Agilent E1437A

20 MSample/second ADC with Filters and FIFO

User's Guide

Agilent Technologies

Agilent Part Number E1437-90002

Printed in U.S.A
Print Date: March 2000, Third Edition

8600 Soper Hill Road Everett, Washington 98205-1209 U.S.A.

Agilent Technologies

NOTICE

The information contained in this document is subject to change without notice.

AGILENT TECHNOLOGIES, INC., MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Agilent Technologies shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

WARRANTY

A copy of the specific warranty terms applicable to your Agilent Technologies product and replacement parts can be obtained from your local Sales and Service Office.

Use of this manual is restricted to this product only.

TRADEMARKS

FibreXpress™ is a trademark of Systran Corporation.

Tachyon™ is a trademark of Agilent Technologies Inc..

RESTRICTED RIGHTS LEGEND

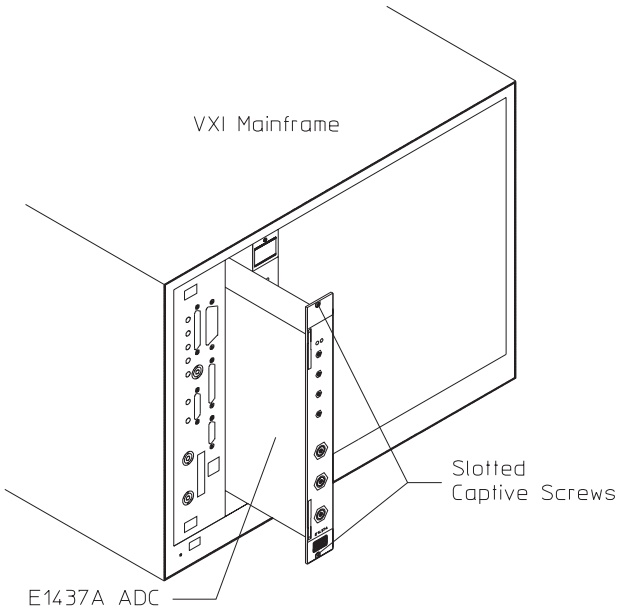
Use, duplication or disclosure by the U.S. Government is subject restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013

Agilent Technologies, Inc.
395 Page Mill Road
Palo Alto, CA 94303-0870 USA

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

The E1437A at a Glance

The E1437A 20 Msample/second Analog-to-Digital Converter with Filtering and Memory provides high precision digitizing for time and frequency domain applications along with signal conditioning, filtering, and memory. The module plugs into a single C-size slot in a VXI mainframe.



Number of Channels	1
Type of Input	50 ohm
Input Bandwidth	40 MHz, 8 MHz alias protected
Sample Rate	20 MSamp/sec
Voltage Range	20 mV to 10.24 Vpeak
Raw ADC resolution	23 bits
VXI Bus Support	VME and Local Bus
VXI Device type	Register/Message based
Size	C-sized, single slot

What you get with the E1437A

The following items are included with your E1437A

Hardware:

- E1437A ADC, C-size VXI module
- Software media:
  - MS-Windows® disks
  - HP-UX tape

Software:

❑ MS-Windows disks

- A setup program which installs:
  - The E1437A VXI*plug&play* libraries and drivers
  - The E1437A HP-VEE driver
  - Soft Front Panel program for the E1437A
  - Windows online help for the E1437A
  - HPDSP function library and online help
  - Example programs
  - Library and example program source files
  - Microsoft® Visual Basic header files

❑ HP-UX tape

- An installation utility which installs:
  - The E1437A C Interface libraries and drivers
  - Helpview online help for the E1437A
  - HPDSP function library and online help
  - E1485 C library binaries
  - Example programs
  - Library and example program source and make files

Documentation:

- E1437A User's Guide (this book)
- Online manual pages for Windows and HP-UX (Windows Help and Helpview Help formats)

In This Book

This book documents the E1437A module. It provides:

- installion information
- verification information
- operational information
- a programmer's reference
- circuit descriptions
- technical specifications

If you plan to use this module with the E1485A/B signal processing module and the 35635T Programmer's Toolkit you should also use the documentation for those products in order to form an application program development environment.

If you are using your E1437A module in the Windows 3.1®, Windows NT®, Windows 95®, or HP-UX environment the programmer's reference and other programming information are available as online help. The online help may be more convenient to use while programming. See the "Getting Started" chapter of this book for information on accessing the online help.

TABLE OF CONTENTS

1

Installing the E1437A

Installing the E1437A

This chapter contains instruction for installing the E1437A VXI ADC Module and its drivers. This chapter also includes instruction for transporting and storing the module.

To inspect the E1437A

The E1437A single channel VXI ADC Module was carefully inspected both mechanically and electrically before shipment. It should be free of marks or scratches and it should meet its published specifications upon receipt.

If the module was damaged in transit, do the following:

- Save all packing materials.
- File a claim with the carrier
- Call your Hewlett-Packard sales and service office.

To install the E1437A

Caution To protect circuits from static discharge, observe anit-static techniques whenever handling the E1437A VXI ADC Module

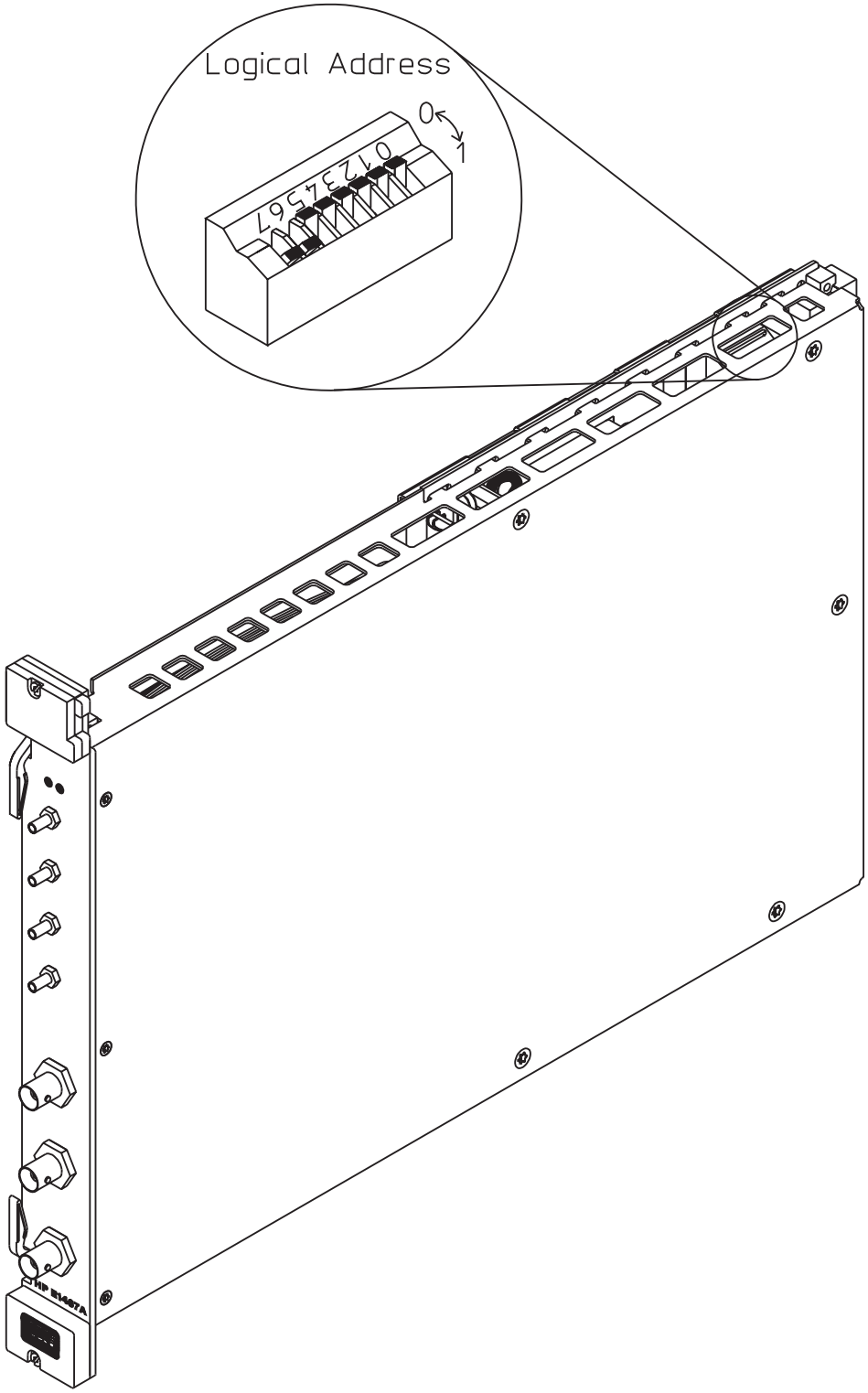
1 Set up your VXI mainframe. See the installation guide for your mainframe.

2 Select a slot in the VXI mainframe for the E1437A module

The E1437A module's local bus receives ECL-level data from the module immediately to its left and outputs ECL-level data to the module immediately to its right. Every module using the local bus is keyed to prevent two modules from fitting next to each other unless they are compatible. If you will be using the local bus, select adjacent slots immediately to the left of the data-receiving module. If the VXI bus is used, maximum data rates will be reduced but the module can be placed in any available slot.

3 Using a small screwdriver or similar tool, set the logical address configuration switch on the E1437A.

(See the illustration on the next page.) Each module in the system must have a unique logical address. The factory default setting is 1100 0000 (192). If an GPIB command module will be controlling the E1437A module, select an address that is a multiple of 8.


Logical Address
0
1
0 1 2 3 4 5 6 7
HP E1437A

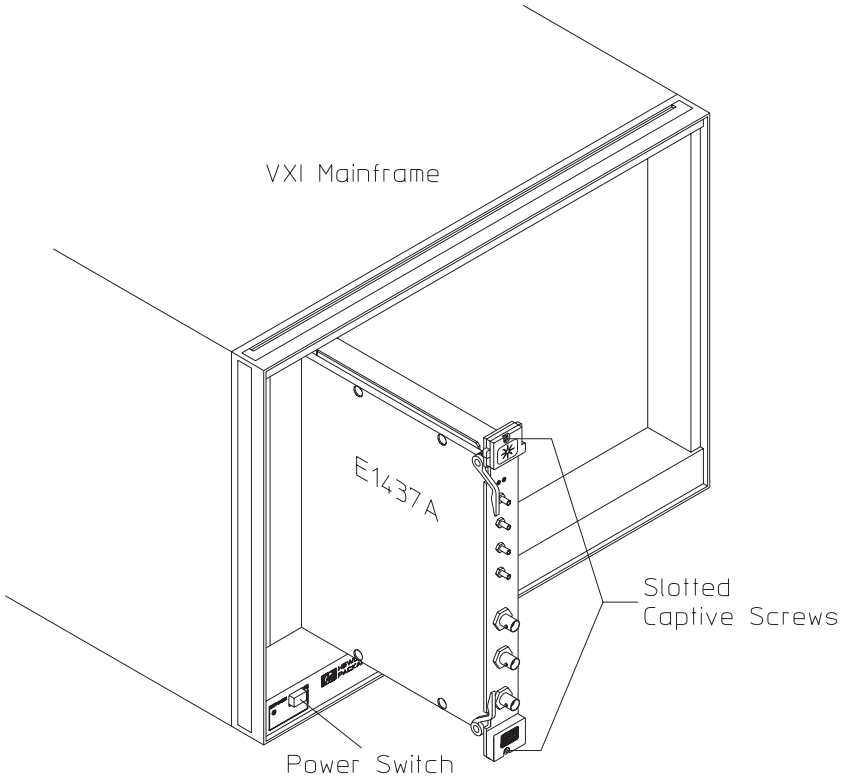
4 Set the mainframe's power switch to off (0).

Caution Installing or removing the module with power on may damage components in the module.

5 Place the module's card edges (top and bottom) into the module guides in the slot.

6 Slide the module into the mainframe until the module connects firmly with the backplane connectors. Make sure the module slides in straight.

7 Attach the module's front panel to the mainframe chassis using the module's captive mounting screws.



To store the module

Store the module in a clean, dry, and static free environment

For other requirements, see storage and transport restriction in "Technical Specifications".

To transport the module

- Package the module using the orignal factory packaging or packaging identical to the factory packaging.
- If returning the module to Hewlett-Packard for service, attach a tag describing the following:
  - Type of service required
  - Return address
  - Model number
  - Full serial number

  In any correspondence, refer to the module by model number and full serial number.
- Mark the container FRAGILE to ensure careful handling.
- *If necessary to package the module in a container other than original packaging, observe the following (use of other packaging is not recommended):*
  - Wrap the module in heavy paper of anti-static plastic.
  - Protect the front panel with cardboard.
  - Use a double-wall carton made of at least 350-pound test material.
  - Cushion the module to prevent damage.

Caution

Do not use styrene pellets in any shape as packing material for the module. The pellets do not adequately cushion the module and do not prevent the module from shifting in the carton. In addition, the pellets create static electricity which can damage electronic components.

2

Getting Started with the E1437A

Introduction

This chapter will help you to get your E1437A running and making simple measurements without programming. It shows you how to install the software libraries and how to run the Soft Front Panel program. It also introduces you to example programs.

Two versions of the Host Interface Library are available. One is the Windows 3.1, Windows 95, and Windows NT Library which communicates with the hardware using VISA (Virtual Instrument Software Architecture). VISA is the input-output standard upon which all the VIXplug&play software components are based. The second version is the HP-UX 9.x C-language Host Interface Library which uses SICL (the Standard Instrument Interface Library) to communicate with the E1437 hardware.

To Install the Programmer's Libraries

System Requirements (Microsoft Windows)

- An IBM-compatible personal computer.
- Microsoft Windows® 3.1, Microsoft Windows 95®, or Microsoft Windows NT®.
- The computer must have a 3 1/2 inch disk drive for the installation media.

System Requirements (HP-UX)

- One of the following workstations
  - An HP/Agilent V743 VXI-embedded workstation.
  - A stand-alone HP/Agilent Series 700 workstation with an E1489 EISA-to-MXIbus card and an E1483B VXI-MXI Bus Extender.
- The workstation must have a DAT drive for the installation media.
- HP-UX (version 9.x)
- HP SICL for HP-UX (version C.03.08a or later). The SICL product number is HP E2091C.

To install the Windows VXI*plug&play* drivers for the E1437A (for Windows 3.1, Windows 95 and Windows NT)

This procedure assumes that you have already installed a VISA (Virtual Instrument Software Architucture) library. If not, you can still install these drivers but you will receive an error message reminding you to install the VISA library.

1. Insert the disk labeled: "Agilent E1437A 20 MSample/sec A-to-D Converter"
2. Run the program: *drive*:\setup.exe

   Where *drive* represents the drive containing the setup disk.
3. Insert the second disk when prompted
4. The setup program asks you to confirm or change the directory path. The default directory path is recommended.
5. A dialog box asks if you want to install startup icons

   This creates a program group called "HPE1437" which includes:

   An icon to run the Soft Front Panel
   An icon for the E1437A Online Help file
   An icon for the HPDSP Online Help file
   An icon for UNINSTALL
   Several icons for example programs
   An icon for a readme file
6. A readme file may be displayed. If so, be sure to read it and follow the instructions.

To install the HP-UX C-language drivers for the E1437A (for HP-UX systems):

1 Log in as root.
2 Insert the "Agilent E1437A 20 MSample/sec A-to-D Converter" tape into the tape drive
3 To run the software installation utility interactively type:

```
/etc/update
```

See the HP-UX Reference manual for information on the update command.

Be sure to read the README file which contains important information on installation, viewing online help, and compiling example programs.

The Resource Manager

The Resource Manager is a program from your hardware interface manufacturer. It looks at the VXI mainframe to determine what modules are installed. You need to run it every time you power up. If you get the message: "No HP E1437A can be found in the system." then run the Resource Manager.

Before running the E1437A software make sure that your hardware is configured correctly and that the Resource Manager runs successfully. Before using your measurement system, you must set up all of its devices, including setting their addresses and local bus locations. No two devices can have the same address. Usually addresses 0 and 1 are taken by the Resource Manager and are not available.

For more information about the Resource Manager, see the documentation with your hardware interface.

