0 - Ch1 50Ω input imedance selected 1 - Ch1 75Ω input impedance select
11	TERM75 2	0 - Ch2 50Ω input imedance selected 1 - Ch2 75Ω input impedance select

Procedure

With the bit position known, set the input impedance as required using the procedure for reading and writing to the shift register.

Enabling the 10 MHz Filter

At power-on or following a reset, the 10 MHz filter is switched out of the signal path (disabled) of the single-ended and differential inputs. The filter can be switched into the path (enabled) using the following bits of the A/D shift register.

Bit Position	Name	Setting
9	FILTER1	0 - Ch1 10 MHz filter disabled 1 - Ch1 10 MHz filter enabled
14	FILTER 2	0 - Ch2 10 MHz filter disabled 1 - Ch2 10 MHz filter enabled

Procedure

With the bit position known, disable/enable the filter as required using the procedure for reading and writing to the shift register.

Setting the Measurement Range

The digitizer measurement range is set using a series of attenuators (Figure 3-3). Table C-1 shows the attenuator settings used to select the corresponding measurement range.

Note

There is a 3 ms relay settling time following each range change. Samples cannot be taken during the settling time.

Table C-4. HP E1429 Digitizer Attenuator Settings

Measurement Range	20 dB Input Attenuator bits 34, 42	20 dB Post Attenuator bits 35, 43	Internal Attenuators 6 dB bits 37,45	14 dB bits 38,46
-0.10230 to 0.10235	OFF	OFF	OFF	OFF
-0.2046 to 0.2047	OFF	OFF	ON	OFF
-0.5115 to 0.51175	OFF	OFF	OFF	ON
-1.0230 to 1.0235	OFF	ON	OFF	OFF
-2.0460 to 2.0470	OFF	ON	ON	OFF
-5.115 to 5.1175	OFF	ON	OFF	ON
-10.230 to 10.235	ON	ON	OFF	OFF
-20.460 to 20.470	ON	ON	ON	OFF
-51.150 to 51.175	ON	ON	OFF	ON
-102.30 to 102.35	ON	ON	ON	ON

The A/D shift register bits used to turn the attenuators off and on are listed below.

Bit Position	Name	Setting
10	PIGGY1	0 - Ch1 attenuators disabled 1 - Ch1 attenuators enabled
34	ATT20DB	0 - Ch1 20dB input attenuator ON 1 - Ch1 20dB input attenuator OFF
35	CH1POST	0 - Ch1 20dB post attenuator ON 1 - Ch1 20dB post attenuator OFF
37	CH1INT	0 - Ch1 6dB internal attenuator ON 1 - Ch1 6dB internal attenuator OFF
38	CH1INT	0 - Ch1 14dB internal attenuator ON 1 - Ch1 14dB internal attenuator OFF
15	PIGGY2	0 - Ch2 attenuators disabled 1 - Ch2 attenuators enabled
42	ATT20DB	0 - Ch2 20dB input attenuator ON 1 - Ch2 20dB input attenuator OFF
43	CH2POST	0 - Ch2 20dB post attenuator ON 1 - Ch2 20dB post attenuator OFF
45	CH2INT	0 - Ch2 6dB internal attenuator ON 1 - Ch2 6dB internal attenuator OFF
46	CH2INT	0 - Ch2 14dB internal attenuator ON 1 - Ch2 14dB internal attenuator OFF

Procedure

1. Route the input signal to the attenuators.

   A. To set the measurement range, the input signal must be routed to the attenuators. This is done by setting bit 10 (channel 1) or bit 15 (channel 2) to '1'.

2. Set the required attenuation.

   A. From Table C-1 and with the bit positions known, set the bits for the required attenuation (measurement range) using the procedure for reading and writing to the shift register.

3. Copy the shift register bits to the shift register latch.

   A. Write a value of 4 to the A/D parallel strobe register (base $+0B_{16}$) to copy the shift register bits to the shift register latch.

Using the Packed Reading Format

When the measurement range is set using registers, the reading resolution used to convert the readings to voltages is unknown to the processor. As a result, the packed data format should be used, and the readings converted by the user as described in Chapter 3 - "Understanding the Digitizer".

Arming and Triggering

This section contains the procedures used to configure the digitizer's arm and triggering (timebase) hardware.

Checking the Idle State

Except as noted, configuring the arming and triggering hardware occurs when the digitizer is in the idle state. The register used to check the idle state is listed below.

- Arm status register
  base + 43_{16}

Procedure

1. Determine if the digitizer is in the idle state by checking bit 1 of the arm status register (base + 43_{16}).

   If bit 1 is set to '0', the digitizer is in the idle state and can be configured.

   If bit 1 is set to '1', the digitizer is in the initiated state and should not be configured.

Setting the Digitizer Configuration

The digitizer configuration covered in this section includes:

- setting the arm sources
- setting the arm count
- setting the arm delay
- setting the reference source
- setting the trigger source
- sending arm immediate
- sending trigger immediate
- aborting measurements

Note The trigger count (pre-arm and post-arm reading counts) and the sample rate are set when the timebase processor is initialized. See "Initializing the Timebase Processor" in the section "Re-initiating the Digitizer" for more information.

Setting the Arm Sources

The registers used to set the arm sources are listed below.

- Arm Source register
  base + 49_{16}
- Arm Control register
  base + $4B_{16}$ (used when changing arm source while initiated)

Procedure The arm source can be set/changed when the digitizer is in the idle state or while it is initiated.

1. Setting/changing the arm source in the idle state:

   A. Write the decimal equivalent bit pattern to the arm source register (base + 49_{16}).

2. Setting/changing the arm source while initiated:

   A. Suspend the arm trigger by setting bit 2 of the arm control register (base + $4B_{16}$) to '1' . Retain the settings of the other register bits.

   B. Write the decimal equivalent bit pattern for the desired arm source to the arm source register (base + 49_{16}).

   C. Enable the arm trigger by setting bit 2 of the arm control register (base + $4B_{16}$) to '0' . Retain the settings of the other register bits.

Setting the Arm Count

The registers used to set the arm count are listed below.

- Arm count registers
  base + 55_{16} and base + 57_{16}

Procedure

1. Load the arm count registers.

   A. With the digitizer in the idle state, write the decimal equivalent of the most significant byte to register 55. Writing the decimal equivalent of the least significant byte to register 57. You can set the arm count from 1 to 65,535 arms. For example, to program an arm count of 20:

MSB	LSB
0 0 0 0 0 0 0 0	0 0 0 1 0 1 0 0
0	20_{10}

   0 is written to register 55
   20_{10} is written to register 57

   To program an arm count of 300:

MSB	LSB
0 0 0 0 0 0 0 1	0 0 1 0 1 1 0 0
1_{10}	44_{10}

   1_{10} is written to register 55
   44_{10} is written to register 57

Setting the Arm Delay

The arm delay is the amount of **additional** delay to occur from when the digitizer accepts the arm signal, to when it enters the wait-for-trigger state. There is always a delay of one reference clock cycle. The registers used to set the arm delay are listed below.

- Arm Control register
  base + $4B_{16}$
- Arm Delay registers
  base + 51_{16} and base + 53_{16}

Procedure

1. Determine the reference period.

   A. The reference from which the arm delay is derived is set with bit 0 of the arm control register. The setting of bit 0 (to '0' or '1') depends on the reference clock and the amount of delay required. Determine the reference from which the delay is derived and set bit 0 accordingly. Retain the settings of bits 7 - 1.

   If delay / reference clock period ≤ 65,534

   bit 0 is set to '1' and the maximum delay is 65,534 * reference period

   If delay / reference clock period > 65,534

   bit 0 is set to '0' and the maximum delay is 655,350 * reference period

2. Load the arm delay registers.

   A. Write the decimal equivalent of the most significant byte to register 51. Write the decimal equivalent of the least significant byte + 1 to register 53. The additional count (1) is required because there is always a one reference cycle delay from when the digitizer is armed to when it enters the wait-for-trigger state (i.e. is ready to begin sampling).