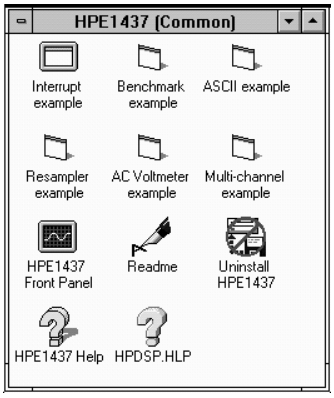
NOTE

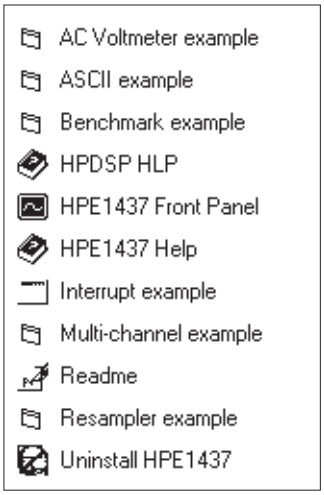
Most Resource Managers will recognize the manufacturer and model number of the E1437A but if your interface requires that you enter this information manually, use the following:

```
Manufacturer number: 4095 (Hex FFF)
Model number: 534 (Hex 216)
```

To Use the Program Group (Windows)

If you chose to install the program group during the installation procedure you will have an icon for a program group similar to one of the two below, depending on which Windows platform you use.



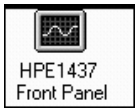


This program group contains icons which access the Soft Front Panel program, online help, and example programs. The following pages provide an overview of these items.

If you did not choose to install the program group, executable files for each of the items represented by group icons are available in the *drive*:\vxipnp directory and its subdirectories.

To Use the VXI*plug&play* Soft Front Panel (SPF)

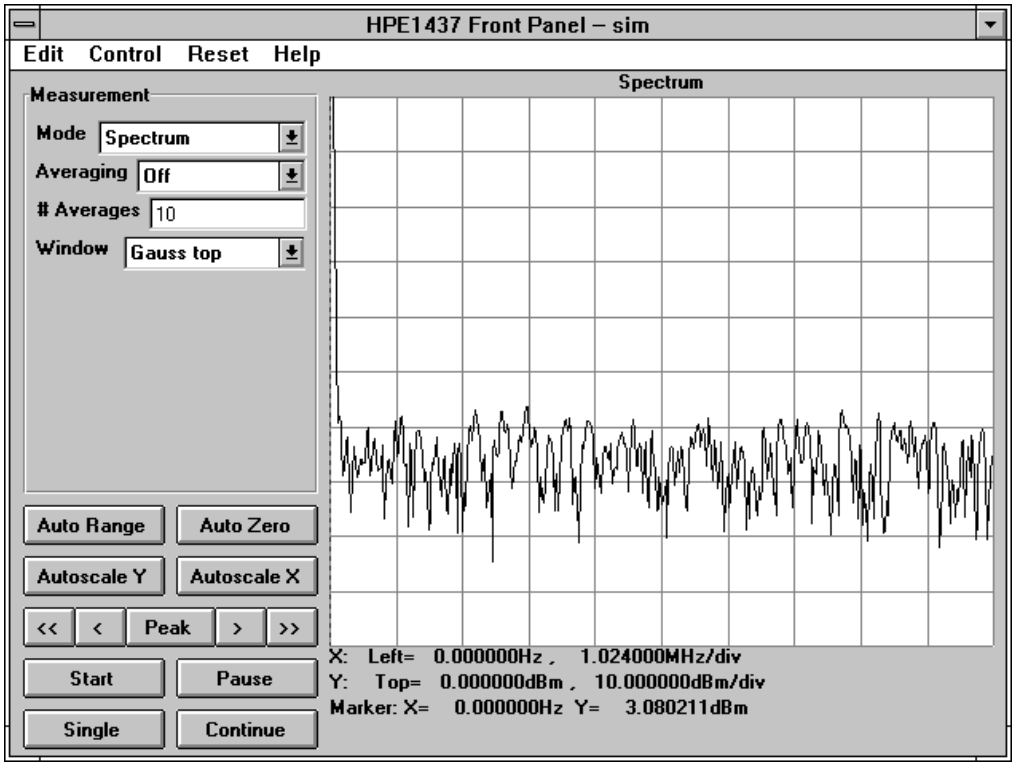
The the best place to start to explore the capabilities of the E1437A is with the Soft Front Panel (SFP). The Soft Front Panel can be useful for checking your system to make sure that is is installed correctly and that all of its parts are working. It can also be used to make actual measurements, since it accesses most of the E1437A's functionality.



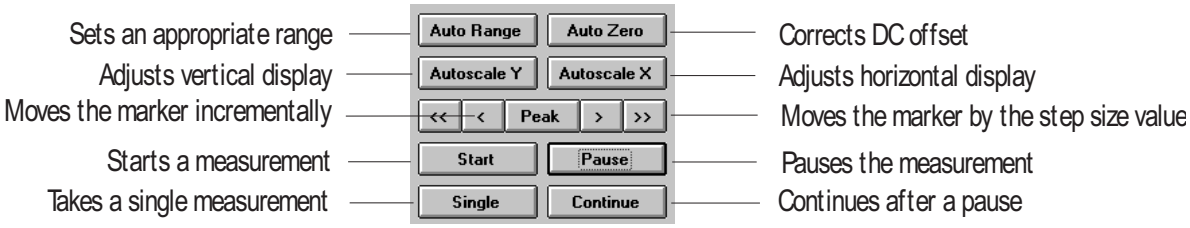
Select the E1437 Front Panel icon in your program group to start the SFP. This assumes you have already installed all required hardware and drivers (including VISA) and have run the configurator and Resource Manager required by your hardware interface.

When prompted for the resource descriptor, use the default "VXI::192" unless the logical address of the E1437 has been changed from its default setting of 192. If it has been changed then type the appropriate logical address instead of 192. Press OK.

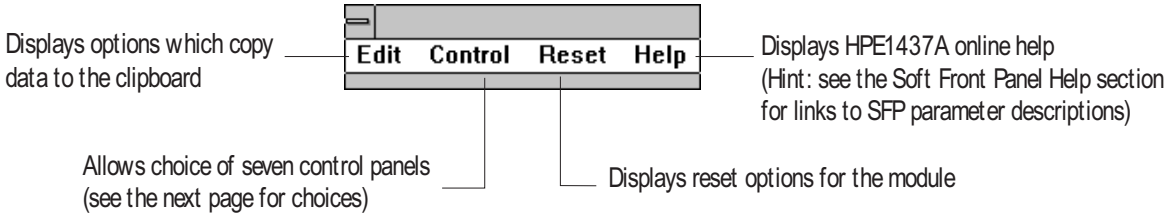
You can also run the SFP in a simulation mode without an E1437 by typing "sim" in place of the resource descriptor.



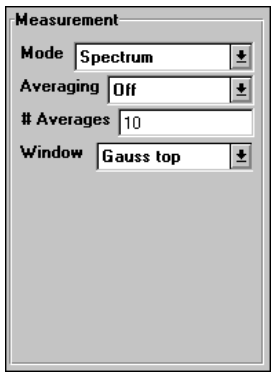
The buttons at the lower left of the SFP are always accessible and control various measurement and control functions.

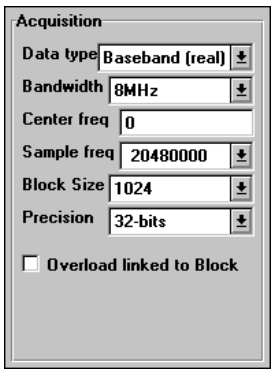


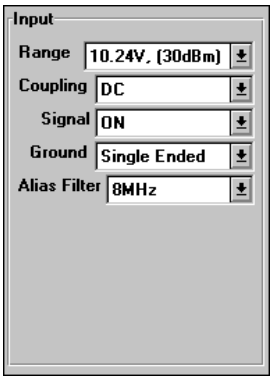
The menu bar at the upper left of the SFP allows you to select pull-down menus.

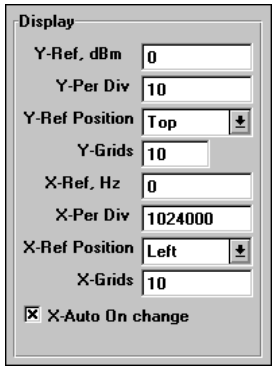


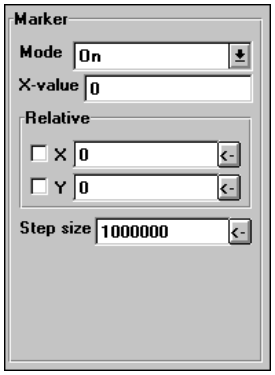
The left center section of the SFP is an area for which you may select various panels to control the measurement and display parameters. These panels are available as selections from the Control pull-down menu:

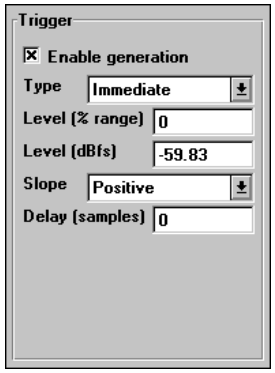






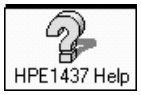






Hint: the E1437 online help, available from the SFP Help menu item or from the program group icon, describes these panels and has links to functions which control and define many of the parameters.

To Use Online Help in Windows



The E1437 Help icon accesses the online help file for the E1437A. The online help includes the programming library as well as general information.

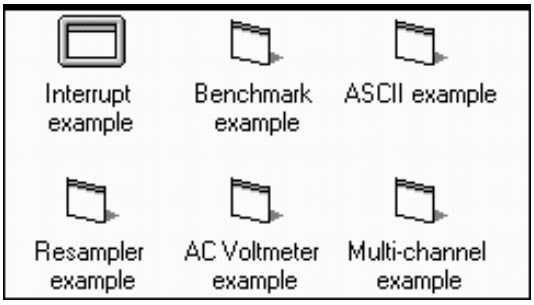
The DSP help icon accesses the online help file for the HPDSP library functions. These functions may be used to synthesize, resample, or perform special computations on data generated by the E1437A.

To Use the Example Programs

Several example programs are included to perform useful tasks for you and to serve as a basis for your own programs. When you installed your E1437A Windows or HP-UX libraries and drivers using the setup program or utility, you also installed executable and source code files for several useful example programs. The programs demonstrate programming the module with "C", Microsoft Visual Basic, and HP-VEE.

The executables for these examples require E1437A and, for Windows, VXIplug&play support; in other words they will not run in simulation mode like the E1437 Soft Front Panel program.

Icons for the executables appear in the E1437 Windows program group if you chose to add it during setup:



In Windows environments executable files and source code for the Microsoft Visual Basic examples are installed in the *drive*:\vxipnp\win[95|NT]\hpe1437\vb40 directory. The VEE examples are in the ...\hpe1437\vee directory, and "C" examples are in the ...\hpe1437\msc directory.

In the HP-UX environment executable files and source code for the C-language examples are installed in /opt/vxipnp/hpux/hpe1437/demo.

The group of programs described here may be supplemented with additional programs later which will be described in the online help or readme file.

acvolts.exe, acvolts_32.exe, acvolts

This is about the simplest practical complete program using the E1437 and functions like an AC voltmeter. It is written in Visual Basic and can be run on Win 3.1 (acvolts.exe), Win95 or WinNT (acvolts_32.exe). It is also available in C for HP-UX (acvolts).

ascii.exe, ascii_32.exe, ascii

This example shows how to control the E1437 without using the C-function library. Since all I/O is performed with ASCII commands and the VXI message protocol, the speed is substantially reduced. This example still uses the VISA I/O library to send and receive ASCII commands, however any environment capable of ASCII I/O to VXI could be used. Users interested in controlling the E1437 via a command module should look at this example. The code is written in Visual Basic and can be run on Win 3.1, Win95, or WinNT.

resamp.exe, resamp_32.exe, resamp

This example shows how to use the resample function included in the HPDSP library shipped with the E1437. It is written in Visual Basic and runs on Win 3.1 (resamp.exe), Win95, or WinNT (resamp_32.exe). It is also available in C for HP-UX (resamp).

multchan.exe, multchan_32.exe, multichan

This example shows how to synchronize two modules to achieve simultaneous sampling, filter decimation, and matched local oscillator phase. It is written in Visual Basic and runs on Win 3.1 (multchan.exe), Win95 or WinNT (multchan_32.exe). It is also available in C for HP-UX (multichan).

bench.exe, bench_32.exe, bench

This performance benchmarking program is really more of a utility than an example, although source code is provided. It allows users to measure data transfer rates and command processing times on their system without having to write new code. The utility is written in Visual Basic and runs on Win 3.1 (bench.exe), Win95 or WinNT (bench_32.exe). It is also available in C for HP-UX (bench).

demo

This is a simple non-interactive oscilloscope display and is written in C for the HP-UX environment only.

interupt.exe

This example shows how to set up and trap a VXI interrupt to indicate an error condition in the E1437. It is written as a consol program in Microsoft Visual C++ and runs only on Win95 or WinNT. Source code is installed on Win 3.1, but no executable is provided.

scope.vee

This is a simple one-channel example written in VEE. In order to view or execute it, the VEE programming environment must be installed on the system. It is not installed on Win 3.1 or HP-UX.

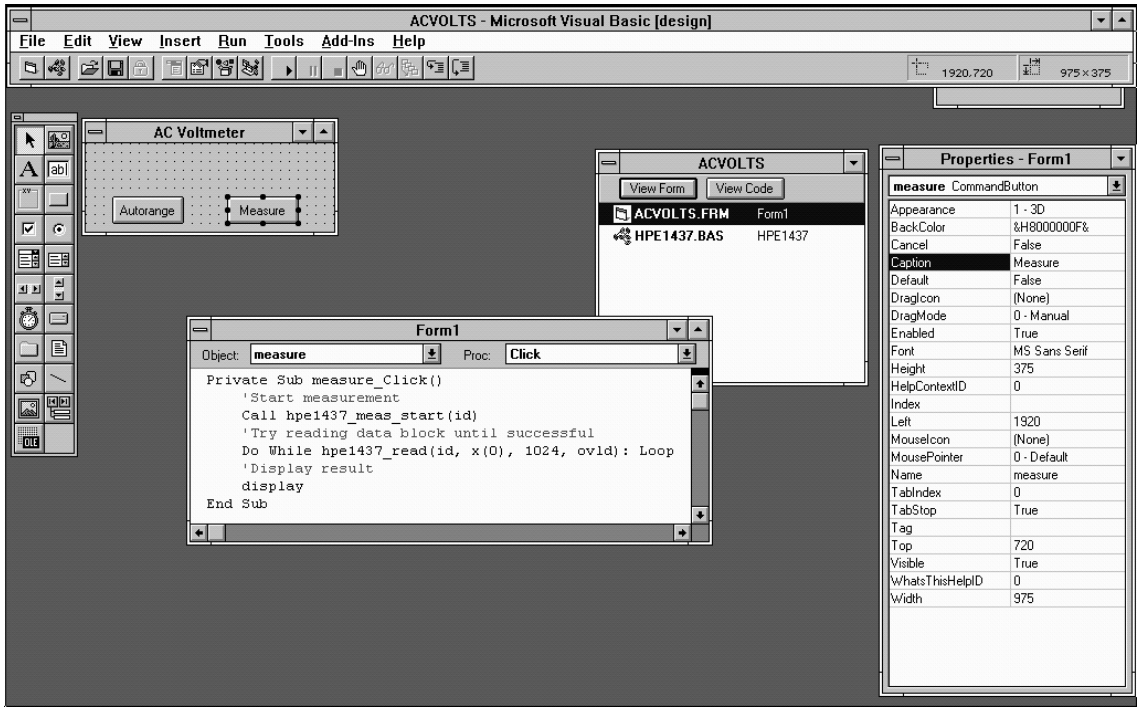
thruput.vee

This VEE example demonstrates how to set up a Local Bus data transfer from the E1437 to an E1562 data disk module. To use this example the VEE programming environment and the E1562 driver must be installed on the system. It is not installed on Win 3.1 or HP-UX.

The next few pages show the structure and some details of a few of the example programs

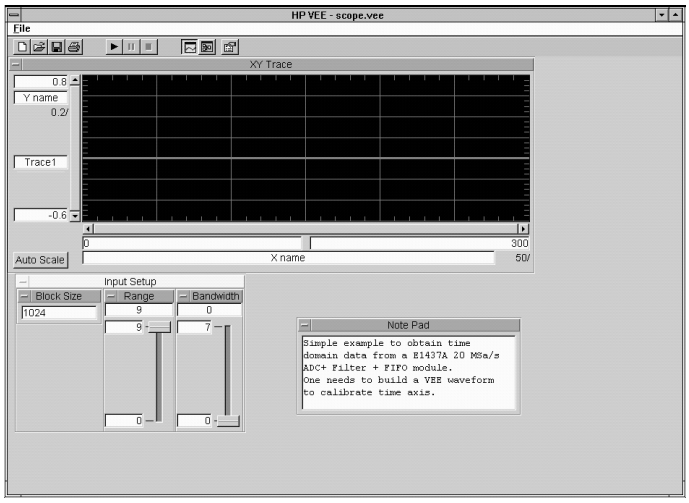
To View the Visual Basic Example Program

The acvolts.vbp project from which the acvolts.exe example program was created demonstrates how to communicate with the E1437A module in Visual Basic. The example below shows the open project with an open form and an open object.

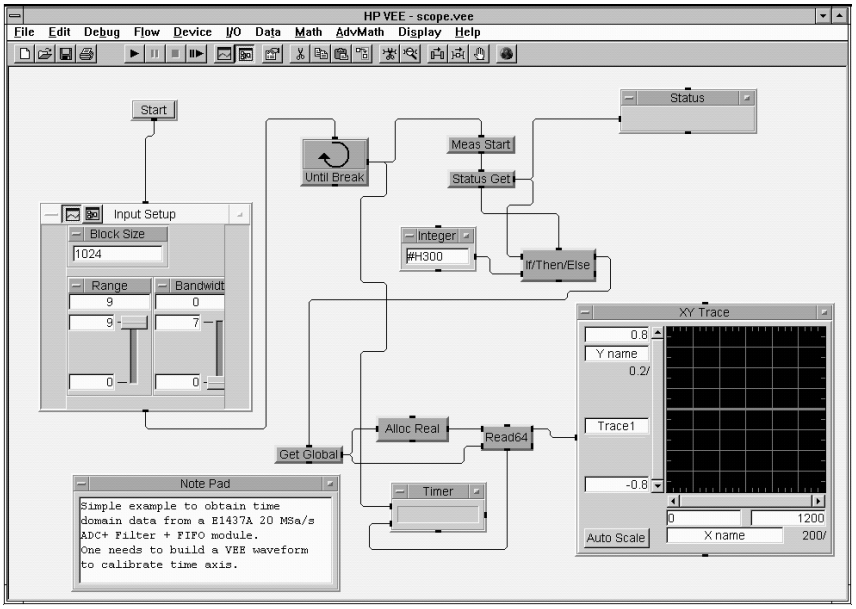


To Use the HP-VEE Example Program

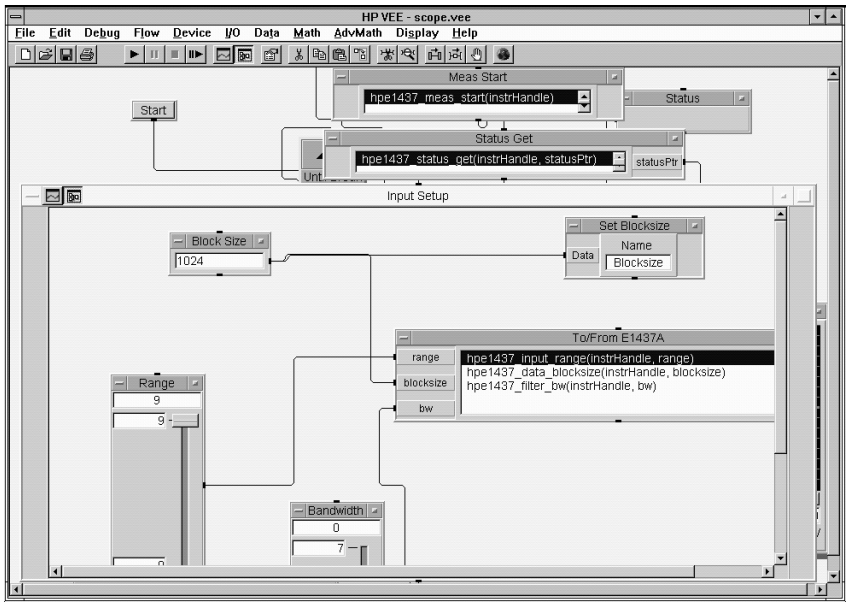
The scope.vee program demonstrates a simple example of how to use the E1437A in a HP-VEE program. Load HP-VEE and the scope.vee. You may run the program to measure a signal and may select input parameter variables in the boxes provided.



You may also view the detail of the HP-VEE program to see how the program is structured:



The view below shows detail within the input setup, meas start and status get boxes. These are examples of how HP-VEE communicates with the E1437A module.

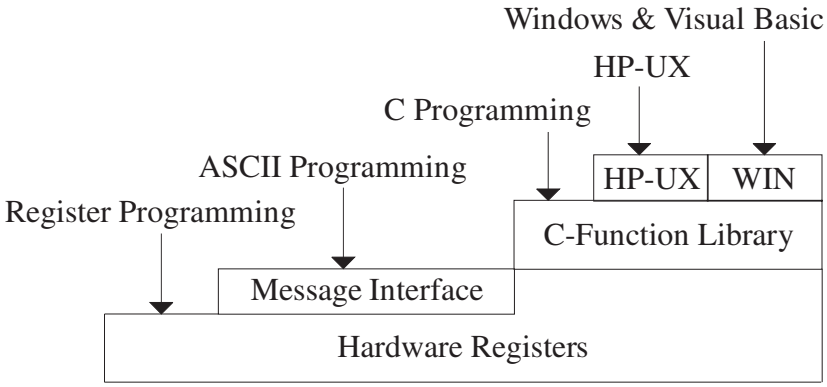


3

Using the E1437A

Programming the E1437A

The E1437A is shipped with software and documentation to support a broad set of choices of controllers, I/O interfaces, programming languages, and operating systems. By virtue of its compliance to the VXI*plug&play* standard, the E1437A is most easily controlled in an environment conforming to one of the supported VXI*plug&play* frameworks. However, support is also supplied for other common hardware and software environments. The relationship among the various levels of programming the E1437A is shown in the diagram below.



WIN framework

The primary development environment supported by the E1437A is the VXI*plug&play* WIN, WIN95, and WinNT framework specifications. It requires the following resources prior to the installation of the E1437A:

- An embedded or a stand-alone IBM compatible PC
- Microsoft Windows 3.1 or higher
- VISA interface library
- VISA compatible hardware interface
- Microsoft Visual C++ and/or Microsoft Visual Basic development system.

Additional details on the WIN framework can be found in the VXI*plug&play* VPP-2 System Frameworks Specification, Revision 2.0.

In addition to the C source code files, the E1437A includes compiled libraries, example programs, an interactive soft front panel program, online help files, and an installation program. The interactive soft front panel program allows the E1437A to be turned on, verified and used for simple tasks without writing any user programs.

Compliance with the VXI*plug&play* WIN framework allows users of the HP-VEE graphical programming system to control the E1437A from that environment. This is accomplished by the capability of HP-VEE to call functions in the C-library. Documentation and support for that capability is included with HP-VEE and is not addressed further in this document.

HP-UX, Series 700 Environment

Although HP-UX will not support an official VXI*plug&play* framework before version 10.2, the HP-UX environment is supported for developers who prefer programming tools provided on the UNIX operating system. The system requirements include:

- HP/Agilent series 700 workstation
- HP-UX operating system 9.x
- Standard Instrument Control Library (SICL)
- SICL compatible VXI hardware interface
- C-language programming system.

In addition to the source code files, the E1437A includes compiled libraries, example programs, online help files, and an installation utility.

C Programming

The E1437A is shipped with a source library of C-functions which can be called from user programs. This elevates the interface above the register level so the programmer no longer has to be concerned with such things as register addresses and packing or splitting parameters into 16-bit register lengths. The library includes ANSI compliant source code files with all machine dependent code constrained to a single source file. By re-writing selected portions of the *machine.h* file, the programmer can create and compile an E1437A library which is compatible with virtually any development environment using the C language. The most common reason for re-writing *machine.h* is to accommodate I/O libraries other than VISA or SICL. In some cases the library may need merely to be re-compiled to target a different processor type for the host computer.

Porting the E1437A library to a different computer environment is likely to be a fairly straight forward task. However, some of the higher level tools shipped with the E1437A may not be as easily ported. The interactive soft front panel and some example programs include human interfaces which depend on certain display and keyboard support which may be system dependent. Although source code is included for these applications, porting them to a different environment may present a greater problem than porting the library itself. The installation and online help utilities are specifically targeted to operate on the supported development environments and may not be available in other environments.

ASCII Programming

For programmers familiar with instrument control using ASCII string commands, the E1437A hardware implements a message based interface using ASCII commands compatible with the IEEE-488.2 standard. This standard defines the command syntax which is used by the Standard Commands for Programmable Instruments (SCPI) specification. For consistency with the new VXI*plug&play* function definitions, the E1437A ASCII command set does not use the SCPI commands.

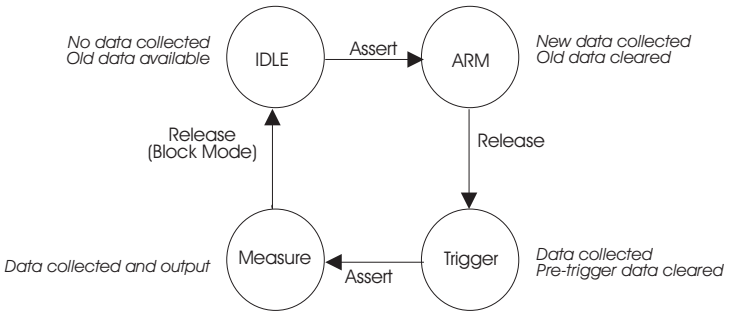
Since the ASCII interpreter is built into the E1437A hardware, no host library is necessary for ASCII programming. Thus, there is no software to install. There is no need for a separate interpreter in the host computer (CSCPI or ISCPI). There is no need to download an interpreter to a separate command module. A key advantage of ASCII programming is that it can be done in virtually any VXI environment which supports message based I/O. A disadvantage of ASCII programming is the lack of host-based tools such as diagnostics and demonstration programs. An additional disadvantage is the reduction in I/O performance due to the character-based serial message interface and interpreter.

Register Programming

The lowest level of programming supported by the E1437A allows direct writing and reading to the binary hardware registers. There is no user-level support for register programming.

The Measurement loop

The measurement loop progresses through four states. The transition from one state to the next is tied to the transition of the SYNC signal. The effect of the SYNC signal is summarized in the following diagram representing the four possible states of an E1437 module.



In the *Idle* state the E1437 places no new data into the FIFO output buffer memory although previously measured data is retained in the buffer memory and is available for output via the VME or local bus I/O ports. The module stays in the Idle state until the SYNC line is asserted.

Upon entering the *Arm* state the E1437 clears old data and starts saving new data into its FIFO. It remains in the Arm state until the SYNC signal is released. If an E1437 is programmed with a pre-trigger delay, it collects enough data samples to satisfy this pre-trigger delay, and then releases the SYNC line. If no pre-trigger delay has been programmed, the module releases the SYNC line immediately. When all E1437s in a system have released the SYNC line the module moves to the *Trigger state*.

Upon entering the *Trigger* state an E1437 continues collecting data into the FIFO, discarding any data prior to the pre-trigger delay. An E1437 remains in the Trigger state until the SYNC line is asserted. The SYNC line may be asserted by a direct command or by any E1437 which encounters a trigger condition and is programmed to assert the SYNC line. When the SYNC signal is asserted, all modules synchronously move to the *Measure* state.

In the *Measure* state the E1437 continues collecting data and sends the data saved in the FIFO memory to the selected I/O port, starting with the sample indicated by the trigger arrival, offset by the trigger delay. This data transfer continues until all data has been transferred or until the module meets the criteria for returning to the Idle state imposed by *block mode* or *continuous mode* operation constraints.

Modules programmed for *block mode* operation will assert the SYNC line until a complete block of data, including any pre-programmed pre- or post-trigger delay, has been collected and is available to the I/O port. The module then releases the SYNC line and returns to the Idle state.

In *continuous mode* a module releases sync immediately but moves to the Idle state only if explicitly programmed to do so or if the FIFO data buffer overflows because data cannot be read from the I/O port fast enough.

The Measurement Loop in Multi-module systems

The following rules generally apply to transitions between states when multiple modules share a SYNC signal:

- If any one module *asserts* the SYNC line a synchronous state transition occurs for all modules in a system.
- All modules in a system must have *released* the SYNC line in order to bring about a synchronous transition to Trigger state.
- In block mode each module releases the SYNC line after its block of data has been collected. After each block mode module has released the SYNC line the individual module moves to the Idle state.
- Immediately upon entering the Measure state in continuous mode each module releases the SYNC line but does not move into the Idle state. It continues to collect and output data until it is programatically signaled to stop or until the FIFO overflows. With the SYNC line released it is then possible to change the center frequency for one or multiple modules without interrupting the measurement. See the section on Synchronizing Changes in Multi-Module Systems.
- A module may be programmed explicitly to inhibit its transition to the Arm state despite SYNC transitions.
- In addition to controlling the progression through the four module states, the SYNC signal is used to synchronize the decimation counters and local oscillators of multiple E1437 modules.

Frequency and Filtering

The E1437's center frequency is normally set at zero (baseband measurement). However, you may set the center frequency to a non-zero value in order to examine a narrower span away from baseband (zoom measurement). The frequency band of interest, represented by digitized time data samples from the ADC, is mixed with the E1437 digital LO, a complex exponential, at the desired center frequency. As a result the frequency band of interest in the input signal is shifted to a complex signal centered around DC. See Synchronizing Changes in Multi-module Systems for special considerations with respect to changing the center frequency in multi-module systems.

The default filter for E1437 measurements is an analog anti-alias filter. However, you may further isolate the frequency band of interest for more detailed analysis by using digital filtering. A decimating digital filter simultaneously decreases the bandwidth of the signal and decreases the sample rate. The built-in digital filters conform to the Nyquist sampling theorem which guarantees that the output sample rate may be reduced by the same factor as the signal bandwidth reduction while still maintaining a complete representation of the underlying bandlimited signal.

For each octave step in bandwidth reduction (except for the first octave) the E1437 digital filters automatically reduce the data rate by discarding alternate output samples. This process, called decimation, results in an output sample rate which is nominally four times the signal bandwidth whenever $sigBw>0$. This is still double the theoretical rate necessary to fully characterize the band limited signal. However, because the digital filters do not have a perfectly abrupt cutoff, the sample rate cannot be reduced to the theoretical limit without some aliasing of signals in the transition frequency band of the filters. In many applications this limited aliasing potential is not important. For this reason you may optionally choose to apply a final factor-of-two decimation. See the Technical Specifications for detailed information on the digital filter shapes.

The decimation process used to reduce the output sample rate is driven from a "decimation counter" which keeps track of which samples to save and which ones to discard for each of the octave bandwidth reduction filter stages. In multi-module systems where synchronous sampling is required, the decimation counters in all the modules must be synchronous with each other. See Synchronizing Changes in Multi-module Systems.

Managing multiple modules

The E1437 supports synchronous operation among multiple E1437s by using a shared ADC clock and SYNC signal to drive all the modules in a system. The shared SYNC signal is used to synchronize critical operations including arming, triggering the beginning of data collection, setting a common phase of the local oscillator for down conversion, and forcing concurrent output sample times whenever decimation is used. The SYNC line transitions are constrained to not occur during the critical (setup and hold) regions of the shared ADC clock. Thus, all modules in the group can be assured of receiving the SYNC signal on exactly the same ADC clock cycle. The following topics provide details on sharing clock and SYNC signals:

Clock distribution

When shared, the ADC clock and SYNC lines are distributed among modules either on the VXI backplane using the ECL Trigger lines, or on the front panel using the SMB Clock/Sync extender connectors. When VXI backplane distribution is used with more than one VXI mainframe, the front panel Clock/Sync connectors can be used to buffer the ADC clock and SYNC lines from one mainframe to another.

Since the SYNC transition timing relative to the ADC clock edges is critical, the module driving the SYNC line should ideally be the same one identified as the master. However, when using backplane distribution, any E1437 in the same mainframe as the master can drive the SYNC line.

When using the multi-sync mode of operation, the selection of front panel or backplane distribution of ADC clock and SYNC signals involves the following considerations:

- Backplane distribution requires the use of the ECL Trigger lines on the backplane, which are then unavailable to other modules.