   For example, to program an arm delay of 1 ms:

MSB	LSB
0 1 0 0 1 1 1 0	0 0 1 0 0 0 0 1
78_{10}	33_{10}

   78_{10} is written to register 51
   33_{10} is written to register 53

Setting the Reference Source

The reference source from which the sample rate is derived is set with the register listed below.

- Reference oscillator register
  base +$4F_{16}$

Procedure

Write the decimal equivalent bit pattern for the desired reference source to the reference source register (base + $4F_{16}$). Retain the settings of bits 7 - 3.

Setting the Trigger Source

The register used to set the digitizer's trigger (sample) source is listed below.

- Trigger source register
  base + $4D_{16}$

Procedure

The trigger source can be set/changed when the digitizer is in the idle state or while it is initiated.

1. Setting/changing the trigger source while in the idle state:

   A. Write the decimal equivalent bit pattern to the trigger source register while retaining the settings of the other bits (base + $4D_{16}$).

2. Setting/changing the trigger source while initiated:

   A. Suspend the sample trigger by setting bit 7 of the trigger source register (base + $4D_{16}$) to '1' . Retain the settings of the other register bits.

   B. Write the decimal equivalent bit pattern for the desired trigger source to the trigger source register.

   C. Enable the sample trigger by setting bit 7 of the trigger control register (base + $4D_{16}$) to '0' . Retain the settings of the other register bits.

Sending an Arm Immediate Signal

An arm immediate signal arms the digitizer, overriding arm source 'hold' and any programmed arm delay. The registers used to send an arm immediate signal are listed below.

- Abort and Arm Immediate register
  base + 41_{16}
- Arm Status register
  base + 43_{16}

Procedure

1. Determine the arm state:

   A. Read bits 7 and 5 of the arm status register (base + 43_{16}). If bit 7 is '1', the digitizer is already armed and it is not necessary to send an arm immediate. If bit 5 is '1', arm immediate **must not** be sent since the pre-arm reading count has not been reached.

2. Send the arm immediate signal.

A. Read the Abort and Arm Immediate register (base + 41_{16}). Reading the registers sends an arm immediate signal.

Sending a Trigger Immediate Signal

A trigger immediate signal triggers the digitizer, overriding the sample/hold bit of the trigger source register.'The registers used to send a trigger immediate signal are:

- Trigger Immediate register
  base + $5D_{16}$
- Trigger Source register
  base + $4D_{16}$

Procedure

1. Suspend sample triggers (e.g. trigger hold).

A. Set trigger source register (base + $4D_{16}$) bit 7 to '1' to suspend sample triggers. Be sure retain the settings of bits 6 - 0.

2. Send a trigger immediate signal.

A. Write any 8-bit value to the trigger immediate register (base + $5D_{16}$).

3. Re-enable sample triggers.

A. Set bit 7 (base + $4D_{16}$) to '0' to enable sampling for the next reading (measurement) sequence. Retain the settings of bits 6 - 0.

Aborting Measurements

The registers used to abort digitizer measurements are shown below. "base" is the A24 base address.

- Trigger Source register
  base + $4D_{16}$
- Abort and Arm Immediate register
  base + 41_{16}

Procedure

1. Suspend sample triggers (e.g. trigger hold).

A. Set trigger source register (base + $4D_{16}$) bit 7 to '1' to suspend sample triggers. Be sure retain the settings of bits 6 - 0.

2. Send abort signal.

A. Write any 8-bit value to the Abort and Arm Immediate register (base + 41_{16}).

3. Re-enable sample triggers.

A. Set bit 7 (base + $4D_{16}$) to '0' to enable sampling for the next reading (measurement) sequence.

Re-initiating the Digitizer

Initiating the digitizer places the digitizer in the wait-for-arm state. When an arm is received while in this state, the digitizer moves to the wait-for-trigger state where it samples when trigger signals are received.

This section describes how the digitizer is re-initiated. A re-initiation is done following a SCPI CONFigure ... INITiate sequence. Using register reads and writes, digitizer parameters can be changed and the digitizer re-initiated at a faster rate than sending another INITiate command. There are two parts to the re-initiation sequence:

1. initializing digitizer memory
2. initializing and initiating the timebase processor

The re-initiation sequence described in this section is restricted to **post-arm readings only**. The other digitizer parameters can be set as required.

Initializing Digitizer Memory

The registers used to initialize digitizer memory are summarized below.

- memory control register
  base + 21_{16}
- terminal address register
  base + $2B_{16}$
- base address registers
  base + $2D_{16}$ and base + $2F_{16}$

Initializing digitizer memory involves initializing the memory control register, setting the ending and beginning addresses in memory where the data will be stored, and enabling data to be written to memory. For the complete digitizer re-initiation to occur, the registers must be read and written to in the sequence covered in the procedure.

Procedure 1. Initialize the memory control register.

A. Set bits 2 - 0 of the memory control register (base + 21_{16}) to '0'. Retain the settings of bits 7 - 3.

2. Determine the starting address of the memory segment.

A. The segment size is the number of readings to be taken and must be divisible by 4. The starting address is computed as:

starting address = ending address - (segment size - 1)

The re-initiation procedure assumes one segment of post-arm readings. Therefore, the ending address (524,287) is the size of digitizer memory.

3. Set the terminal (ending) address.

A. The terminal address is the ending address of the memory segment. This address is computed as:

terminal address = (ending address - 4095) / 2048

Again, the procedure is for one segment of post-arm readings. Therefore, the terminal address is:

terminal address = (524,287 - 4095) / 2048 = 254

B. Divide the starting address (see Step 2) by 4. If this value is > 65,535 ($FFFF_{16}$), add 1 to the terminal address value (254).

C. Write the terminal address to the terminal address register (base + $2B_{16}$).

4. Set the base (starting) address.

A. The base address is the (starting address / 4) of the memory segment. Write the most significant byte of (starting address / 4) to the base 0 address register (base + $2D_{16}$). Write the least significant byte of (starting address / 4) to the base 1 address register (base + $2F_{16}$).

For example, if the segment size is 10,000 readings, the terminal and base addresses would be:

start address = 524,287 - 9,999 = 514,288

514,288 / 4 = 128,572 = $1F63C_{16}$

terminal address = 254 + 1 (since 128,572 > 65,535)

base address = F63C (the '1' is part of the terminal address: 254 + 1)

MSB	LSB
1 1 1 1 0 1 1 0	0 0 1 1 1 1 0 0
246_{10}	60_{10}

246_{10} is written to register base + $2D_{16}$
60_{10} is written to register base + $2F_{16}$

5. Enable memory to be written to.

A. Enable data to be written to digitizer memory and enable the address counter by setting memory control register (base + 21_{16}) bits 2 - 0 to '1 0 1'.

Initializing and Initiating the Timebase Processor

The second step in re-initiating the digitizer is initializing and initiating the timebase processor. The timebase processor must be initialized each time the pre-arm reading count, post-arm reading count, or sample rate is changed.

The registers used to initialize the processor are summarized below.

Timebase reset

- Abort and arm immediate register
  base + 41_{16}
- Timebase reset register
  base + $5F_{16}$
- Interpolator control register
  base + 65_{16}
- Stop data register
  base + 67_{16}
- Interpolator calibration register
  base + 69_{16}
- Self test register
  base + $6B_{16}$
- Time base registers
  base + $7D_{16}$ and base + $7F_{16}$

Trigger count

- Pre-arm reading count registers
  base + 73_{16} to base +75_{16}
- Post-arm reading count registers
  base + 77_{16} to base + $7B_{16}$

Sample rate

- Arm control register
  base + $4B_{16}$
- Decade division register
  base + 61_{16}
- Binary division register
  base + 63_{16}

Data source

- Traffic register
  base + 02_{16}
- Pulse register
  base + 08_{16}

Timebase initiate

- Timebase initiation register
  base + 45_{16}
- Arm count latch register
  base + 59_{16}

For the complete digitizer re-initiation to occur, the registers must be read and written to in the sequence given in the procedure.

Procedure 1. Reset and initialize the timebase processor.

A. Write the following data to the registers indicated. Except for the abort and arm immediate register (base $+41_{16}$), these registers must not be set to any other values; therefore, they are not listed in the "Register Descriptions" section.

write any 8-bit value to base + 41_{16}
write any 8-bit value to base + $5F_{16}$
write a value of 1 to base + 65_{16}
write a value of 5 to base + 67_{16}
write a value of 0 to base + 69_{16}
write a value of 0 to base + $6B_{16}$
write a value of 0 to base + $7D_{16}$
write a value of 0 to base + $7F_{16}$

2. Set the Trigger count.

The number of readings the digitizer takes each time it is armed is set with the pre-arm and post-arm reading count registers listed below.

- Pre-arm reading count registers
  base + 73_{16} and base $+75_{16}$
- Post-arm reading count registers
  base + 77_{16} and base + $7B_{16}$

When taking (x) pre-arm and (y) post-arm readings, and (x) and (y) are the intended number of each set of readings, the count loaded into the pre-arm reading count registers is x-2. The count loaded into the post-arm reading count registers is y-6. **When taking post-arm readings only, the pre-arm count is 1 and the post-arm count is y-3.**

A. Load the pre-arm reading count registers. With the digitizer in the idle state, write the decimal equivalent of the least significant byte to register 73. Write the decimal equivalent of the most significant byte to register 75. You can set the pre-arm reading count from 3 to 65,535 readings. For example, to program 500 pre-arm readings, a pre-arm reading count of 498 (500 - 2) is loaded into the registers.

MSB	LSB
0 0 0 0 0 0 0 1	1 1 1 1 0 0 1 0
1_{10}	242_{10}

242_{10} is written to register 73
1 is written to register 75

B. Load the post-arm reading count registers. Write the decimal equivalent of the least significant byte to register 77. Write the decimal equivalent of the middle significant byte to register 79. Write the decimal equivalent of the most significant byte to register 7B. You can set the post-arm reading count from 7 to 16,777,215 readings. For example, to program 100,000 post-arm readings with no pre-arm readings, a post-arm reading count of 99,997 (100,000 - 3) is loaded into the post-arm registers. (A count of 1 is written to the pre-arm registers.)

MSB	MIDSB	LSB
0 0 0 0 0 0 0 1	1 0 0 0 0 1 1 0	1 0 0 1 1 1 0 1
1_{10}	134_{10}	157_{10}

157_{10} is written to register 77
134_{10} is written to register 79
1_{10} is written to register 7B

3. Initialize the sample rate registers.

A. Write the decimal value of 129 to register base $+63_{16}$, and write the decimal value of 255 to register base $+61_{16}$.

B. Disable reference divider reclocking (for divider values greater than 100,000 - see Step 4). Set arm control register (base $+4B_{16}$) bit 4 to '0' while retaining the settings of the other bits.

4. Set the sample rate.

A. If the trigger source is not reference period / N (base + $4D_{16}$ bits 4 - 2 =1 0 0), the decade division register and binary division register must be set as follows:

write a value of 1 to base + 61_{16}
write a value of 132 to base + 63_{16}

B. If the trigger source is the reference period / N (base + $4D_{16}$ bits 4 - 2 =1 0 0), then to set the sample rate (period) you must know the rate you want and the reference frequency. From these values, a value (N) is determined as shown below.

N = reference frequency / required sample rate

The reference frequency is then divided by N (Ref/N) to obtain the sample rate. N is represented by the decade division register (61_{16}) and the binary division register (63_{16}).

For example, to program a sample rate of 1 kHz (1 ms period) using the digitizer's internal 20 MHz reference:

N = 20 MHz / 1 kHz = 20,000

16_{10} is written to the decade division register (base + 61_{16}) to set a division by 10,000. All other bits are set to 0. 132_{10} is written to the binary division register (base + 63_{16}) to set a division by 2. (Bits 7 - 4 must always be set to 1 0 0 0.) These settings divide the reference frequency by 20,000 which gives the required sample rate of 1 kHz.

5. Set the digitizer A/D converter as the high-speed clock source and data source for memory.

A. Using the traffic register (base +02_{16}), set the pulse register as the high-speed clock source by setting bits 3 - 2 to '0 0'. Retain the settings of the other bits.

B. Send a clock pulse to the internal high-speed bus by reading or writing (any value) to the pulse register (base +08_{16}).

C. Using the traffic register (base +02_{16}), set the A/D as the high-speed clock source and as the high-speed data source by setting bits 3 - 0 to '0 1 0 0'. Retain the settings of the other bits.

6. Load the arm count and send the initiate pulse.

A. Load the arm count and initialize the trigger counters by writing (any value) to the arm count latch register (base + 59_{16}) **three times**.

B. Initiate the timebase processor by writing any value to the timebase initiation register (base $+45_{16}$).

Note The memory address registers (base $+23_{16}$ to base $+27_{16}$) are useful for determining if the digitizer has been re-initiated by indicating the number of readings that have been taken. For example, if 100 readings are to be taken when the digitizer is re-initiated, the memory address registers point to the location where the 100th reading is to be stored.

Retrieving Data from Memory

This section explains how to use register reads/writes to retrieve readings from memory and transfer them over the VME (VXI data transfer) bus.

The procedure given in this section for reading data from memory assumes that data is stored under the following digitizer configuration restrictions:

- a single burst of post-arm readings

Initializing Digitizer Memory to Retrieve Data

As with storing data in memory, digitizer memory must be initialized before data is retrieved from memory. The registers used in the data retrieval process are:

- traffic register
  base + 02_{16}
- pulse register
  base + 08_{16}
- memory control register
  base + 21_{16}
- terminal address register
  base + $2B_{16}$
- base address registers
  base + $2D_{16}$ and base + $2F_{16}$
- data register
  base + $0C_{16}$

Procedure

1. Place the last reading (from the A/D) into memory.

   A. Using the traffic register (base + 02_{16}) set the pulse register as the high-speed clock source by setting bits 3 - 2 to 0 0. Retain the settings of the other bits.

   B. Send a clock pulse to the internal high-speed bus by reading or writing (any value) to the pulse register (base + 08_{16}).

2. Initialize the memory control register.

   A. Set bits 2 - 0 of the memory control register (base + 21_{16}) to '0'. Retain the settings of bits 7 - 3.

3. Enable memory.

   A. Enable data to be retrieved from digitizer memory by setting the memory control register (base + 21_{16}) as indicated. Retain the settings of the other bits.

   bits 2 - 0 = 1 1 0

   B. Using the traffic register (base + 02_{16}), set digitizer memory as the data source for the internal high-speed bus, and set the pulse register as the high-speed clock source by setting the following traffic register bits as indicated:

   bits 3 - 2 = 0 0(clock source is pulse register)
   bits 1 - 0 = 1 1(memory is internal bus data source)

   (The internal high-speed bus links digitizer memory to the VME bus which is accessed by the user (Figure 3-7) .)

   C. Send a clock pulse to the internal high-speed bus by reading or writing (any value) to the pulse register (base + 08_{16}).

4. Determine the starting address of the memory segment.

   A. The segment size (which is the number of readings that were taken) must be divisible by 4. The starting address is computed as:

   starting address = ending address - (segment size - 1)

   The data retrieval procedure is for one segment of post-arm readings. Therefore, the ending address (524,287) is the size of digitizer memory.