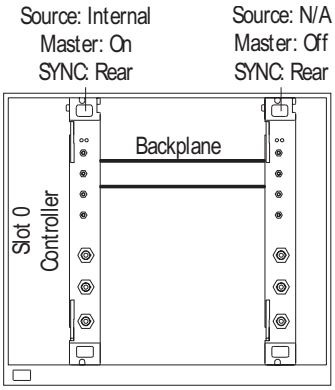
- The overall time skew between the arrival of ADC clock edges is smaller when using backplane distribution, particularly if the master (or buffer) module is physically located in the center of the mainframe.

- Backplane distribution is more susceptible to pickup of jitter on the ADC clock from other digital activity on the VXI backplane. The extent of this pickup depends on the mainframe and on the other modules in the mainframe. One important step in reducing this pickup is to disable, whenever possible, the 10 MHz VXI clock generated by the slot-0 controller.

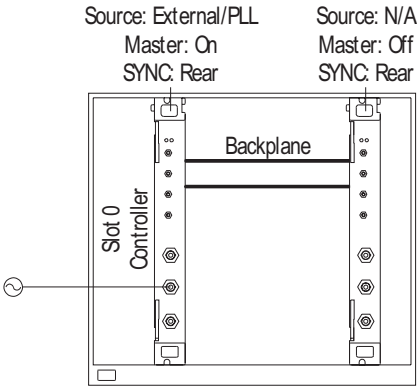
- For backplane distribution make sure that all modules conform to VXI specification 1.4 or later with regard to their attachment to the ECL Trigger lines. See the Technical Specifications for the clock jitter (phase noise) specification degradation using backplane distribution.

- Front panel distribution requires the use of two short, relatively well matched cables with SMB connectors between modules. In addition, unused SMB connectors on modules being used for front panel distribution must be terminated in 50 ohms.

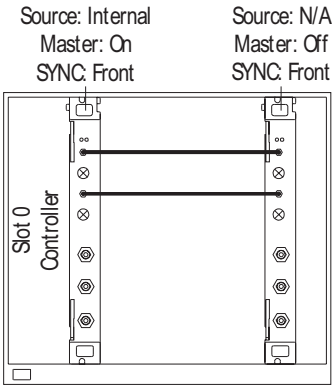
Managing Multi-module Systems



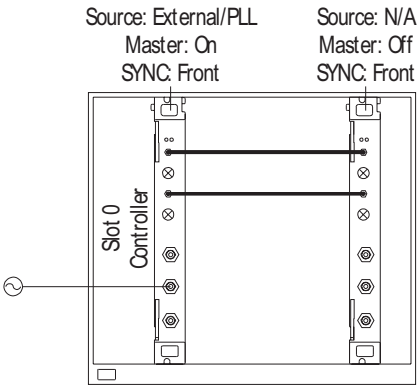
ADC clock and SYNC distribution using VXI backplane ECL trigger lines.



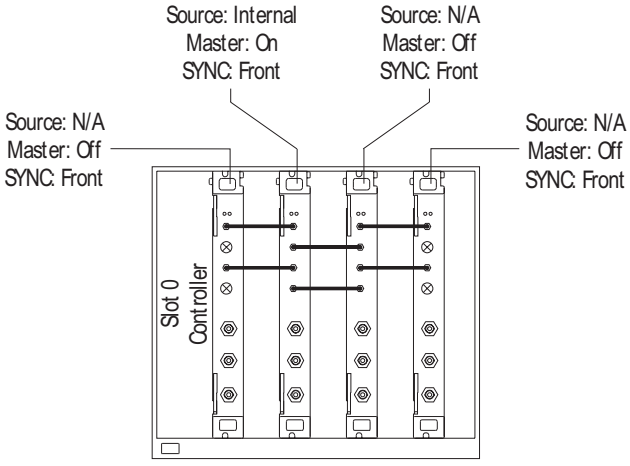
External clock and SYNC distribution using VXI backplane ECL trigger lines.



ADC clock and SYNC distribution using front panel SMB clock and SYNC extender connections.

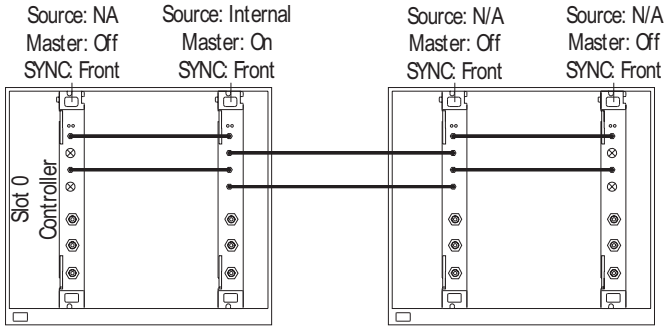


External clock and SYNC distribution using front panel SMB clock and SYNC extender connections.

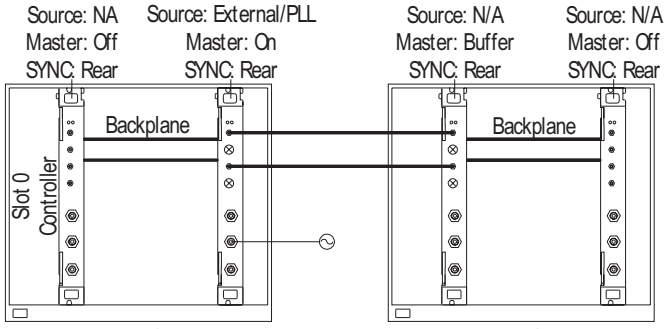


Sharing clock and SYNC among several modules via front panel distribution.

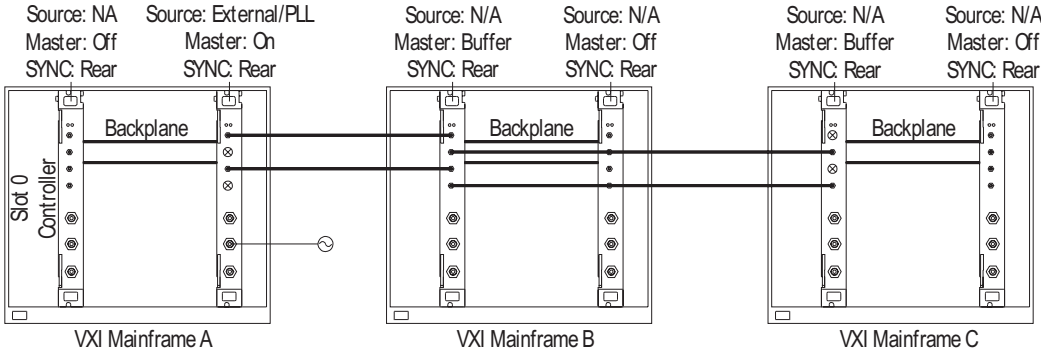
Managing Multi-Mainframe Systems



Clock and SYNC distribution using front panel extender connections within and between mainframes.



Clock and SYNC distribution using VXI backplane lines within mainframes and using front panel extender connections between mainframes.



Three or more mainframes with clock and SYNC distribution using VXI backplane lines within mainframes and using front panel extender connections between mainframes.

Synchronizing Changes in Multi-module Systems

Multi-module systems require special treatment with respect to timing of frequency and filter changes. Center frequency changes may involve synchronizing the local oscillators of all modules in a system. Digital filters changes in multi-module systems require that the decimation counters be synchronized.

Synchronous Digital Filter Changes

In multi-module systems where synchronous sampling is required, the decimation counters in all the modules must be synchronous with each other. This condition can be forced by preparing each module in the system in advance. Any measurement in progress is terminated at this time and the module is placed in the Idle state. After each module is prepared, the next SYNC line transition causes the digital decimation counter to be reset and started at the same time. Once this is done the decimation counters will stay synchronized as long as the same ADC clock is used.

If you also intend to change the center frequency along with the digital filters, you should synchronize the digitial filters first. Otherwise the center frequency phase becomes unsynchronized when the digital filters are changed.

Synchronous Center Frequency Changes

In multi-module systems you may prepare each module in advance of a frequency change, then perform the change synchronously by asserting the SYNC line. This preserves the phase relationship of the local oscillators for all modules in the system. Certain special considerations apply to multi-module frequency changes:

- If all modules in a system are in the Idle state when the SYNC line transition occurs, the LO frequency will be updated and the next measurement will be armed.
- If all modules are in the measurement state in continuous mode when the SYNC line transition occurs, the LO frequency will be synchronously updated, and the measurement will continue.
- In continuous mode care must be taken to assure that all modules are in the same state, either the Idle state or the Measure state, before the SYNC line transition occurs, otherwise some modules will re-arm while others will continue the current measurement.
- In block mode the SYNC line transition will be ignored unless all modules are currently in the Idle state.
- If you also intend to change the digital filters along with the center frequency, you should synchronize the digitial filters first. Otherwise the center frequency phase becomes unsynchronized when the digital filters are changed.

Transferring data

You can transfer data from the E1437 two different ways.

- The VMEbus is the universal data bus for VXI architecture. It provides flexibility and versatility in transferring data. Transfers over the VMEbus are 16 bits wide.
- The Local Bus supports faster transfer rates than the VMEbus. For example, if you are transferring data from the E1437 to the HP/Agilent E1485A/B, the Local Bus provides a direct pipeline to the HP/Agilent E1485's DSPs.

  Using the Local Bus, you can transfer data in the background while processing data in a signal-processing module. All Local Bus data transfers originate in the E1437 and move towards a signal processing module to the right of the E1437. If other modules generate data to the left of the input module, the E1437 will pass the data to its right and insert or append its own data at the beginning or end of the frame.

4

E1437A VXI*plug&play* Programmer's Reference

Introduction

The programmer's reference is presented as a set of VXI*plug&play* functions since this is the primary targeted environment. However, when you performed the setup for the E1437A, drivers were installed to support various programming environments as described in the Programing Overview section in the "Using the E1437A" chapter.

The function descriptions in the programmer's reference are valid for all environments except ASCII, which is treated in a separate chapter. Be sure to follow the instructions in the "Getting Started" chapter to assure proper installation and to become aware of the capabilities of your E1437A software in various programming environments. You will find the example programs particularly helpful for programming in different environments.

Many of the function descriptions in the programming reference include several related functions. You may use the primary function to set all related parameters or you may use the other functions within the group to set or query a single parameter.

Parameter variables are presented as alphanumeric values which are descriptive and easy to remember. However, for faster programming you may use the numeric equivalents for the parameter variables listed at the end of this chapter. The numeric equivalents are available as popups in the online help, a good reason to use the online help, if it is available in your environment, rather than this printed document.

Unless noted otherwise, all functions in this library return 0 if they complete succesfully and a non-zero integer if they fail. Always check the the return value and take appropriate action. The error descriptions are listed at the end of this chapter and in the online help.

Functions Listed by Functional Group

The following pages have the programming functions grouped by related functions. The a brief description of each group follows:

- Initializing the E1437:
  You must first initialize the I/O driver and set up each module.
- Configuring the Analog Inputs:
  The functions in this group determine how the analog input section is configured.
- Formatting Data:
  An E1437 can collect either real or complex data in 16-bit or 32-bit format. It can collect data into various blocksizes or in a continuous mode. This data can be transferred either on the VXI backplane of over the Local Bus. You can append status information to each block of data indicating ADC overloads or ADC errors during the block.
- Configuring Digital Processing:
  The decimation filter provides bandpass filtering and decimation capabilities. You may also select limited frequency spans away from baseband.
- Controlling Measurements:
  These functions initiate or terminate the measurement loop.
- Timing:
  The clock signals for the ADC sample clock and DSP decimation and zoom can be set from a variety of sources. One E1437 can be enabled to drive the sample clock line on the VXI backplane or front panel to enable synchronization of multiple E1437s.
- Triggering:
  These functions set all parameters associated with triggering the beginning of data collection.
- Controlling Multiple Modules:
  These functions support synchronous operation among multiple E1437s by using a shared ADC clock and SYNC signal to drive all the modules in a system.
- Reading Data:
  These functions read data from either the VME or the Local Bus data port. This data can optionally be scaled and converted to floating point.
- Programming Interrupts:
  The E1437 can be programmed to interrupt via the VXI backplane whenever certain status conditions are present.
- Debugging your Program:
  Error messages allow you to identify program problems.
- Diagnostics:
  Hardware diagnostic routines verify correct hardware operation of the E1437.

Analog Setup

hpe1437_input_setup - sets all the analog input parameters

hpe1437_input_alias_filter - include/bypasse the built-in analog anti-alias filter

hpe1437_input_alias_filter_get - gets the anti-alias filter state

hpe1437_input_autozero - nulls out the input DC offset

hpe1437_input_coupling - selects AC or DC input coupling

hpe1437_input_coupling_get - get the input coupling type

hpe1437_input_float - enables/disables floating the input connector

hpe1437_input_float_get - gets the input connector state

hpe1437_input_range - sets the full scale range

hpe1437_input_range_auto - performs auto-ranging

hpe1437_input_range_get - gets the input range

hpe1437_input_signal - include/bypass the input buffer amplifier

hpe1437_input_signal_get - gets the input buffer amplifier state

Data Format

hpe1437_data_ -sets all format and data output flow parameters

hpe1437_data_append_status - enables/disables appending status information to a data block

hpe1437_data_append_status_get - gets the append status state

hpe1437_data_blocksize - determines the size of the output data block

hpe1437_data_blocksize_get - gets the output data block size

hpe1437_data_memsize_get - returns module's memory size

hpe1437_data_mode - selects block mode or continuous mode

hpe1437_data_mode_get - gets the data mode

hpe1437_data_port - selects VME bus or local bus transmission

hpe1437_data_port_get - gets the output port designation

hpe1437_data_resolution - selects 16 or 32 bits data resolution

hpe1437_data_resolution_get - gets the data resolution

hpe1437_data_type - selects real or complex output data

hpe1437_data_type_get - gets output data type

hpe1437_lbus_mode - sets the transmission mode of the local bus

hpe1437_lbus_mode_get - gets the local bus mode

hpe1437_lbus_reset - resets local bus mode

hpe1437_lbus_reset_get - gets the local bus mode reset state

Debugging

hpe1437_error_message - returns error information obtained from function calls
hpe1437_error_query - queries the module for the most recent error
hpe1437_revision_query - returns strings that identify the date of the firmware revision.
hpe1437_status_get - retreives module's status register information

Digital Processing

hpe1437_filter_setup - sets the digital filter bandwidth and decimation filter parameters
hpe1437_filter_bw - selects a signal filter bandwidth
hpe1437_filter_bw_get - gets the signal filter bandwidth
hpe1437_filter_decimate - enables/disables and extra factor of 2 decimation
hpe1437_filter_decimate_get - gets current state of extra decimation
hpe1437_filter_resp_get - returns the module's complex frequency response.
hpe1437_filter_sync - synchronizes the decimation filter counter
hpe1437_frequency_setup - sets all center frequency parameters
hpe1437_frequency_center - sets the center frequency
hpe1437_frequency_center_get - gets the current center frequency
hpe1437_frequency_center_raw - A fast way to set the center frequency
hpe1437_frequency_cmplxdc - selects a complex baseband measurement
hpe1437_frequency_cmplxdc_get - gets the state of the baseband measurement mode
hpe1437_frequency_sync - prepares the module for a synchronous frequency change
hpe1437_frequency_sync_get - gets the state of the synchronus change mode

Diagnostics

hpe1437_self_test - performs a self-test on the module and returns the result

Initialization

hpe1437_init - initializes the I/O driver for a module
hpe1437_close - closes the module's software connection

Interrupts

hpe1437_attrib_get - allows direct access to the I/O library functions
hpe1437_interrupt_setup - sets all interrupt parameters
hpe1437_interrupt_mask_get - gets the interrupt event mask
hpe1437_interrupt_priority_get - gets the VME interrupt line
hpe1437_interrupt_restore - restores the interrupt masks to the most recent setting

Measurement

hpe1437_meas_control - initiates and controls measurements in multi-module systems
hpe1437_meas_start - initiates measurements in single module systems
hpe1437_reset - places the module in a known state