5. Set the terminal (ending) address.

A. The terminal address is the ending address of the memory segment. This address is computed as:

terminal address = (ending address - 4095) / 2048

Again, the procedure is for one segment of post-arm readings. Therefore, the terminal address is:

terminal address = (524,287 - 4095) / 2048 = 254

B. Divide the starting address (see Step 4) by 4. If this value is > 65,535 ($FFFF_{16}$), add 1 to the terminal address value (254).

C. Write the terminal address to the terminal address register (base + $2B_{16}$).

6. Set the base (starting) address.

A. The base address is the (starting address / 4) of the memory segment. Write the most significant byte of (starting address / 4) to the base 0 address register (base + $2D_{16}$). Write the least significant byte of starting address / 4 to the base 1 address register (base + $2F_{16}$).

For example, if the segment size is 10,000 readings, the terminal and base addresses would be:

start address = 524,287 - 9,999 = 514,288

514,288 / 4 = 128,572 = $1F63C_{16}$

terminal address = 254 + 1 (since 128,572 > 65,536)

base address = F63C (the '1' is part of the terminal address: 254 + 1)

MSB	LSB
1 1 1 1 0 1 1 0	0 0 1 1 1 1 0 0
246_{10}	60_{10}

246_{10} is written to register base + $2D_{16}$
60_{10} is written to register base + $2F_{16}$

7. Send three clock pulses to the internal high-speed bus by reading or writing (any value) to the pulse register (base + 08_{16}) three times. This initializes the transfer stages.

8. Set the clock source to the digitizer data register.

A. In the traffic register (base + 02_{16}), set the digitizer's data register as the high-speed clock source by setting bits 3 - 2 as indicated:

bits 3 - 2 = 1 0

Setting the data register (base + $0C_{16}$) as the clock source transfers a reading from memory to the VME bus each time the data register is read.

Example Program

The following program demonstrates the procedures used to register program the digitizer. The program was developed using the configuration listed on page C-1. To adapt the program for use with an embedded controller, you will need to change the A24 base address accordingly, as well as make the modifications noted in the program listing.

This program accomplishes the following:

- SCPI programming
  - configures the digitizer to take 20 readings on the 5V range using the CONFigure command, and then retrieves the readings using the READ? command.
- Register programming
  - changes the measurement range to 1V
  - changes the trigger source to the reference oscillator period/N
  - sets the post-arm reading count to 20 readings
  - changes the sample rate to 10 kHz (100 µs)
  - re-initiates the digitizer to take the next 20 readings on the 1V range
  - retrieves the readings from memory by reading the data register. This places the readings on the VME(VXI data transfer) bus.

All other digitizer parameters (e.g. arm source, arm count) remain as set by the previous CONFigure command.

Note The following program contains a C function which re-initiates the digitizer (void initiate (long base_addr)). This function shows the **exact** sequence the registers must be written to in order to successfully re-initate the digitizer. When modifying this program, be sure that the sequence for re-initiating the digitizer remains the same.

REG_PROG.C

```
/* REG_PROG.C - This program configures the digitizer using register reads */
/* and writes. The program sets an initial digitizer configuration using */
/* the CONFigure command and takes a set of readings using the READ? */
/* command. Register reads and writes are then used to change the measurement */
/* range, change the trigger source, change the post-arm reading count, re-initiate */
/* the digitizer, and retrieve the readings. */


/* Include the following header files */

#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>
#include   <malloc.h>
#include   <cfunc.h>            /* This file is from the HP-IB Command Library */

#define ADDR 70905L             /* I/O path between the digitizer and PC */
#define CMD_MOD 70900L     /* I/O path between the PC and the Command Module */

/* Function prototypes */

long get_base_addr(void);
void rst_clr(void);
void conf_read(void);
void input_config(long base_addr);
void set_trig_source(long base_addr);
void initialize(long base_addr);
void initiate(long base_addr);
void memory_retrieve(long base_addr);
void data_read(long base_addr);
void check_error(char *function);

/*****************************************************************************/
void main(void)
{
      long base_addr;           /* variable for digitizer A24 base address */

      rst_clr();                          /* reset and clear the digitizer */
      base_addr = get_base_addr();        /* function call to calculate and */
                                          /* return digitizer A24 base address */
      conf_read();                        /* function call to take first set of readings */
      input_config(base_addr);             /* function call to set the 1V measurement range */
      set_trig_source(base_addr);         /* function call which sets the trigger source */
      initialize(base_addr);
```

Continued on Next Page

```
        initiate(base_addr);              /* function call which prepares memory to store readings */
        memory_retrieve(base_addr);  /* function call which prepares memory to retrieve readings */
        data_read(base_addr);             /* function call which reads the data register */

}

/*****************************************************************************/
void conf_read(void)
{
/* This function uses the CONFigure command to set up 20 readings on the */
/* 5V range. The READ? command is used to take the readings. */

        char go;
        int    readings = 20, i = 0;
        float *rdgs, *readcnt;

        /* dynamically allocate memory for readings */

        rdgs = malloc(20 * sizeof(float));
        readcnt = rdgs;

        /* configure the digitizer for 20 readings, 5V range, on input */
        /* port 3 */

        IOOUTPUTS(ADDR, "CONF1:ARR:VOLT (20),5,(@3)", 26);

        /* check for errors in the CONFigure command */

        check_error("conf_read");

        /* initiate the digitizer and read (fetch) the readings */

        IOOUTPUTS(ADDR, "READ?", 5);

        IOENTERA(ADDR, rdgs, &readings);

           for (i = 0; i < readings; i++)
           {
                   printf("\nReading %d = %f", i, *readcnt++);
           }
        free(rdgs);

        printf("\nPress Enter (return) to change configuration with register reads and writes");
        scanf("%c", go);

}
```

Continued on Next Page

```
/*****************************************************************************/
void input_config(long base_addr)
{
/* This function changes the measurement range from 5V as set by the CONFigure */
/* command, to 1V using the A/D shift register. The range is changed by */
/* enabling/disabling various attenuators on the input signal path. */

        int    shift_count = 55, bit_set = 0;
        float bit = 0;
        char stat_read[80], stat_write[80], strobe_write[80];

        /* create DIAG:PEEK? command which reads the A/D status register */
        sprintf(stat_read, "DIAG:PEEK? %ld, %d", base_addr+0x03,8);

        /* Set channel 1 range to 1V by setting bits 38, 37, 35, 34, and 10 */
        /* as required in the A/D shift register. */

        for(shift_count = 55; shift_count >= 0; shift_count--)
           {
           switch (shift_count)
           {
           /* turn channel 1 14 dB internal attenuator off */

           /* read bit 38 by reading the A/D status register */
           case 38:IOOUTPUTS(CMD_MOD, stat_read, strlen(stat_read));
                   IOENTER(CMD_MOD, &bit);

           /* set bit 38 to '1' by writing to the A/D serial register */
                   sprintf(stat_write,"DIAG:POKE %ld, %d, %d", base_addr+0x05,8,1);
                   IOOUTPUTS(CMD_MOD, stat_write, strlen(stat_write));
                   break;

           /* turn channel 1 6 dB internal attenuator off */

           /* read bit 37 by reading the A/D status register */
           case 37:IOOUTPUTS(CMD_MOD, stat_read, strlen(stat_read));
                   IOENTER(CMD_MOD, &bit);

           /* set bit 37 to '1' by writing to the A/D serial register */
                   sprintf(stat_write,"DIAG:POKE %ld, %d, %d", base_addr+0x05,8,1);
                   IOOUTPUTS(CMD_MOD, stat_write, strlen(stat_write));
                   break;
```

Continued on Next Page

```
        /* turn channel 1 20 dB post attenuator on */

        /* read bit 35 by reading the A/D status register */
        case 35:IOOUTPUTS(CMD_MOD, stat_read, strlen(stat_read));
                IOENTER(CMD_MOD, &bit);

        /* set bit 35 to '0' by writing to the A/D serial register */
                sprintf(stat_write,"DIAG:POKE %ld, %d, %d", base_addr+0x05,8,0);
                IOOUTPUTS(CMD_MOD, stat_write, strlen(stat_write));
                break;

        /* turn channel 1 20 dB input attenuator off */

          /* read bit 34 by reading the A/D status register */
        case 34:IOOUTPUTS(CMD_MOD, stat_read, strlen(stat_read));
                IOENTER(CMD_MOD, &bit);

         /* set bit 34 to '1' by writing to the A/D serial register */
                sprintf(stat_write,"DIAG:POKE %ld, %d, %d", base_addr+0x05,8,1);
                IOOUTPUTS(CMD_MOD, stat_write, strlen(stat_write));
                break;

        /* route input signal to channel 1 attenuators */

        /* read bit 10 by reading the A/D status register */
        case 10:IOOUTPUTS(CMD_MOD, stat_read, strlen(stat_read));
                IOENTER(CMD_MOD, &bit);

        /* set bit 10 to '1' by writing to the A/D serial register */
                sprintf(stat_write,"DIAG:POKE %ld, %d, %d", base_addr+0x05,8,1);
                IOOUTPUTS(CMD_MOD, stat_write, strlen(stat_write));
                break;

        /* read and shift all other shift register bits to restore each bit */
        /* to its original position */

        default:IOOUTPUTS(CMD_MOD, stat_read, strlen(stat_read));
                IOENTER(CMD_MOD, &bit);
                bit_set = (int)(bit + ((bit >= 0) ? .5 : -.5));
                sprintf(stat_write,"DIAG:POKE %ld, %d, %d", base_addr+0x05,8,bit_set);
                IOOUTPUTS(CMD_MOD, stat_write, strlen(stat_write));
                break;
        }

    }
```

Continued on Next Page

```
        /* copy the shift register bits to the shift register latch by */
        /* writing a value of 4 to the strobe register */
        sprintf(strobe_write,"DIAG:POKE %ld, %d, %d", base_addr+0x0B,8,4);
        IOOUTPUTS(CMD_MOD, strobe_write, strlen(strobe_write));

}

/*****************************************************************************/
void set_trig_source(long base_addr)
{
/* This function sets the digitizer trigger source to the reference period / n. */
/* The reference source is the digitizer's internal 20 MHz oscillator which */
/* was set by the CONFigure command. */

        char command[80];
        int    bit_reg = 0;
        float bit_pat = 0;

        /* read trigger source register  */

        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x4D,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);
        /* retain register settings, set trigger source to reference period / n */
        bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xE3) | 0x10;
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x4D,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

}

/*****************************************************************************/
void initialize(long base_addr)
{
/* This function initializes digitizer memory. This includes initializing the */
/* memory control register, specifying the data storage locations in memory, */
/* and enabling data to be written to memory. Note that the registers must be */
/* read and written to in the sequence shown. */

        char command[80];
        int    bit_reg = 0;
        float bit_pat = 0;
```

Continued on Next Page

```
        /* initialize the memory control register by setting bits 2 - 0 to '0'*/

        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x21,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);
        /* retain register settings, set memory control register bits 2 - 0 */
        /* to '0' */
        bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xF8);
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x21,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* write the terminal (ending) address to the terminal address register */

        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x2B,8,255);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* set the base (starting) address - most significant byte */

        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x2D,8,255);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* set the base (starting) address - least significant byte */

        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x2F,8,251);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* enable memory to be written to and enable the address counter */
        /* by setting bits 2 - 0 to '1 0 1'. */

        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x21,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);
        /* retain register settings, set memory control register bits 2 and 0 */
        /* to '1' */
       bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xF8) | 0x05;
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x21,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

}
```

Continued on Next Page

```
/*****************************************************************************/
void initiate(long base_addr)
{
/* This function initializes and initiates the timebase processor. This includes */
/* initializing the processor, setting the trigger (post-arm) count, setting */
/* the sample rate, setting the A/D as the data source for memory, and initiating */
/* the timebase processor. Note that the registers must be read and written to */
/* in the sequence shown. */

        char command[80];
        float bit_pat = 0;
        int    *addr_ptr, *data_ptr, bit_reg = 0;
        int    tb_addr [] = {0x41,0x5F, 0x65, 0x67, 0x69, 0x6B, 0x7D, 0x7F, -1};
        int    tb_data [] = {0, 1, 1, 5, 0, 0, 0, 0};

        data_ptr = tb_data;

        /* reset the timebase processor by writing the values to the corresponding */
        /* registers shown above */

        for (addr_ptr = tb_addr; *addr_ptr != -1; addr_ptr++)
        {
           sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr + *addr_ptr, 8, *(data_ptr++));
           IOOUTPUTS(CMD_MOD, command, strlen(command));
        }

        /* since only post-arm readings are taken, set the pre-arm reading count */
        /* to 1 */

        /* write least significant byte of pre-arm count */
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x73,8,1);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* write most-significant-byte of pre-arm count */
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x75,8,0);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* set the trigger count (post-arm readings only) by loading the post-arm */
        /* reading count registers. This program sets up 20 post-arm readings. */

        /* write least-significant-byte of post-arm count (count - 3) */
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x77,8,17);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
```

Continued on Next Page

```
/* write middle-significant-byte of post-arm count */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x79,8,0);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* write most-significant-byte of post-arm count */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x7B,8,0);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* initialize the sample rate registers */

sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x63,8,129);
IOOUTPUTS(CMD_MOD, command, strlen(command));

sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x61,8,255);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* disable reclocking of the reference divider (enabled for reference */
/* divider values of 100,000 or greater) */

sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x4B,8);
IOOUTPUTS(CMD_MOD, command, strlen(command));
IOENTER(CMD_MOD, &bit_pat);

/* retain register settings, disable reclocking */
bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
bit_reg = (bit_reg & 0xEF);
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x4B,8,bit_reg);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* set the sample rate to 10 kHz (100 us) */

/* set decade division register for a division by 1,000 */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x61,8,8);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* set binary division register for a division by 2 */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x63,8,132);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* For systems using an embedded controller, it may be necessary to monitor */
/* bit 1 of the arm status register (base + 43) until it is cleared (set to '0') */
/* before continuing with the next set of instructions. */

/* using the traffic register, set the pulse register as the high-speed */
/* clock source, write to the pulse register to remove any "old" readings */
/* from the data bus */
```

Continued on Next Page

```
        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x02,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);

        /* retain register settings, set pulse register as the high-speed */
        /* clock source */
        bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xF3);
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x02,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* Send a clock pulse to the internal high-speed bus */
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x08,8,0);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* using the traffic register, set the digitizer A/D as the clock */
        /* source and as the data source for memory */

        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x02,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);

        /* retain register settings, set the A/D as the high-speed */
        /* clock source and as the data source */
        bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xF0) | 0x04;
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x02,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* initiate the timebase processor */

        /* load the arm count and initialize the trigger counters */
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x59,8,0);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* send the timebase processor initiate pulse */
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x45,8,0);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

}
```