Reading data

hpe1437_data_scale_get - gets data scale factor

hpe1437_read - reads scaled 32-bit float data from FIFO

hpe1437_read64 - reads scaled 64-bit float data from FIFO, specifically for VEE applications

hpe1437_read_raw - - reads raw data from FIFO

Timing

hpe1437_clock_setup - sets all timing parameters

hpe1437_clock_dsp - selects the clock used to drive the decimation/zoom section

hpe1437_clock_dsp_get - gets the current decimation clock source

hpe1437_clock_fs - provides the frequency of an external sample clock

hpe1437_clock_fs_get - gets the current external sample clock frequency

hpe1437_clock_master - determines whether a module drives the VXI clock line with its ADC clock

hpe1437_clock_master_get - gets the module's clock master state

hpe1437_clock_multi_sync - specifies whether the module uses a shared clock and sync

hpe1437_clock_multi_sync_get - gets the module's current shared clock and sync state

hpe1437_clock_source - selects the source of the ADC clock

hpe1437_clock_source_get - gets the ADC clock source

Trigger

hpe1437_trigger_setup - sets all parameters associated with triggering the beginning of data collection

hpe1437_trigger_adclevel - specifies the threshold for the ADC trigger

hpe1437_trigger_adclevel_get - gets the trigger threshold

hpe1437_trigger_delay - specifies a pre- or post-trigger delay time

hpe1437_trigger_delay_get - gets the trigger delay time

hpe1437_trigger_delay_actual_get - gets the actual delay time from the most recent trigger event

hpe1437_trigger_gen - determines whether a module can generate a trigger

hpe1437_trigger_gen_get - gets the trigger generation status

hpe1437_trigger_maglevel - specifies the threshold for a magnitude trigger

hpe1437_trigger_maglevel_get - gets magnitude trigger threshold

hpe1437_trigger_phase_actual_get - gets the actual trigger phase from the most recent trigger event

hpe1437_trigger_phase_capture - Allows LO phase capture in frequency-synchronized, multi-module zoom measurements.

hpe1437_trigger_slope - selects a positive or negative trigger

hpe1437_trigger_slope_get - gets trigger slope

hpe1437_trigger_type - determines the trigger type

hpe1437_trigger_type_get - gets trigger type

Synchronization

hpe1437_clock_master - determines whether a module drives the VXI clock line with its ADC clock

hpe1437_clock_master_get - gets the module's clock master state

hpe1437_clock_multi_sync - specifies whether the module uses a shared clock and sync

hpe1437_clock_multi_sync_get - gets the module's current shared clock and sync state

hpe1437_clock_source - selects the source of the ADC clock

hpe1437_clock_source_get - gets the ADC clock source

hpe1437_filter_sync - synchronizes the decimation filter counter

hpe1437_frequency_sync - prepares the module for a frequency change

hpe1437_meas_control - synchronizes arming and triggering in multi-module systems

hpe1437_trigger_gen - determines whether a module can generate a trigger

hpe1437_trigger_gen_get - gets the trigger generation status

hpe1437_wait - facilitates the synchronization and control of multi-module systems

Functions Listed alphabetically

hpe1437_attrib_get - allows direct access to the I/O library functions

hpe1437_clock_dsp - selects the clock used to drive the decimation/zoom section

hpe1437_clock_dsp_get - gets the current decimation clock source

hpe1437_clock_fs - provides the frequency of an external sample clock

hpe1437_clock_fs_get - gets the current external sample clock frequency

hpe1437_clock_master - determines whether a module drives the VXI clock line with its ADC clock

hpe1437_clock_master_get - gets the module's clock master state

hpe1437_clock_multi_sync - specifies whether the module uses a shared clock and sync

hpe1437_clock_multi_sync_get - gets the module's current shared clock and sync state

hpe1437_clock_setup - sets all timing parameters

hpe1437_clock_source - selects the source of the ADC clock

hpe1437_clock_source_get - gets the ADC clock source

hpe1437_close - closes the module's software connection

hpe1437_data_append_status - enables/disables appending status information to a data block

hpe1437_data_append_status_get - gets the append status state

hpe1437_data_blocksize - determines the size of the output data block

hpe1437_data_blocksize_get - gets the output data block size

hpe1437_data_memsize_get - returns module's memory size

hpe1437_data_mode - selects block mode or continuous mode

hpe1437_data_mode_get - gets the data mode

hpe1437_data_port - selects VME bus or local bus transmission

hpe1437_data_port_get - gets the output port designation

hpe1437_data_resolution - selects 16 or 32 bits data resolution

hpe1437_data_resolution_get - gets the data resolution

hpe1437_data_scale_get - gets data scale factor

hpe1437_data_ -sets all format and data output flow parameters

hpe1437_data_type - selects real or complex output data

hpe1437_data_type_get - gets output data type

hpe1437_error_message - returns error information obtained from function calls

hpe1437_error_query - queries the module for the most recent error

hpe1437_filter_bw - selects a signal filter bandwidth

hpe1437_filter_bw_get - gets the signal filter bandwidth

hpe1437_filter_decimate - enables/disables and extra factor of 2 decimation

hpe1437_filter_decimate_get - gets current state of extra decimation

hpe1437_filter_resp_get - returns the module's complex frequency response.

hpe1437_filter_setup - sets the digital filter bandwidth and decimation filter parameters

hpe1437_filter_sync - synchronizes the decimation filter counter
hpe1437_frequency_center - sets the center frequency
hpe1437_frequency_center_get - gets the current center frequency
hpe1437_frequency_center_raw - A fast way to set the center frequency
hpe1437_frequency_cmplxdc - selects a complex baseband measurement
hpe1437_frequency_cmplxdc_get - gets the state of the baseband measurement mode
hpe1437_frequency_setup - sets all center frequency parameters
hpe1437_frequency_sync - prepares the module for a synchronous frequency change
hpe1437_frequency_sync_get - gets the state of the synchronus change mode
hpe1437_init - initializes the I/O driver for a module
hpe1437_input_alias_filter - include/bypass the built-in analog anti-alias filter
hpe1437_input_alias_filter_get - gets the anti-alias filter state
hpe1437_input_ - nulls out the input DC offset
hpe1437_input_coupling - selects AC or DC input coupling
hpe1437_input_coupling_get - get the input coupling type
hpe1437_input_float - enables/disables floating the input connector
hpe1437_input_float_get - gets the input connector state
hpe1437_input_range - sets the full scale range
hpe1437_input_range_auto - performs auto-ranging
hpe1437_input_range_get - gets the input range
hpe1437_input_setup - sets all the analog input parameters
hpe1437_input_signal - include/bypass the input buffer amplifier
hpe1437_input_signal_get - gets the input buffer amplifier state
hpe1437_interrupt_mask_get - gets the interrupt event mask
hpe1437_interrupt_priority_get - gets the VME interrupt line
hpe1437_interrupt_restore - restores the interrupt masks to the most recent setting
hpe1437_interrupt_setup - sets all interrupt parameters
hpe1437_lbus_mode - sets the transmission mode of the local bus
hpe1437_lbus_mode_get - gets the local bus mode
hpe1437_lbus_reset - resets local bus
hpe1437_lbus_reset_get - gets the current local bus reset state
hpe1437_meas_control - initiates and controls measurements in multi-module systems
hpe1437_meas_start - initiates measurements in single module systems
hpe1437_read - reads scaled 32-bit float data from FIFO
hpe1437_read64 - reads scaled 64-bit float data from FIFO, specifically for VEE applications
hpe1437_read_raw - - reads raw data from FIFO
hpe1437_reset - places the module in a known state
hpe1437_revision_query - returns strings that identify the date of the firmware revision
hpe1437_self_test - performs a self-test on the module and returns the result
hpe1437_status_get - retreives module’s status register information

hpe1437_trigger_adclevel - specifies the threshold for the ADC trigger

hpe1437_trigger_adclevel_get - gets the ADC trigger threshold

hpe1437_trigger_delay - specifies a pre- or post-trigger delay time

hpe1437_trigger_delay_get - gets the trigger delay time

hpe1437_trigger_delay_actual_get - gets a representation of the phase value of the LO at the trigger point

hpe1437_trigger_gen - determines whether a module can generate a trigger

hpe1437_trigger_gen_get - gets the trigger generation status

hpe1437_trigger_maglevel - specifies the threshold for a magnitude trigger

hpe1437_trigger_maglevel_get - gets magnitude trigger threshold

hpe1437_trigger_phase_actual_get - gets the actual trigger phase from the most recent trigger event

hpe1437_trigger_phase_capture - Allows LO phase capture in frequency-synchronized, multi-module zoom measurements.

hpe1437_trigger_setup - sets all parameters associated with triggering the beginning of data collection

hpe1437_trigger_slope - selects a positive or negative trigger

hpe1437_trigger_slope_get - gets trigger slope

hpe1437_trigger_type - determines the trigger type

hpe1437_trigger_type_get - gets trigger type

hpe1437_wait - facilitates the synchronization and control of multi-module systems

VXI *plug&play* Programming Reference

hpe1437_attrib_get

Allows direct access to the I/O library functions.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_attrib_get**(ViSession ***id***, ViInt16 ***attrib***, ViPint32 ***value***);

Description

hpe1437_attrib_get is used primarily to manage the use of interrupts. Since interrupts are a shared resource across all modules using the VXI interface, it is not possible for the E1437 library, which governs single modules, to provide the functions to properly manage interrupts.

This function is used to access either the I/O library handle or the mapped I/O base address of the module. You should refer to the appropriate VISA or SICL documentation for descriptions of the I/O library functions.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

attrib designates the type of attribute to return. **HPE1437_IO_HANDLE** accesses the I/O library handle. **HPE1437_IO_ADDRESS** points to the mapped I/O base address of the module. **HPE1437_RM_HANDLE** accesses the I/O library handle of the default resource manager. **HPE1437_DATA_REGISTER** points to the mapped address of the E1437 data register. One or both of these parameters are used when calling I/O library functions directly.

value is the value of the requested attribute. For a VTL/VISA I/O library the value of the handle attribute corresponds to the *vi* parameter used by the majority of the I/O functions. For the SICL I/O library the handle is equivalent to the session parameter used by the majority of the I/O functions. In the case of SICL the long handle value should be cast to a short in order to be type compatible with the SICL session. The address attribute points to the base of the mapped I/O address space, regardless of which underlying I/O library is used.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_interrupt_setup

hpe1437_clock_setup

hpe1437_clock_setup sets all timing parameters. This description also includes information on the following functions which set or query the timing parameters individually:

hpe1437_clock_dsp selects the clock used to drive the decimation/zoom section.

hpe1437_clock_dsp_get gets the current decimation clock source

hpe1437_clock_fs provides the frequency of an external sample clock.

hpe1437_clock_fs_get gets the current external sample clock frequency

hpe1437_clock_master determines whether a module shares its ADC clock.

hpe1437_clock_master_get gets the module's clock master state

hpe1437_clock_multi_sync specifies whether the module uses a shared clock and sync

hpe1437_clock_multi_sync_get gets the module's current shared clock and sync state

hpe1437_clock_source selects the source of the ADC clock

hpe1437_clock_source_get gets the ADC clock source

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_clock_setup**(ViSession ***id***, ViInt16 ***sync***, ViInt16 ***source***, ViInt16 ***dsp***, ViInt16 ***master***, ViReal64 ***fs***);

ViStatus **hpe1437_clock_dsp**(ViSession ***id***, ViInt16 ***dsp***);

ViStatus **hpe1437_clock_dsp_get**(ViSession ***id***, ViPInt16 ***dspPtr***);

ViStatus **hpe1437_clock_fs**(ViSession ***id***, ViReal64 ***fs***);

ViStatus **hpe1437_clock_fs_get(**ViSession ***id***, ViPReal64 ***fsPtr***);

ViStatus **hpe1437_clock_master**(ViSession ***id***, ViInt16 ***master***);

ViStatus **hpe1437_clock_master_get**(ViSession ***id***, ViPInt16 ***masterPtr***);

ViStatus **hpe1437_clock_multi_sync**(ViSession ***id***, ViInt16 ***sync***);

ViStatus **hpe1437_clock_multi_sync_get**(ViSession ***id***, ViPInt16 ***syncPtr***);

ViStatus **hpe1437_clock_source**(ViSession ***id***, ViInt16 ***source***);

ViStatus **hpe1437_clock_source_get**(ViSession ***id***, ViPInt16 ***sourcePtr***);

Description

hpe1437_clock_setup is used to configure all timing parameters used for sampling (ADC clock) and decimation/zoom (DSP clock). This function, as well as the other **hpe1437_clock_** functions covered in this description, is used to select the source and distribution of clocking and synchronization signals used by the E1437 module. The primary clock signal used by the module is the ADC clock, for which the rising edges indicate the time for each sample of the analog-to-digital converter. Another clock signal is the DSP clock, which drives the digital signal processing and memory sections of the module. Normally the DSP clock is the same as the ADC clock, and data is transferred synchronously from the ADC to the DSP portion of the module. However, in certain situations the two clocks may be independent, requiring asynchronous data transfers from the ADC to the DSP. The remaining **hpe1437_clock_** functions listed above set or query the parameters individually.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

sync is used to specify whether the module uses a shared ADC clock and SYNC signal. If the *sync* parameter is set to **HPE1437_OFF** the ADC clock and SYNC are generated locally. If *sync* is set to **HPE1437_REAR** the module uses the shared ADC clock and SYNC signals which are distributed on the VXI backplane using the ECL trigger lines. If *sync* is set to **HPE1437_FRONT** the module uses the shared clock and SYNC provided on the front panel distribution connectors. Modules in multi-module systems must all have the same *sync* parameter setting.

syncPtr contains the current value of the *sync* parameter.

source selects the clock source that is used to drive the analog to digital converter (ADC) for single module operation or when a module is used as the master ADC clock source for a multi-module system. When set to **HPE1437_20000KHZ** the clock source is the internal 20 MHz oscillator. When set to **HPE1437_20480KHZ** the clock source is the internal 20.48 MHz oscillator. **HPE1437_EXTERNAL** selects the TTL, ECL, or sine signal on the external BNC front panel clock input connector. When using an external clock the *fs* parameter is used to provide the module with the frequency of the external clock. **HPE1437_EXT_PLL_REF** takes a 10 MHz reference from another instrument on the external BNC front panel clock input connector and uses a PLL to convert it to a 20 MHz reference. In multi-module systems the *source* parameter is ignored for all but the master module.

sourcePtr contains the current value of the *source* parameter.

dsp selects the clock used to drive the decimation/zoom section within the E1437. Normally, the DSP clock should be coupled to the ADC clock whenever possible since the spurious performance specification is degraded when the clocks are independent. However, when a slow or intermittent ADC clock results in greater than 1 µs between clock edges, the DSP clock must be generated from the internal oscillator to avoid data loss in the dynamic RAM. Setting this parameter to **HPE1437_ADC** forces the DSP clock to be driven by the ADC clock. **HPE1437_OSCILLATOR** will cause the DSP clock to be the internally generated 20.48 MHz oscillator. Note that the computed results will be the same in either case.

dspPtr contains the current value of the *dsp* parameter.

master determines whether an E1437 makes its local ADC clock available to other modules as a shared clock. Multi-module synchronization requires one and only one of the modules to be identified as the master, that is, the source of the shared ADC clock. Setting this parameter to **HPE1437_ON** when *sync* = **HPE1437_FRONT** causes the E1437 to drive the front panel ADC clock; or if *sync* = **HPE1437_REAR** causes the module to use its ADC clock to drive the VXI backplane in the mainframe in which it resides. **HPE1437_OFF** means that the E1437 is driving neither the front panel nor the backplane and is the correct variable to use for all non-master modules in a multi-module system. Setting this parameter to **HPE1437_BUFFER** allows the ADC clock and SYNC lines from the module's front panel connectors to drive the backplane of a mainframe not containing the master. Only one module per mainframe may be set to **ON** or to **BUFFER**. In multi-module and multi-mainframe systems only one module may be set to **ON** within the entire system. In multi-mainframe systems using backplane clock and sync distribution only one module per any mainframe not containing the master may be set to **BUFFER**.

masterPtr contains the current value of the *master* parameter.

fs provides the module with the frequency of an external sample clock (from >0 to 20600000) connected to the Ext Clk TTL connector. When using an external clock or when a module is a non-master in a multi-module group, the frequency of the ADC clock is unknown by the module. It is the responsibility of the programmer to provide the correct frequency so that library functions dependent on fs will operate properly. This value has no effect if the module is set up to use the internal ADC clock.

fsPtr contains the current value of the sample clock frequency. If the E1437 is set to the internal ADC clock, the value of that clock frequency is returned. If the E1437 is set to the external clock, the last value entered via the **hpe1437_clock_fs** function is returned.

Comments

For more details on the interaction among *source*, *master*, and *sync* with multiple modules and multiple mainframes see Managing multiple modules.