Continued on Next Page

```
/*****************************************************************************/
void memory_retrieve(long base_addr)
{
/* This function sets the address locations of the readings in digitizer memory, */
/* and enables the readings to be retrieved using the digitizer's data register. */

        char command[80];
         int    bit_reg = 0;
         float bit_pat = 0;

         /* using the traffic register set the pulse register as the high-speed */
         /* clock source, the A/D is still the data source */

         sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x02,8);
         IOOUTPUTS(CMD_MOD, command, strlen(command));
         IOENTER(CMD_MOD, &bit_pat);

         /* retain register settings, set pulse register as the high-speed */
         /* clock source */
         bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
         bit_reg = (bit_reg & 0xF3);
         sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x02,8,bit_reg);
         IOOUTPUTS(CMD_MOD, command, strlen(command));

         /* send a clock pulse to the internal high-speed bus to place the last */
         /* A/D reading into memory */
         sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x08,8,0);
         IOOUTPUTS(CMD_MOD, command, strlen(command));

         /* initialize the memory control register by setting bits 2 - 0 to '0 0 0' */

         sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x21,8);
         IOOUTPUTS(CMD_MOD, command, strlen(command));
         IOENTER(CMD_MOD, &bit_pat);

         /* retain register settings, set memory control register bits 2 - 0 */
         /* to '0' */
         bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
         bit_reg = (bit_reg & 0xF8);
         sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x21,8,bit_reg);
         IOOUTPUTS(CMD_MOD, command, strlen(command));
```

Continued on Next Page

```
/* enable data to be read from memory and enable the address counter */

sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x21,8);
IOOUTPUTS(CMD_MOD, command, strlen(command));
IOENTER(CMD_MOD, &bit_pat);

/* retain register settings, set memory control register bits 2 - 0 */
/* to '1 1 0' */
bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
bit_reg = (bit_reg & 0xF8) | 0x06;
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x21,8,bit_reg);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* set digitizer memory as the high-speed bus data source and the */
/* pulse register as the clock source */

sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x02,8);
IOOUTPUTS(CMD_MOD, command, strlen(command));
IOENTER(CMD_MOD, &bit_pat);

/* retain register settings, set the data source and clock source */
bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
bit_reg = (bit_reg & 0xF0) | 0x03;
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x02,8,bit_reg);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* Send a clock pulse to the internal high-speed bus */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x08,8,0);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* write the terminal (ending) address to the terminal address register */

sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x2B,8,255);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* set the base (starting) address - most significant byte */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x2D,8,255);
IOOUTPUTS(CMD_MOD, command, strlen(command));

/* set the base (starting) address - least significant byte */
sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x2F,8,251);
IOOUTPUTS(CMD_MOD, command, strlen(command));
```

Continued on Next Page

```
        /* initialize the transfer stages by sending three clock pulses to the */
        /* internal high-speed bus */

        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x08,8,0);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOOUTPUTS(CMD_MOD, command, strlen(command));

        /* set the digitizer data register as the high-speed clock source */
        /* (digitizer memory is still the data source) */

        sprintf(command, "DIAG:PEEK? %ld, %d", base_addr+0x02,8);
        IOOUTPUTS(CMD_MOD, command, strlen(command));
        IOENTER(CMD_MOD, &bit_pat);

        /* retain register settings, set data register as the clock source */
        bit_reg = (int)(bit_pat + ((bit_pat >= 0) ? .5 : -.5));
        bit_reg = (bit_reg & 0xF3) | 0x0B;
        sprintf(command, "DIAG:POKE %ld, %d, %d", base_addr+0x02,8,bit_reg);
        IOOUTPUTS(CMD_MOD, command, strlen(command));

}

/*****************************************************************************/
void data_read(long base_addr)
{
/* This function retrieves the new set of readings (those taken on the 1V range) */
/* by reading the data register using the HP E1406 Command Module */
/* DIAGnostic:UPLoad:SADDress? command. Reading the data register places the */
/* readings on the VME (VXI data transfer) bus. */

        int    *rdgs, i = 0, swap = 0, bytes = 0, length = 1;
        char rd_mem[80], lf_remove[1];

        rdgs = malloc(20 * sizeof(int));

        swap = sizeof(int);
        bytes = 20 * swap;

        /* Create the (HP E1406 Command Module) command string which reads the data register */

        sprintf(rd_mem, "DIAG:UPL:SADD? %ld, %d", base_addr+0x0C,40);
```

Continued on Next Page

```
        /* Send the DIAG:UPL:SADD? command which accesses the data register */
        /* and retrieves the readings */

        IOOUTPUTS(CMD_MOD, rd_mem, strlen(rd_mem));  /* retrieve and enter the readings, */
        IOENTERAB(CMD_MOD, rdgs, &bytes, swap);          /* remove the block header */

        /* remove the line feed which trails the last data byte */
        IOENTERS(CMD_MOD, lf_remove, &length);

        /* convert and print each reading as a voltage */

        for (i = 0; i < 20; i++)
           {
           rdgs[i] /= 16;       /* remove label from each reading */
           if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
                  printf("\nReading overrange");
           else
                  printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0005));
           }

        free(rdgs);
}

/*****************************************************************************/
long get_base_addr(void)
{
/* This function returns the digitizer's A24 base address. */

        /* digitizer logical address */
        long logical_addr = (ADDR - 70900L) * 8;

       /* base address of (A24) offset register in A16 address space */
        long base_addr = (0x1FC000 + (logical_addr * 64)) + 6;

         float a24offst;        /* A24 offset from A16 offset register */

        char rd_addr[80];    /* command string variable */

           /* Create the command string which reads the A24 base address */
        sprintf(rd_addr, "DIAG:PEEK? %ld, %d", base_addr,16);

           /* Send DIAG:PEEK? command */
        IOOUTPUTS(CMD_MOD, rd_addr, strlen(rd_addr));

           /* Read value from offset register */
        IOENTER(CMD_MOD, &a24offst);
```

Continued on Next Page

```
        /* Multiply offset value by 256 for 24-bit address value */
    a24offst *= 256.;

    return (long)a24offst;
}

/*****************************************************************************/
void rst_clr(void)
{
    /* Reset and clear the digitizer */

    IOOUTPUTS(ADDR, "*RST;*CLS", 9);
}

/*****************************************************************************/
void check_error(char *func_tion)
{
    char    into[161];
    int     length = 160;

    IOOUTPUTS(ADDR, "SYST:ERR?", 9);    /* Query error register */
    IOENTERS(ADDR, into, &length);      /* Enter error message */

    if (atoi(into) != 0)    /* Determine if error is present */
                            /* If errors present, print and exit */
    {
        while (atoi(into) != 0)
        {
            printf("Error %s in function %s\n\n", into, func_tion);
            IOOUTPUTS(ADDR, "SYST:ERR?", 9);
            IOENTERS(ADDR, into, &length);
        }

        exit(1);
    }
}
```

Notes

Appendix D
Local Bus Interleaved Transfers

Appendix Contents

This appendix contains information on interleaved data transfers using the HP E1429B and the Local bus.

Interleaved Transfers

In an interleaved transfer, multiple digitizers transfer one reading or one set of readings (both channels) per block. The leftmost digitizer is set to the GENerate mode and the inner digitizer(s) is set to the INSert mode. A device such as the HP E1488 memory card is usually the consumer. Readings are taken directly from the digitizer A/Ds.

As an example, the transfer sequence from digitizers D_1 - D_3, where D_1 is in **generator** mode (leftmost slot) and D_2 and D_3 (rightmost slot) are in **insert** mode would appear as:

EOF EOB **D_1** EOB **D_2** EOB **D_3** ----->consumer

where EOF is the end-of-frame flag, EOB is the end-of-block flag, and D_n is either a two byte (one channel) or four byte (two channel) reading.

Note If you are using the interleaved transfer mode of the HP E1429B with the HP E1485 Digital Signal Processor module, note that the E1485 can accept no more than 256 blocks.

Maximum Data Transfer Rate

The maximum data transfer rate over the Local bus is 80 MBytes/second which is equivalent to 40 MSamples/second. The HP E1429B digitizer transfers data at 40 MSamples/second when readings are taken on two digitizer channels at the maximum sample rate of 20 MHz.

When doing interleaved transfers, the transfer rate cannot exceed 40 MSamples/second, regardless of the number of digitizers used. The maximum sample (transfer) rate allowed is determined by:

40 MSamples / number of channels

For example, if a configuration consists of two digitizers and both channels on each digitizer are used, the maximum sample rate for each digitizer is:

40 MSamples / 4 channels = 10 MHz

As another example, if a configuration consists of two digitizers and only three channels are used, the maximum sample rate for each digitizer is:

40 MSamples / 3 channels = 13.3 MHz

For the digitizer using two channels this would be 26.6 MSamples/ second since a single sample trigger causes both channels to sample. For the digitizer using only a single channel, the sample rate would be 13.3 MSamples/second. Together, the two digitizers are within the 40 MSample (80 MByte)/second Local bus transfer specification.

Note The maximum sample rates computed may not always be available using the digitizer's internal reference. In those instances, select a slower sample rate that is available from the internal reference, or use an external reference or an external trigger source.

Setting the Interleaved Transfer Mode

In addition to resetting the digitizer's Local bus chip, setting the Local bus mode, and setting the data source, the interleaved transfer mode must be set. This is done using the commands:

VINStrument:CONFigure:LBUS:SEND:POINts <count >
VINStrument:CONFigure:LBUS:POINts:AUTO <mode >

<count > is the number of readings per block. If readings are taken on only one channel, <count > is set to 1. If readings are taken on both channels, <count > is set to 2.

Setting <mode > to OFF sets the interleaved transfer mode. In this mode the end-of-block flag is sent after each reading or set of readings. The end-of-frame flag is sent with the GENerator's end-of-block flag.

These commands are sent as a single command string in order to prevent "Settings conflict" errors. Note that the readings per block and the interleaved transfer mode need only be specified when doing interleaved Local bus transfers.

Programming Procedure

The programming procedure for interleaved transfers is:

1. Use the CONFigure command and the low-level digitizer commands to configure the digitizers for the required measurements.

   All digitizers must have the same trigger source and the same sample rate. The digitizers must be synchronized such that readings are taken and the frame is transferred before the next sample occurs. The example program which follows shows how the digitizers are synchronized using one trigger source for both digitizers.

2. **Use the VINStrument subsystem to reset the leftmost digitizer's Local bus chip, to set the Local bus transfer mode to GENerate, to set the data source to the digitizer's A/D (CONVerter:CHANneln), and to set the interleaved transfer mode.**

   Use the VINStrument subsystem to reset the inner digitizer's Local bus chip, to set the Local bus transfer mode to INSert, to set the data source to the digitizer's A/D (CONVerter:CHANneln), and to set the interleaved transfer mode.

   VINStrument:CONFigure:LBUS:RESet
   VINStrument:CONFigure:LBUS:MODE <mode >
   VINStrument:CONFigure:LBUS:FEED <source >
   VINStrument:CONFigure:LBUS:SEND:POINts <count >
   VINStrument:CONFigure:LBUS:POINts:AUTO <mode >

   Only the INSert (and GENerate) mode is supported for interleaved transfers.

3. Reset the consumer's (i.e. memory card's) Local bus chip and configure the consumer to receive data.

4. Activate (initiate) the consumer.

5. Use INITiate:IMMediate to activate the leftmost (GENerator) digitizer.

6. Use INITiate:IMMediate to activate the inner (INSerter) digitizer.

7. Use INITiate:IMMediate to activate the rightmost (INSerter) digitizer.

8. Beginning with the leftmost (GENerator) digitizer, abort each digitizer before using the Local bus again.

Example Program

This program demonstrates how to use multiple digitizers to transfer readings, interleaved, to an HP E1488 memory card (consumer). The program takes 10 readings on both channels of two digitizers. Therefore, 10 frames of data (40 readings) are sent to the memory card. The example follows the programming procedure listed above.

LBUSINTR.C

```
/* LBUSINTR.C - This program demonstrates how to transfer interleaved readings */
/* from multiple digitizers to the HP E1488 memory card. In an interleaved transfer, */
/* each digitizer transfers one set of readings or one reading (if using a */
/* single channel) per block. The readings are taken directly from the A/Ds. */
/* The leftmost digitizer is set to the GENerate mode and the inner digitizer */
/* is set to the INSert mode. */

      /* Include the following header files */

#include   <stdio.h>
#include   <stdlib.h>
#include   <string.h>
#include   <malloc.h>
#include   <cfunc.h>                  /* This file is from the HP-IB Command Library Disk */

#define    ADDR_G 70905L              /* I/O path from the PC to the generator digitizer */
#define ADDR_I 70906L                 /* I/O path from the PC to the inserter digitizer */
#define ADDR_MEM 70903L               /* I/O path from the PC to the memory card */

/* Function Prototypes */

void rst_clr(long address);
void configure(void);
void initiate(void);
void check_error(char *func_tion, long address);
```

Continued on Next Page

```
/*****************************************************************************/
void main(void)                 /* run the program */
{
        clrscr();
        rst_clr(ADDR_G);        /* reset generator digitizer */
        rst_clr(ADDR_I);        /* reset inserter digitizer */
        rst_clr(ADDR_MEM);      /* reset memory card */
        configure();            /* configure the digitizers and the memory card */
        initiate();             /* initiate the digitizers and the memory card */
                                /* retrieve the readings from the memory card */
}
/*****************************************************************************/
void configure(void)
{
        int    length = 0, loop = 0;

        /* use the "digitizer1" array to configure the generator digitizer */

        char static *digitizer1[] =
        {"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                                /* channel 1, input port 3 */
         "CONF2:ARR:VOLT (10),5,(@4)",          /* configure channel 2, port 4 */
           "TRIG:STAR:SOUR ECLT0",                      /* set trigger source */
           "VINS:LBUS:RES",                     /* reset the Local bus chip */
           "VINS:LBUS:MODE GEN",                /* set Local bus mode to GENerate */
           "VINS:LBUS:FEED 'CONV:BOTH'",                /* set Local bus feed (A/Ds) */
           "VINS:LBUS:SEND:POIN 2; POIN:AUTO OFF"}; /* set number of readings per block */
                                                        /* send end-of-block and end-of frame */
                                                        /* after each reading */

        /* use the "digitizer2" array to configure the inserter digitizer */

        char static *digitizer2[] =
        {"CONF1:ARR:VOLT (10),5,(@3)",          /* set 10 readings, 5V range, */
                                                /* channel, 1 input port 3 */
         "CONF2:ARR:VOLT (10),5,(@4)",          /* configure channel 2, port 4 */
           "TRIG:STAR:SOUR TIM",                /* set trigger source */
           "TRIG:STAR:TIM 1E-6",                        /* set sample rate (1 MHz) */
           "OUTP:ECLT0:FEED 'TRIG'",                    /* feed trigger to generator from ECLT0 */
           "OUTP:ECLT0:STAT ON",                /* enable feed */
           "VINS:LBUS:RES",                     /* reset the Local bus chip */
           "VINS:LBUS:MODE INS",                /* set Local bus mode to INSert */
           "VINS:LBUS:FEED 'CONV:BOTH'",                /* set Local bus feed (A/Ds) */
           "VINS:LBUS:SEND:POIN 2; POIN:AUTO OFF"}; /* set number of readings per block */
                                                        /* send end-of-block and end-of frame */
                                                        /* after each reading */
```