The *master*, *sync*, *source*, and *dsp* parameters are interdependent with legitimate combinations being as follows (along with the resultant DSP clock rates):

MASTER	SYNC	SOURCE	DSP	DSP CLOCK RATE
N/A	OFF	20.x	N/A	Internal Source
N/A	OFF	EXT	ADC	External Source
N/A	OFF	EXT	OSC	20.48
N/A	OFF	EXT_PLL	N/A	20
OFF|BUFFER	FRONT	N/A	ADC	Master ADC
OFF|BUFFER	FRONT	N/A	OSC	20.48
OFF	REAR	N/A	ADC	Master ADC
OFF	REAR	N/A	OSC	20.48
ON	FRONT	20.x	N/A	Internal Source
ON	FRONT	EXT	ADC	External Source
ON	FRONT	EXT	OSC	20.48
ON	FRONT	EXT_PLL	N/A	20
ON	REAR	20.x	N/A	Internal Source
ON	REAR	EXT	ADC	External Source
ON	REAR	EXT	OSC	20.48
ON	REAR	EXT_PLL	N/A	20
BUFFER	REAR	N/A	ADC	Master ADC
BUFFER	REAR	N/A	OSC	20.48

The maximum rate at which data may be transferred to memory is determined by the DSP clock rate: Max bytes/s = 4 × DSP clock rate. In continuous mode the maximum rate is limited to (4 × DSP clock rate) ÷ 2. However, you may successfully perform this type of measurement by adding a level of decimation to reduce the sample rate.

If *fs*>20480000 then *dsp* must=ADC.

Example

The program **multichan.exe** described in Example Programs provides an example of how to correctly set up a multi-module system with synchronous clocks.

Reset Values

sync	OFF
source	20480KHZ
dsp	ADC
master	OFF
fs	20.48 e6

Effect on Active Measurement

Commands in this group, other than those ending in _get and HPE1437_clock_fs, abort any measurement in progress.

See Also

hpe1437_init, hpe1437_filter_setup, hpe1437_data_

hpe1437_close

Closes the module's software connection.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_close**(ViSession ***id***);

Description

hpe1437_close terminates the software connection to the module, deallocates system resources, and places the module in the IDLE state. After this function has been executed the specified *id* identifier is no longer a valid parameter for function calls.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init

hpe1437_data_memsize_get

Returns the module's memory size in megabytes.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_data_memsize_get**(ViSession ***id***, ViPInt16 ***memSizePtr***);

Description

This command allows you to determine whether your module contains standard memory of 8 Mbytes or a larger memory option.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

memSizePtr contains the memory size in number of Megabytes.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init , hpe1437_data_blocksize

hpe1437_data_scale_get

Gets data scale factor.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_data_scale_get**(ViSession ***id***, ViPReal64 ***scalePtr***);

Description

hpe1437_data_scale_get calculates the correct scale factor for raw data using the current data resolution and range. The factor returned by this function is used to multiply raw data to get data in volts.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

scalePtr contains the calculated scale factor with which to scale raw data to volts.

NOTE

If hpe1437_input_range_auto is pending or in progress this command returns an error.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_ - , hpe1437_read_raw -

hpe1437_data_setup

hpe1437_data_setup sets all format and data output flow parameters. This description also includes information on the following functions which set or query the format and flow parameters individually:

hpe1437_data_append_status appends status information to a data block.

hpe1437_data_append_status_get gets the append status state

hpe1437_data_blocksize determines the size of the output data block.

hpe1437_data_blocksize_get gets the output data block size

hpe1437_data_mode selects block mode or continuous mode.

hpe1437_data_mode_get gets the data mode

hpe1437_data_port selects VME bus or local bus output port.

hpe1437_data_port_get gets the output port designation

hpe1437_data_resolution selects 16 or 32 bits data resolution.

hpe1437_data_resolution_get gets the data resolution

hpe1437_data_type selects real or complex output data.

hpe1437_data_type_get gets output data type

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_data_setup**(ViSession ***id***, ViInt16 ***dType***, ViInt16 ***resolution***, ViInt16 ***mode***, ViInt32 ***blocksize***, ViInt16 ***appendStatus***, ViInt16 ***port***);

ViStatus **hpe1437_data_append_status**(ViSession ***id***, ViInt16 ***appendStatus***);

ViStatus **hpe1437_data_append_status_get**(ViSession ***id***, ViPInt16 ***appendStatusPtr***);

ViStatus **hpe1437_data_blocksize**(ViSession ***id***, ViInt32 ***blocksize***);

ViStatus **hpe1437_data_blocksize_get**(ViSession ***id***, ViPint32 ***blocksizePtr***);

ViStatus **hpe1437_data_mode**(ViSession ***id***, ViInt16 ***mode***);

ViStatus **hpe1437_data_mode_get**(ViSession ***id***, ViPInt16 ***modePtr***);

ViStatus **hpe1437_data_port**(ViSession ***id***, ViInt16 ***port***);

ViStatus **hpe1437_data_port_get**(ViSession ***id***, ViPInt16 ***portPtr***);

ViStatus **hpe1437_data_resolution**(ViSession ***id***, ViInt16 ***resolution***);

ViStatus **hpe1437_data_resolution_get**(ViSession ***id***, ViPInt16 ***resolutionPtr***);

ViStatus **hpe1437_data_type**(ViSession ***id***, ViInt16 ***dType***);

ViStatus **hpe1437_data_type_get**(ViSession ***id***, ViPInt16 ***dTypePtr***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

dType determines whether the E1437 collects and returns real or complex data. Setting this parameter to **HPE1437_REAL** causes only the real part of the data to be returned for each sample. **HPE1437_COMPLEX** causes the real data followed by the imaginary data to be returned in each sample. Normally, if the frequency set with the **hpe1437_frequency_setup** function is zero, the type should be set to **HPE1437_REAL** since the imaginary component of each sample is zero anyway. When non-zero center frequencies are used the type should normally be set to **HPE1437_COMPLEX**. Otherwise the imaginary component of the signal will be lost.

dTypePtr points to the current value of the *dType* parameter.

resolution selects data resolution of either 16 or 32 bits by using *resolution* values of **HPE1437_16BIT** or **HPE1437_32BIT** respectively. Choosing 16-bit precision allows for more samples in the FIFO memory. Choosing 32 bits allows more dynamic range. Because of the broadband white noise present on the input of the analog-to-digital converter, it is normally sufficient to use 16 bit resolution whenever the **hpe1437_filter_setup** function specifies a signal bandwidth greater than 250 kHz. For narrower bandwidths much of the broadband white noise is filtered out, resulting in lower noise in the output data. To take advantage of this lower noise, the 32-bit data resolution should be used.

resolutionPtr contains the current value of the *resolution* parameter.

mode selects whether the E1437's data collection operates in block mode or continuous mode. **HPE1437_BLOCK** selects block transfer mode in which the measurement is halted after each block of data. To start collection of the next data block the module must be armed and triggered again. This mode is used whenever each block of data is to be associated with an individual trigger "event". **HPE1437_CONTINUOUS** means that a single arm and trigger event starts a measurement which runs continuously with no gaps between output data blocks. As long as the data is read out fast enough to prevent overflow in the output FIFO, the measurement will continue. The continuous mode is useful for continuous signal processing applications where data gaps are unacceptable.

modePtr contains the current value of the *mode* parameter.continuous mode;selecting

blocksize determines the number of sample points in each output data block. The range of available block sizes depends on the number of bytes required for each sample. The command accepts any number between 1 and memory size (in bytes)/2. The actual number used is the first integer power of 2 equal to or larger than the requested *blocksize*. If the requested block size falls outside the range shown in the table the closest valid value will be used and a status register flag (bit 6) will be set indicating a setup error. If a subsequent change in another parameter permits a block size closer to the originally requested value, the module will adjust the block size to that value.

The following table summarizes the available block sizes for each setting of the dType and resolution parameters.

data port	data type	resolution	bytes per Sample	min block size	max block size (with standard 8 MByte memory)*
vme	real	16	2	3	4,194,304
vme	real	32	4	2	2,097,152
vme	complex	16	4	2	2,097,152
vme	complex	32	8	1	1,048,576
lbus	real	16	2	6	4,194,304
lbus	real	32	4	3	2,097,152
lbus	complex	16	4	3	2,097,152
lbus	complex	32	8	2	1,048,576

* For optional additional memory, multiply by the appropriate memory size multiplier. For example, for 32 MByte memory option multiply max block size by 4.

NOTE Block size does not need to be a power of two. Considerably more samples may need to be taken in order to set the block available status bit.

blocksizePtr contains the current value of the *blocksize* parameter. The returned value will be closest valid value to the requested blocksize.

appendStatus selects whether or not status information is appended to a data block. Specifying **HPE1437_ON** means that an extra byte of status information is appended to the end of each data block to indicate whether an ADC overload or error occurred during the collection of that block of data. In this status byte, Bit 0 will be set if an ADC overload occurred and bit 1 will be set for an ADC error. The other bits are undefined. When the appended byte is transferred via the VME backplane, the byte is located in the lower 8 bits of the 16 bit word after the end of the sampled data block. The upper 8 bits are undefined. When the appended byte is output via the local bus (as a 32-bit word), it is marked as the last byte of a transfer block. This status byte should be read separately from any block read operations in order to not affect the alignment of subsequent elements. **HPE1437_OFF** disables this feature.

appendStatusPtr contains the current value of the *status* parameter.

port determines which output port is used to take data from the E1437 module. Setting *port* to **HPE1437_VME** means the data is to be output using standard VME register reads. Setting *port* to **HPE1437_LBUS** means the data is to be output as a byte-serial data stream via the VXI local bus. When using the local bus port the module immediately to the right of the E1437 must be capable of receiving the local bus byte sequence. The following table summarizes the output word or byte sequence for each combination of *dType*, *resolution*, and *port* parameters:

type	resolution	port	sequence
real	16BIT	VME	R0[15:0],R1[15:0],...
complex	16BIT	VME	R0[15:0],Q0[15:0],R1[15:0],Q1[15:0],...
real	32BIT	VME	R0[31:16],R0[15:0],R1[31:16],R1[15:0],...
complex	32BIT	VME	R0[31:16],R0[15:0],Q0[31:16],Q0[15:0], R1[31:16]...
real	16BIT	LBUS	R0[15:8],R0[7:0],R1[15:8],R1[7:0],...
complex	16BIT	LBUS	R0[15:8],R0[7:0],Q0[15:8],Q0[7:0], R1[15:8]..
real	32BIT	LBUS	R0[31:24],R0[23:16],R0[15:8],R0[7:0], R1[31:24],...
complex	32BIT	LBUS	R0[31:24],R0[23:16],R0[15:8],R0[7:0], Q0[31:24],Q0[23:16],Q0[15:8],Q0[7:0], R1[31:24],...

portPtr contains the current value of the *port* parameter.

Comments

The maximum rate at which data may be transferred to memory is determined by the DSP clock rate: Max bytes/s. = 4 × DSP clock rate. In continuous mode the maximum rate is limited to (4 × DSP clock rate) ÷ 2. However, you may successfully perform this type of measurement by adding a level of decimation to reduce the sample rate.

A limitation also applies to 32-bit, complex data transfers. Because this type of transfer cannot be made at the full sample rate, a level of decimation must be added in order to reduce the sample rate.

The following table summarizes the relationship between data parameter combinations, decimation, filter bandwidth, and whether the combination permits block or continuous measurements:

Resolution	Type	Decimation	Filter BW	Block	Continuous	Sample rate (MBytes)
16	Complex	False	0 or 1	Yes	No	40
32	Real	False	0 or 1	Yes	No	80
32	Complex	False	0 or 1	No	No	40
32	Complex	True	0 or 1	Yes	No	40
32	Complex	False	2	Yes	No	40
All other combinations				Yes	Yes	<40

Reset Values

dType	REAL
resolution	32BIT
mode	BLOCK
blocksize	1024
appendStatus	OFF
port	VME

Effect on Active Measurement

With the exception of the commands ending in **_get**, all commands in the group abort any measurement in progress when any parameter value is changed.

See Also

hpe1437_init, hpe1437_frequency_setup, hpe1437_filter_decimate, hpe1437_meas_control, hpe1437_clock_dsp

hpe1437_error_message

Returns error information obtained from function calls.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_error_message**(ViSession ***id***, ViStatus ***errNum***, ViPString ***errMessage***);

Description

hpe1437_error_message takes an error return value generated by a function and translates it to a readable string. This function includes host errors as well as firmware errors.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

errNum represents the instrument numeric error code.

errMessage represents the error message string up to 80 characters long.

NOTE

If you are using this function in Visual Basic you should allocate memory for the return string. For example:

```
DIM VarName as String *80
```

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_error_queryPAGE 26

hpe1437_error_query

Queries the module for the first error in the queue.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_error_query**(ViSession ***id***, ViPint32 ***errNumPtr***, ViPString ***errMessage***);

Description

hpe1437_error_query queries the module for the oldest error and returns the corresponding error message. This function does not trap host errors.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

errNumPtr contains the instrument numeric error code.

errMessagePtr contains the error message string up to 80 characters long. This message also indicates what function call generated the error.

NOTE

If you are using this function in Visual Basic you should allocate memory for the return string. For example:

```
DIM VarName as String *80
```

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_error_message

hpe1437_filter_resp_get

Returns the module's complex frequency response.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_filter_resp_get**(ViSession ***id***, ViReal64 ***resp***[ ], ViInt32 ***n***, ViReal64 ***fmin***, ViReal64 ***fmax***);

Description

This function uses the current filter and center frequency settings to return the complex frequency response. The requested number of samples are equally spaced from the requested minimum frequency to the requested maximum frequency.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

resp returns the response in the format:

```
resp(re0, im0, re1, im1,..., re(n-1), im(n-1)
```

n is the number of samples desired.

fmin is the minimum frequency in Hertz.

fmax is the maximum frequency in Hertz.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_filter_setup, hpe1437_frequency_setup

hpe1437_filter_setup

hpe1437_filter_setup sets the digital filter bandwidth and decimation filter parameters. This description also includes information on the following functions which set or query the decimation filter parameters individually

hpe1437_filter_decimate selects an extra factor of 2 decimation.

hpe1437_filter_decimate_get gets current state of extra decimation

hpe1437_filter_bw selects a signal filter bandwidth.

hpe1437_filter_bw_get gets the signal filter bandwidth

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_filter_setup**(ViSession ***id***, ViInt16 ***sigBw***, ViInt16 ***decimate***);

ViStatus **hpe1437_filter_decimate** (ViSession ***id***, ViInt16 ***decimate***);

ViStatus **hpe1437_filter_decimate_get**(ViSession ***id***, ViPInt16 ***decimatePtr***);

ViStatus **hpe1437_filter_bw** (ViSession ***id***, ViInt16 ***sigBw***);

ViStatus **hpe1437_filter_bw_get**(ViSession ***id***, ViPInt16 ***sigBwPtr***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

sigBw selects an alias protected signal filter bandwidth that is roughly *fs*/(2.56 * 2^(sigBw)) where *fs* is the ADC sample frequency. In zoom applications, where the center frequency is generally not zero, the zoom filter bandwidth is centered on the frequency programmed with the **hpe1437_frequency_setup** function. For baseband measurements the filter may equivalently be considered as a low pass filter of approximately bandwidth *fs*/(2.56 * 2^(sigBw)) since the negative frequencies are generally of no interest. The valid range of *sigBw* is 0 through 24. When *sigBw* = 0, no digital filtering is applied to the signal and the module relies on the analog anti-alias filter to limit the signal bandwidth to *fs*/2.56.

To more accurately calculate the bandwidth use the calculation *fs* * k/2^(sigBw) where:

k=.36 for .25 dB bandwidth

k=.44 for 3 dB bandwidth

k=.5 for 15 dB bandwidth

k=.62 for 110 dB bandwidth

For even more accuracy use the **hpe1437_filter_resp_get** function.

sigBwPtr contains the current value of the *sigBw* parameter.

decimate selects the data output sample rate. When this parameter is set to **HPE1437_OFF** the output sample rate is: *fs* when *sigBw*=0 or *fs*/2^(sigBw-1) when *sigBw*>0. When decimate is set to **HPE1437_ON** the output sample rate is reduced by an additional factor of two by discarding alternate samples. You would normally want to add the extra level of decimation in order to increase the displayed span.

CAUTION

Turning decimation ON when *sigBw*=0 results in aliasing (garbage data) due to upper limit of the sampling frequency.

Comments

To ensure full alias-free operation the analog anti-alias filter (set by the **hpe1437_input_alias_filter** function) should be ON unless the application inherently bandlimits the input signal to less than *fs*/2. The analog anti-alias filter has a fixed bandwidth and thus is fully effective only when *fs*≥20 MHz. If a slower external ADC clock is used, an additional analog filter of the appropriate bandwidth may be required for full alias protection.

The decimation process used to reduce the output sample rate is driven from a "decimation counter" which keeps track of which samples to save and which ones to discard for each of the octave bandwidth reduction filter stages. In multi-module systems where synchronous sampling is required, the decimation counters in all the modules must be synchronous with each other. This condition can be forced by using the **hpe1437_filter_sync** function.