Continued on Next Page

```
        /* use the "memory" array to configure the memory card */

        char static *memory[] =
        {"FORM:DATA PACK",          /* set packed data format */
         "TRAC:DEL:ALL",            /* delete all readings on memory card */
         "TRAC:DEF SET1, 80",       /* store readings (80 bytes) in "SET1" */
         "VINS:LBUS:RES",                   /* reset the Local bus chip */
         "VINS:LBUS:MODE CONS",     /* set Local bus mode to consume */
         "STAT:OPC:INIT OFF"};      /* execute *OPC? after INIT is parsed */

        /* Execute each command in "digitizer1" using a loop */

       length = (sizeof(digitizer1) / sizeof(char*));

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR_G, digitizer1[loop], strlen(digitizer1[loop]));
           }

        /* Execute each command in "digitizer2" using a loop */

       length = (sizeof(digitizer2) / sizeof(char*));

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR_I, digitizer2[loop], strlen(digitizer2[loop]));
           }

        /* Execute each command in "memory" */

        length = (sizeof(memory) / sizeof(char*));

        for (loop = 0; loop < length; loop++)
           {
           IOOUTPUTS(ADDR_MEM, memory[loop], strlen(memory[loop]));
           }

        /* check for digitizer and memory card configuration errors */

        check_error("digitizer1", ADDR_G);

        check_error("digitizer2", ADDR_I);

        check_error("memory", ADDR_MEM);
}
```

Continued on Next Page

```
/*****************************************************************************/
void initiate(void)
{
        int    i = 0, readings = 40, swap = 0, bytes = 0, length = 1, *rdgs;
        float rdy;
        char lf_remove[1];


        /* dynamically allocate memory for readings */

        rdgs = malloc(40 * sizeof(float));          /* allocate computer memory for reading storage */
        swap = sizeof(int);                         /* each reading in memory is two bytes */
        bytes = 40 * swap;                          /* read 80 bytes */

        IOOUTPUTS(ADDR_MEM, "INIT", 4);             /* initiate the memory card */
        IOOUTPUTS(ADDR_MEM, "*OPC?", 5);    /* wait for INIT to parse before continuing */
        IOENTER(ADDR_MEM, &rdy);                    /* enter *OPC? response from memory card */

        IOOUTPUTS(ADDR_G, "INIT", 4);               /* initiate the generator digitizer */

        IOOUTPUTS(ADDR_I, "INIT", 4);               /* initiate the inserter digitizer */

        IOOUTPUTS(ADDR_G, "*OPC?", 5);              /* wait for generator digitizer readings */
        IOENTER(ADDR_G, &rdy);                      /* to complete */

        IOOUTPUTS(ADDR_MEM, "TRAC:DATA? SET1", 15); /* retrieve readings from memory card
*/
        IOENTERAB(ADDR_MEM, rdgs, &bytes, swap); /* enter readings and remove block header */

        /* remove line feed which trails the last data byte */

        IOENTERS(ADDR_MEM, lf_remove, &length);

        IOOUTPUTS(ADDR_G, "ABOR", 4);               /* abort generator digitizer */
        IOOUTPUTS(ADDR_I, "ABOR", 4);               /* abort inserter digitizer */

        /* convert and display the readings; the reading sequence is: */
        /* inserter digitizer channel 2 reading 1 */
        /* inserter digitizer channel 1 reading 1 */
        /* generator digitizer channel 2 reading 1 */
        /* generator digitizer channel 1 reading 1 */
        /* inserter digitizer channel 2 reading 2 and so on */

        for (i = 0; i < readings; i++)
        {
           rdgs[i] /= 16;
           if (rdgs[i] >= 2047 || rdgs[i] <= -2046)
                   printf("\nReading overrange");
```

```
            else
                printf("\nReading %d = %.6E", i, (rdgs[i] * 0.0025));

        }

        free(rdgs);
}

/*****************************************************************************/
void rst_clr(long address)
{
            /* Reset and clear the instruments */

        IOOUTPUTS(address, "*RST;*CLS", 9);

}

/*****************************************************************************/
void check_error(char *array, long address)
{
        char      into[161];
        int length = 160;

        IOOUTPUTS(address, "SYST:ERR?", 9);          /* Query error register */
        IOENTERS(address, into, &length);            /* Enter error message */

        if (atoi(into) != 0)                         /* Determine if error is present */
                                                     /* If errors present, print    and exit */
        {
            while (atoi(into) != 0)
            {
                printf("Error %s in %s\n\n", into, array);
                length =160;
                IOOUTPUTS(address, "SYST:ERR?", 9);
                IOENTERS(address, into, &length);
            }

            exit(1);
        }
}
```

Comments

1. **GENerator Digitizer Configuration.** Both channels of the GENerator digitizer are CONFigured for 10 readings on the 5V range. Thus, each block of data generated is four bytes (two bytes/reading), followed by the end-of-block (EOB) and end-of-frame (EOF) flags.

The GENerator digitizer's trigger source is set to ECLT0. This VXI backplane trigger line is controlled by the INSerter digitizer which feeds its (internal) trigger signal to the GENerator digitizer. This causes both digitizers to sample and transfer readings at approximately the same rate - which is required for interleaved transfers.

Before setting the digitizer's Local bus configuration, the Local bus chip is reset. Next, the Local bus mode is set to GENerate and the feed (data source) is set to CONVerter:BOTH. Finally, the interleaved transfer mode is set. The number of readings per block is set to 2, and the end-of-block and end-of-frame flags are sent after each set of readings. These settings are sent as a single command string in order to prevent "Settings conflict" errors.

2. **INSerter Digitizer Configuration.** Like the GENerator digitizer, both channels of the INSerter digitizer are CONFigured for 10 readings on the 5V range. Each block of data generated is four bytes (two bytes/reading), followed by the end-of-block (EOB) flag.

The INSerter digitizer's trigger source is its internal trigger. This trigger is transferred to the GENerator digitizer over the VXI backplane ECLT0 trigger line. In this program the INSerter digitizer's sample rate is set to 1 µs, thus, both digitizers sample and transfer readings at approximately a 1MHz rate.

For all digitizers in the interleaved transfer configuation, the Local bus chip must be reset first. Next, the Local bus mode is set to INSert and the feed (data source) is set to CONVerter:BOTH. Finally, the interleaved transfer mode is set.

3. **Digitizer Trigger Sources and Sample Rates.** When doing interleaved transfers over the Local bus, all digitizers must have the same trigger source and the same sample rate. The maximum Local bus transfer rate is 80 MBytes/second which is equivalent to 40 MSamples/ second. The transfer rate cannot exceed 40 MSamples/second regardless of the number of digitizers.

4. **Arming the Digitizers.** In order for all digitizers to sample when a trigger is received, the digitizers to the left of the rightmost digitizer are armed (initiated) before the rightmost digitizer is triggered.

5. **Aborting the Digitizers.** Following interleaved transfers, the Local bus chip of each digitizer is left in an active (running) state. Starting with the leftmost (GENerator) digitizer, it is necessary to ABORt each digitizer before the next use of the Local bus.

6. **Reading Sequence and Format.** When this program executes, the readings are transferred to the memory card (and later displayed) in the following sequence:

INSerter digitizer channel 2 reading 1
INSerter digitizer channel 1 reading 1
GENerator digitizer channel 2 reading 1
GENerator digitizer channel 1 reading 1
.
.
.
INSerter digitizer channel 2 reading n
INSerter digitizer channel 1 reading n
GENerator digitizer channel 2 reading n
GENerator digitizer channel 1 reading n

The memory card was set up to store the readings in the digitizer's packed data format. The packed readings are signed, 16-bit numbers preceded by the ANSI/IEEE Standard 488.2-1987 Definite Length Arbitrary Block header. Packed readings are always numbers between -1.0230 (-2046) and +1.0235 (2047). To convert the readings to voltages, each reading is divided by 16 to remove the data label bits (0 - 3), and is multipled by 0.0025 which is the reading resolution for the 5V range (refer to Chapter 3 for more information on data labels and packed reading conversions).

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W