The following table summarizes the relationship between data parameter combinations, decimation, filter bandwidth, and whether the particular combination permits block and/or continuous measurements:

Resolution	Type	Decimation	Filter BW	Block	Continuous	Sample rate (MBytes)
16	Complex	False	0 or 1	Yes	No	40
32	Real	False	0 or 1	Yes	No	80
32	Complex	False	0 or 1	No	No	40
32	Complex	True	0 or 1	Yes	No	40
32	Complex	False	2	Yes	No	40
All other combinations				Yes	Yes	<40

Example

Here are some bandwidth and sample rate results using the “k” calculation for bandwidth:

```
=====================================================================
              Fs = 20.48 MHz default internal ADC clock
                        (all data in MHz)
=====================================================================
         | Signal Bandwidth    |          Sample rate
 sigBw   | .25 dB   |  15 dB   |  Decimate OFF  | Decimate ON
_________|__________|__________|________________|_____________________
    0    |  7.37    | 10.24    |     20.48      |   10.24 (see Caution)
    1    |  3.69    |  5.12    |     20.48      |   10.24
    2    |  1.84    |  2.56    |     10.24      |    5.12
    3    |  0.92    |  1.28    |      5.12      |    2.56
    4    |  0.46    |  0.64    |      2.56      |    1.28
           ... Continue to decrease by factors of two ...
```

Reset Values

sigBw 0

decimate OFF

Effect on Active Measurement

With the exception of the commands ending in **_get**, all commands in the group abort any measurement in progress when any parameter value is changed.

See Also

hpe1437_init, hpe1437_clock_fs_get, hpe1437_filter_resp_get, hpe1437_frequency_setup, hpe1437_filter_sync,hpe1437_input_alias_filter, hpe1437_data_mode

hpe1437_filter_sync

Synchronizes the decimation counter.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_filter_sync**(ViSession ***id***);

Description

This function causes the digital decimation counter to be reset by the next SYNC line rising transition. Any measurement in progress is terminated and the module is placed in the idle state. By calling **hpe1437_filter_sync** for every E1437 module using a shared ADC clock, and then calling **hpe1437_meas_control** to cause a SYNC transition, the decimation counters will be started at the same time. Once this is done the decimation counters will stay synchronized as long as the same ADC clock is used. It is not necessary to resynchronize the decimation counters when the digital filter bandwidths are changed.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Comment

If you also want to synchronize frequency or phase, see hpe1437_frequency_sync and multi module information.

Example

The program **multichan.exe** described in Example Programs provides an example of how to correctly set up a multi-module system with synchronous filters.

NOTE

Resetting the decimation counter causes a transient in the digital filters. The transient takes about 30 output sample periods to decay 120 dB. See the impulse response graphs in the specification section for more detail.

Effect on Active Measurement

This command aborts any measurement in progress when any parameter value is changed.

See Also

hpe1437_init, hpe1437_filter_setup, hpe1437_frequency_setup , hpe1437_meas_control

hpe1437_frequency_center_raw

Provides a fast way to set the center frequency.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_frequency_center_raw**(ViSession ***id***, ViInt16 ***coarse***, ViInt32 ***fine***);

Description

hpe1437_frequency_raw sets the center frequency without relying on the internal E1437 microprocessor to do any floating point computations, since the internal microprocessor does not have a floating point co-processor. The resulting center frequency is approximately:

*fs**((*coarse*/2048)+(*fine*/1.024*10^12)) where *fs* is the ADC clock frequency.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

coarse is used to set high frequencies or a low resolution frequency component.

fine is used to set very low frequencies or a high resolution frequency component.

Effect on Active Measurement

These commands do not abort any measurement in progress

See Also

hpe1437_init, hpe1437_frequency_setup, hpe1437_clock_fs_get, hpe1437_data_type, hpe1437_meas_control

hpe1437_frequency_setup

hpe1437_frequency_setup sets all the zoom center frequency parameters. This description also includes information on the following functions which set or query frequency parameters individually:

hpe1437_frequency_cmplxdc selects a complex baseband measurement

hpe1437_frequency_cmplxdc_get gets the state of the baseband measurement mode

hpe1437_frequency_sync prepares the module for a synchronous frequency change

hpe1437_frequency_sync_get gets the state of the synchronous change mode

hpe1437_frequency_center sets the center frequency

hpe1437_frequency_center_get gets the current center frequency

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_frequency_setup**(ViSession ***id***, ViInt16 ***cmplxDc***, ViInt16 ***sync***, ViReal64 ***freq***);

ViStatus **hpe1437_frequency_cmplxdc**(ViSession ***id***, ViInt16 ***cmplxDc***);

ViStatus **hpe1437_frequency_cmplxdc_get**(ViSession ***id***, ViPInt16 ***cmplxDcPtr***);

ViStatus **hpe1437_frequency_sync**(ViSession ***id***, ViInt16 ***sync***);

ViStatus **hpe1437_frequency_sync_get**(ViSession ***id***, ViPInt16 ***syncPtr***);

ViStatus **hpe1437_frequency_center**(ViSession ***id***, ViReal64 ***freq***);

ViStatus **hpe1437_frequency_center_get**(ViSession ***id***, ViPReal64 ***freqPtr***);

Description

hpe1437_frequency_setup sets the center frequency of a zoomed measurement. The center of a frequency band of interest is converted to DC with this function. The frequency transition is phase continuous unless the center frequency is set to zero in which case the transition may be selected either to be phase continuous or phase reset. This function may also be used to synchronously change frequency in multiple-module systems.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

cmplxDc selects either a phase continuous or phase reset transition when the *freq* = 0. **HPE1437_OFF**, combined with a frequency change to zero, causes phase to be reset to zero. **HPE1437_ON**, combined with a frequency change to zero, does not reset the phase, thereby generating a complex DC measurement at baseband. The state of this parameter does not affect any transition where *freq*≠ 0. Whether the real or complex data is saved and ultimately sent to the output port is determined by the **hpe1437_data_type** function.

cmplxDcPtr contains the value of the *cmplxDc* parameter.

sync when set to **HPE1437_OFF** allows an immediate frequency change. In multiple-module systems, setting this parameter to **HPE1437_ON** prepares the modules for a frequency change, but does not actually bring about the change until the next ADC clock corresponding to the next assertion of the shared SYNC signal. The SYNC transition is generated by calling the **hpe1437_meas_control** function. Note that returning *sync* to OFF before the SYNC signal transition has occurred forces an immediate asynchronous frequency change.

syncPtr returns the value of the *sync* parameter.

freq is a number between –0.5 and +0.5, which will be interpreted as a fraction of the sample frequency. *freq* is the desired center frequency divided by the ADC sample frequency. For example, selecting .25 with a sample clock frequency of 20 MHz will yield a center frequency of 5.0 MHz. The ADC sample frequency is returned by the **hpe1437_clock_fs_get** function. Negative frequencies select the negative image of the signal, which is spectrally inverted from the input signal.

freqPtr contains the current actual value of the center frequency (as a fraction of the sample clock frequency).

Comments

Although the *freq* parameter is a double floating point number, its effective resolution is 1/(1024*10^9) or 20 µHz when fs=20.48 MHz. The actual frequency will be set to the nearest available value. This value is returned by the **hpe1437_frequency_center_get** function. In multi-module systems this value represents the pending value rather than the current value when a frequency change is incomplete due to a pending SYNC signal transition,.

In multiple-module systems it is often desirable to force the frequency change to occur synchronously in order to preserve the phase relationship of the LOs. This is accomplished by setting the *sync* parameter to ON for all the modules which are to be changed. See the first example below.

In configurations involving synchronous operation of multiple E1437 modules, the **hpe1437_frequency_setup** function provides a mechanism to force all LOs to the same phase. This can be done by first setting the frequency to zero. See the second example below.

Example

The program **multichan.exe** described in Example Programs provides an example of how to correctly perform synchronous frequency changes in a multi-module system.

Reset Values

cmplxDc	OFF
sync	OFF
freq	0

Effect on Active Measurement

These commands do not abort any measurement in progress

See Also

hpe1437_init, hpe1437_clock_fs_get, hpe1437_data_type , hpe1437_clock_multi_sync, hpe1437_meas_control

hpe1437_init

Initializes the I/O driver for a module.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_init**(ViRsrc ***instrDesc***, ViBoolean ***idQuery***, ViBoolean ***rst***, ViPSession ***id***);

Description

hpe1437_init must be the first routine called when using the E1437 library. It establishes communication with the module and returns a module identification which is used with all subsequent functions involving this module. This function performs whatever initialization the I/O driver needs for the environment in which this library is running.

Parameters

instrDesc specifies the interface and logical address. This descriptor varies depending on your I/O library.

An example of the descriptor form for a VTL I/O library is:
VXI[Board]::*VXIlogical address* [::INSTR]

An example of the descriptor form for a SICL I/O library is:
vxi,*logical address*

idQuery set to **HPE1437_ON** verifies the identity of the instrument by checking the manufacturer ID and model number in the module's VXI register set. If set to **HPE1437_OFF** the function does not verify the module's identity. It is helpful to disable the ID query if you want to use the driver with a similar module but do not need to modify the driver source code.

rst places the module in the reset state when set to **HPE1437_ON**. If set to **HPE1437_OFF**, the function disables the reset. Disabling the reset is useful for debugging in cases where resetting would take the instrument out of the state you want to test.

id is a pointer to the VXI instrument Session identifier returned by this function for the module. This identifier is then used with all other functions which address this module.

Comments

If you receive a resource descriptor error, see your I/O library documentation to determine the correct descriptor form.

Effect on Active Measurement

This command aborts any measurement in progress.

See Also

hpe1437_close

hpe1437_input_autozero

Nulls out the input DC offset voltage

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_input_autozero**(ViSession ***id***);

Description

hpe1437_input_autozero updates a table of DC offset corrections to be used with each input setup condition. The applicable correction from this table is automatically added to the input offset parameter to achieve the correct DC offset value. Because of the length of time needed to execute this function, it is not automatically called when the module is reset. Thus, the user program is responsible for explicitly initiating the autozero. This function should be called at least once after the temperature of the module has stabilized. The interval between calls after that depends on the importance of DC accuracy in the user application. It is not necessary to call the autozero function for every change of input setup parameters since the correction table maintains values for all setup conditions.

NOTE

Calling hpe1437_input_autozero aborts any measurement already in progress and eliminates LO phase coherence and filter synchronization in a synchronous multi-module system. See the hpe1437_frequency_sync and hpe1437_frequency_sync functions for details on how to re-establish LO phase coherence and filter synchronization.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Effect on Active Measurement

This command aborts any measurement in progress.

See Also

hpe1437_init, hpe1437_input_setup, hpe1437_filter_sync , hpe1437_frequency_sync

hpe1437_input_setup

hpe1437_input_setup sets all the analog input parameters. This description also includes information on the following functions which set or query the input parameters individually:

hpe1437_input_alias_filter selects the built-in analog anti-alias filter

hpe1437_input_alias_filter_get gets the anti-alias filter state

hpe1437_input_coupling selects AC or DC input coupling

hpe1437_input_coupling_get get the input coupling type

hpe1437_input_float selects floating the input connector

hpe1437_input_float_get gets the input connector state

hpe1437_input_range sets the full scale range

hpe1437_input_range_get gets the input range

hpe1437_input_signal selects the input buffer amplifier

hpe1437_input_signal_get gets the input buffer amplifier state

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_input_setup**(ViSession ***id***, ViInt16 ***range***, ViInt16 ***coupling***, ViInt16 ***antiAlias***, ViInt16 ***signal***, ViInt16 ***floatIn***);

ViStatus **hpe1437_input_alias_filter**(ViSession ***id***, ViInt16 ***antiAlias***);

ViStatus **hpe1437_input_alias_filter_get**(ViSession ***id***, ViPInt16 ***antiAliasPtr***);

ViStatus **hpe1437_input_coupling**(ViSession ***id***, ViInt16 ***coupling***);

ViStatus **hpe1437_input_coupling_get**(ViSession ***id***, ViPInt16 ***couplingPtr***);

ViStatus **hpe1437_input_float**(ViSession ***id***, ViInt16 ***floatIn***);

ViStatus **hpe1437_input_float_get**(ViSession ***id***, ViPInt16 ***floatInPtr***);

ViStatus **hpe1437_input_range**(ViSession ***id***, ViInt16 ***range***);

ViStatus **hpe1437_input_range_get**(ViSession ***id***, ViPInt16 ***rangePtr***);

ViStatus **hpe1437_input_signal**(ViSession ***id***, ViInt16 ***signal***);

ViStatus **hpe1437_input_signal_get**(ViSession ***id***, ViPInt16 ***signalPtr***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

range is a range index number between 0 and 9 which is transformed to a full scale voltage value. The corresponding discrete legal values of full scale vary from 0.02 volt to 10.24 volts with factor-of-two steps (.02 × 2^*range*). If *range* is greater than 9 the full scale value used is 10.24 volts. Signal inputs with an absolute value larger than full scale generate an ADC overflow error.

Range	Full scale voltage	Full Scale dBm
0	.02	-24
1	.04	-18
2	.08	-12
3	.16	-6
4	.32	0
5	.64	6
6	1.28	12
7	2.56	18
8	5.12	24
9	10.24	30

rangePtr contains the current value of the range parameter.

NOTE

If a hpe1437_input_range_auto command is pending or in progress it is aborted when an hpe1437_input_range or hpe1437_input_range_get command is received. hpe1437_input_range_get also returns an error if an autorange is pending or in progress.

coupling specifies the AC or DC coupling mode of the input. Using **HPE1437_DC** will connect the input directly to the 50 Ohm buffer amplifier. **HPE1437_AC** inserts a 0.2 µF capacitor between the input connector and the 50 Ohm buffer amplifier.

couplingPtr contains the current value of the coupling parameter for an E1437 or group of E1437s.

antiAlias determines whether or not to use the built-in analog anti-alias filter. **HPE1437_ON** inserts a sharp-cutoff (11-pole) 8 MHz lowpass filter ahead of the analog-to-digital converter. Using **HPE1437_OFF** disables this filter. It is recommended that you leave the filter on at all times to insure bandlimited, anti-aliased data.

antiAliasPtr contains the current value of the state parameter.

signal determines whether or not the input signal is sent to the buffer amplifier. **HPE1437_ON** attaches the input signal to the 50 Ohm buffer amplifier. **HPE1437_OFF** redirects the input signal to a dummy 50 Ohm load, and feeds the buffer amplifier from an internally grounded 50 Ohm source resistance. The signal OFF setting is useful for making reference measurements without the signal applied. When using AC coupling the 0.2 µF capacitor remains between the input connector and its 50 Ohm termination.

signalPtr contains the current value of the *signal* parameter.

floatIn determines whether or not to allow the outer shield of the input connector to float relative to chassis ground. Using **HPE1437_ON** allows the connector to float in order to reduce potential ground loop induced pick-up at low frequencies. Using **HPE1437_OFF** disables floating by attaching the outer shield of the input connector directly to chassis ground. See the specifications section for more details.

floatInPtr contains the current value of the *floatin* parameter.

Comments

To ensure full alias-free operation the analog anti-alias filter should be ON unless the application inherently bandlimits the input signal to less than *fs*/2. The analog anti-alias filter has a fixed bandwidth and thus is fully effective only when *fs*≥20 MHz. If a slower external ADC clock is used, an additional analog filter of the appropriate bandwidth may be required for full alias protection.

When using the analog anti-alias filter, the *range* parameter may need to be set higher than the actual range of the input signal. The reason for this is that step changes of input voltage cause an overshoot and ringing response at the output of the anti-alias filter. The peak overshoot will actually exceed the input voltage step by about 20%. The range setting must accommodate this overshoot to avoid an ADC overflow.

Reset Values

range	10.24
coupling	DC
antialias	ON
signal	ON
floatin	OFF

Effect on Active Measurement

Commands in the group do not abort any measurement in progress when parameter values are changed.

See Also

hpe1437_init, hpe1437_input_range_auto

hpe1437_input_range_auto

Performs auto-ranging.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_input_range_auto**(ViSession ***id***, ViReal64 ***sec***);

Description

hpe1437_input_range_auto sets the range of a E1437 to the lowest value that will not cause an ADC overload to occur. The algorithm will start at the lowest range and move up until there is no ADC overload.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

sec is the time in seconds to take data at each range to insure that an overload is detected. Setting this parameter to 0.0 will result in this time being set automatically according to an algorithm that depends on block size and filter bandwidth.

NOTE

An autorange that is pending or in progress will be aborted if a input_range or another input_range_auto command is received.

Reset Values

sec	0

Effect on Active Measurement

This command does not aborts any measurement in progress.

See Also

hpe1437_init, hpe1437_input_setup

hpe1437_interrupt_restore

Restores the interrupt masks to the setting last programmed with **hpe1437_interrupt_setup**.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_interrupt_restore**(ViSession ***id***);

Description

The interrupt masks set by the **hpe1437_interrupt_setup** function are cleared during the interrupt acknowledge cycle. This function restores the cleared interrupt masks.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_interrupt_setup

hpe1437_interrupt_setup

hpe1437_interrupt_setup sets both interrupt parameters. This description also includes information on the following functions which query the interrupt parameters individually:

hpe1437_interrupt_mask_get gets the interrupt event mask

hpe1437_interrupt_priority_get gets the VME interrupt line

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_interrupt_setup**(ViSession ***id***, ViInt16 ***intrNum***, ViInt16 ***priority***, ViInt16 ***mask***);

ViStatus **hpe1437_interrupt_mask_get**(ViSession ***id***, ViInt16 ***intrNum***, ViPInt16 ***maskPtr***);

ViStatus **hpe1437_interrupt_priority_get**(ViSession ***id***, ViInt16 ***intrNum***, ViPInt16 ***priorityPtr***);

Description

An E1437 has two independent interrupt generators, each capable of interrupting on one of the seven VME interrupt lines when a status condition specified by a mask occurs.

hpe1437_interrupt_setup sets the interrupt mask, priority and which of the two interrupt generators on the E1437 is to be used. The remaining **hpe1437_interrupt_** functions query the mask and priority individually:

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

intrNum is the number of the interrupt generator. The only values accepted are 0 and 1.

mask specifies the mask of events on which to interrupt. This mask is created by ORing together the bits defined in bits 8 through 15 of the status register. The mask parameter format is 0xMM00 where MM represents the maskable upper 8 bits. The lower 8 bits cannot be used for generating interrupts, and therefore must be set to zero in this function call.

priority specifies which of the seven VME interrupt lines to use. The only legal values are 0 through 7. Specifying 0 turns the interrupt off, while 7 is the highest priority.

maskPtr and ***priorityPtr*** contain the current value of the either the interrupt mask or priority parameter.

Comments

The mask is cleared during the interrupt acknowledge cycle. Therefore, the command must be sent again or restored with **hpe1437_interrupt_restore** in order to generate further interrupts.

Example

The program **interupt.exe** described in Example Programs provides an example of how to use interrupts correctly.

Reset Values

priority 0

mask 0

Effect on Active Measurement

The commands in this group do not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_status_getPAGE 56, hpe1437_attrib_get

hpe1437_lbus_mode

Sets the local bus mode. This description also includes the query:

hpe1437_lbus_mode_get gets the current local bus mode.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_lbus_mode**(ViSession ***id***, ViInt16 ***lbusMode***);

ViStatus **hpe1437_lbus_mode_get**(ViSession ***id***, ViPInt16 ***lbusModePtr***);

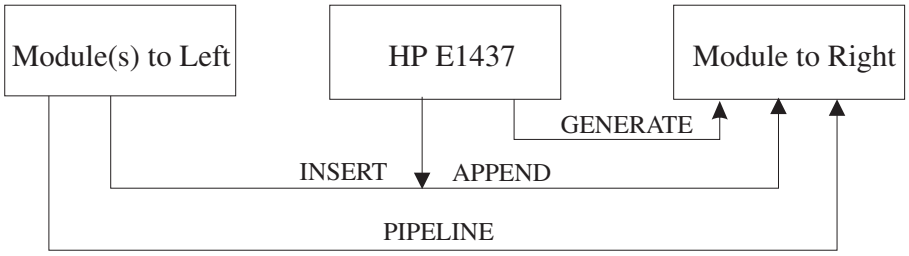
Description

hpe1437_lbus_mode sets the local bus to either generate, append, insert or pipeline data. The data port must be set to the local bus with the **hpe1437_data_port** function before these modes take effect.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

lbusMode selects the transmission mode of the local bus when it is enabled by the **hpe1437_data_port** function. **HPE1437_GENERATE** forces the module at *id* to generate data only, not passing through data from other modules on the local bus. **HPE1437_APPEND** causes the E1437 to pass data through from modules on its left and append its data to the end. **HPE1437_INSERT** causes the E1437 to place its data on the local bus and then pass data through from modules on its left. **HPE1437_PIPELINE** causes the E1437 to pipe data through from modules on its left without appending or inserting its own data. The state of this parameter is unaffected by switching back and forth between the local bus and the VME backplane with the **hpe1437_data_port** function.



lbusModePtr contains the current value of the *lbusMode* parameter.

Reset Values

lbusMode PIPELINE

Effect on Active Measurement

This command aborts any measurement in progress when any parameter value is changed.

See Also

hpe1437_init, hpe1437_data_port

hpe1437_lbus_reset

Resets the local bus. This description also includes the query:

hpe1437_lbus_reset_get - gets the current local bus reset state

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_lbus_reset**(ViSession ***id***, ViInt16 ***lbusReset***);

ViStatus **hpe1437_lbus_reset_get**(ViSession ***id***, ViPInt16 ***lbusResetPtr***);

Description

In order to avoid glitches in the local bus data, the local bus interface has strict requirements as to the order in which modules in a VXI mainframe have their local bus interface reset. Upon powerup or whenever any single module in the mainframe is put into a reset state, all modules should be placed into the reset state from left to right. Then all modules can be take out of reset from left to right.

lbusReset puts the E1437's local bus into reset or takes it out of reset. **HPE1437_ON** puts the E1437's local bus into reset while **HPE1437_OFF** takes the E1437 out of reset.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

lbusResetPtr contains the current value of the *lbusReset* parameter.

Example

When E1437s are used with the E1485 measurement controller, the E1485 must be reset while all of the E1437s are being held in reset to avoid initial glitches in the local bus data. The E1437s should be taken out of reset only after the first **hpe1437_meas_control** release is issued. The correct way to reset the local bus is as follows:

```
lbus_control(LBUS_CTL_RESET, 0);  /* reset the E1485 lbus */
for all id{
        hpe1437_lbus_reset(id, HPE1437_ON);  /* hold HP E1437s in reset */
}
           /*Set LBUS mode for all modules....{
           ....*/}
for all id{
      hpe1437_meas_control(id, HPE1437_RELEASE, HPE1437_ASSERT);
       /* first arming */
     hpe1437_lbus_reset(id, HPE1437_OFF);
       /* remove reset from HP E1437s, has no effect after first time */
}
lbus_control(LBUS_CTL_RESET, 1);  /* unreset the E1485 lbus */
```

Reset Values

lbusReset ON

Effect on Active Measurement

This command does not abort any measurement in progress .

See Also

hpe1437_init

hpe1437_meas_control

Initiates and controls measurements in multi-module systems.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_meas_control**(ViSession ***id***, ViInt16 ***idle***, ViInt16 ***sync***);

Description

hpe1437_meas_control explicitly controls the measurement state.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

idle selects the condition of the IDLE state. **HPE1437_ASSERT** holds the module in the IDLE state. **HPE1437_RELEASE** reverses a previous **HPE1437_ASSERT** or ensures that no forced IDLE is active.

hpe1437_meas_control also changes the state of the SYNC signal, which is used to arm or trigger an E1437 module. In systems containing multiple E1437 modules the SYNC signal is used to arm or trigger all modules simultaneously, and also to synchronize decimation counters and local oscillators among the E1437 modules.

sync selects the state of the sync signal. **HPE1437_ASSERT** causes the module to assert the SYNC signal. **HPE1437_RELEASE** causes the module to release the SYNC signal. When the *sync* parameter of the **hpe1437_clock_setup** function is set to **HPE1437_FRONT** or **HPE1437_REAR**, the SYNC signal is shared with other E1437 modules. If any one of these modules asserts this shared SYNC signal then it becomes asserted for all of them. All modules must release it before the shared SYNC signal is released. Asserting then releasing the SYNC line is used to start a measurement, load local oscillator values, or take a digital filter out of reset. These situations require a SYNC line transition but do not require that the SYNC line be held in a asserted state.

NOTE

When the SYNC line is asserted, it will remain asserted for an adequate number of ADC clock cycles to ensure that the signal effect will have propagated to all the modules in the system. You can determine when the command is completed by looking as the Sync/Idle Complete bit in the Status Register.

Comments

See The Measurement Loop section for details on how a measurement progresses through the four states.

Special conditions prevail during the Measure state. If programmed for block mode operation in the Measure state, the module will assert the SYNC signal (regardless of the **hpe1437_meas_control** *sync* parameter setting) until a complete block of data has been collected and is available to the I/O port. When the shared SYNC signal is released, indicating that all block mode data collection is finished, all block mode modules move synchronously to the idle state. In continuous mode the module releases the SYNC signal immediately after moving into the measure state. This allows the **hpe1437_meas_control** function to manipulate the SYNC signal to cause synchronous changes to LO frequency while a continuous measurement is in progress. In continuous mode a module moves to the idle state only if explicitly programmed to do so or whenever the FIFO data buffer overflows.

In addition to controlling the progression through the four module states, the SYNC signal is used to allow for synchronizing the decimation counters and local oscillators of multiple E1437 modules. This is done by calling **hpe1437_filter_sync** and/or **hpe1437_frequency_sync** prior to asserting SYNC with **hpe1437_meas_control**. This is normally done with the module in the Idle state; however, the center frequency can also be changed in the Measure state with **hpe1437_frequency_sync** if the modules are all programmed for continuous (non-block mode) data collection.

If all modules in a multi-module system are in the Idle state when the **hpe1437_meas_control** *sync* parameter is asserted, the LO frequency will be updated and the next measurement will be armed. If all modules are in the measurement state in continuous mode, the LO frequency will be synchronously updated, and the measurement will continue. In continuous mode you should ensure that all modules are in the same state, either the Idle state or the Measure state, before using **hpe1437_meas_control** to assert SYNC. Otherwise some modules will re-arm while others will continue the current measurement. In block mode the *sync* assertion will be ignored unless all modules are in the Idle state.

The **hpe1437_meas_control** function assures that a single module is in a valid state by checking that the *hardware complete* and *sync valid* bits in the status register are both true. In synchronous multi-module systems you should use the **hpe1437_wait** function for each module to assure a valid state in non-master modules within a synchronous group.

In the case of systems made up of multiple mainframes you must be aware that only modules in mainframe A may assert *sync*. Any *sync* asserted in other mainframes is ignored.

Example

The program **multichan.exe** described in Example Programs provides an example of how to correctly set up a multi-module measurement using **hpe1437_meas_control** to initiate state transitions.

Reset Values

idle	RELEASE
sync	RELEASE

Effect on Active Measurement

This command may or may not abort any measurement in progress when any parameter value is changed, depending on the write value.

See Also

hpe1437_init, hpe1437_status_getPAGE 56, hpe1437_data_, hpe1437_filter_sync, hpe1437_frequency_sync, hpe1437_clock_setup, hpe1437_waitPAGE 64

hpe1437_meas_start

Initiates a measurement in single-module systems.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_meas_start**(ViSession ***id***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

hpe1437_meas_start provides an easy way to initiate a measurement in a single module system. This command moves the module through the IDLE state and the SYNC state while checking the status to assure a valid state.

Comments

See The Measurement Loop section for details on how a measurement progresses through the four states.

The **hpe1437_meas_start** function assures that the module is in a valid state by checking that the *hardware set* and *idle/sync complete* bits in the status register are both true.

Example

The program **acvolts.exe** described in Example Programs provides an example of how to initiate a very simple measurement using **hpe1437_meas_start**.

Effect on Active Measurement

This command aborts any measurement in progress when any parameter value is changed.

See Also

hpe1437_init, hpe1437_status_getPAGE 56, hpe1437_clock_setup, hpe1437_waitPAGE 64

hpe1437_read

Reads scaled 32-bit float data from FIFO . This description also includes the following function:

hpe1437_read64 reads scaled 64-bit float data, implemented specifically for VEE applications.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_read**(ViSession ***id***, ViReal32 ***rec***[ ], ViInt32 ***sampleCount***, ViPInt16 ***overloadPtr***);

ViStatus **hpe1437_read64**(ViSession ***id***, ViReal64 ***rec***[ ], ViInt32 ***sampleCount***, ViPInt16 ***overloadPtr***);

Description

hpe1437_read returns a block of floating point data from the E1437 that has been scaled to be in volts. The function waits for a block of data to be ready before attempting to read the block.

These function can only read data from the VME backplane register. The data port of the E1437 must be set to **HPE1437_VME** by the **hpe1437_data_port** function for these functions to be effective.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

rec is a pointer to the array into which the floating point data is to be placed. Be sure to allocate sufficient storage space at this location to hold the full data record as determined by the *samplecount* parameter. Note that when the module is set to complex data type, the output data record contains 2 ×*samplecount* floating point values. For real data the record contains *samplecount* floating point values.

sampleCount determines the number of sample points to read into the data array. This should never be set larger than the *blocksize* parameter set in the **hpe1437_data_blocksize** function. In continuous data collection mode or when append *status* is turned on, *samplecount* should be set equal to *blocksize* to ensure that the entire data block is read out and that the last word corresponds to *appendStatus*.

overloadPtr is a pointer to a short integer which is set to 1 if an ADC overload was encountered during the collection of the data record and if *appendStatus* is turned on. The value is set to 0 with no overload.

Return Value

Returns the following:

0	the read is complete
1	a read is still in progress and data is not yet available
2	measurement is aborted
3	the module is waiting for a trigger
4	the module is still acquiring pre-trigger data.

Effect on Active Measurement

These commands do not abort any measurement in progress when any parameter value is changed.

See Also

hpe1437_init, hpe1437_data_port, hpe1437_data_blocksize , hpe1437_data_scale_getPAGE 19

hpe1437_read_raw

Reads raw, unscaled data from FIFO

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_read_raw**(ViSession ***id***, ViInt16 ***rec***[ ], ViInt32 ***wordCount***);

Description

hpe1437_read_raw returns a block of raw, unscaled data from the FIFO.

This function can only read data from the VME backplane register. The data port of the E1437 must be set to **HPE1437_VME** by the **hpe1437_data_port** function for this function to be effective.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

rec is a pointer to the array into which the raw data record is to be place. Be sure to allocate sufficient storage space to hold the full data record as determined by the *wordcount* parameter.

wordCount is the number of short data values to read into the data array from the E1437 output FIFO. The maximum *wordcount* depends on the *blocksize*, data *type*, data *resolution*, and *appendStatus* parameter settings according to the following formula:

maxwordcount= W × blocksize + A

where W-=1 for 16-bit real data, W=2 for 32-bit real data, W=2 for 16-bit complex data, W=4 for 32-bit complex data. A=1 if append ADC status is turned on, or A=0 if append ADC status is off. In continuous data collection mode or when append ADC status is turned on, *wordcount* should be set equal to *maxwordcount* to ensure that the entire data block is read out and that the last word corresponds to *appendStatus*.

NOTE

The primary purpose of the hpe1437_read_raw function is to provide the fastest possible way to read blocks of data from the module. It reads data regardless of the instrument state, whether a block of data is available or not. The resulting data ordering is dependent on the data type and resolution. The array may be cast as a long before reading the data to provide whole words.

Effect on Active Measurement

This command does not abort any measurement in progress when any parameter value is changed.

See Also

hpe1437_ - , hpe1437_data_scale_getPAGE 19

hpe1437_reset

Places the module in a known state.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_reset**(ViSession ***id***);

Description

hpe1437_reset returns the module and its internal data structures to the power-up state. This function can be called separately by this function, or may be selected in conjunction with the **hpe1437_init** function.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Comments

The reset values are listed with each command description.

The following are not affected by this command:

- Calibration constants

Effect on Active Measurement

This command aborts any measurement in progress.

See Also

hpe1437_init

hpe1437_revision_query

Returns strings that identify the date of the firmware revision.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_revision_query**(ViSession ***id***, ViString ***driverRev***, ViString ***instRev***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

driverRev returns the date and time of the module's driver revision in the form:

mm-dd-yyyy hh:mm

instRev returns the date, time, and board number of the module's firmware revision in the form:

mm-dd-yyyy hh:mm board#

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init

hpe1437_self_test

Performs a self-test and returns the result of that self test.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_self_test**(ViSession ***id***, ViPInt16 ***testResultPtr***, ViString ***testMessage***);

Description

The E1437 self test includes the following tests:

- Digital: rails the front end to a full scale value then turns on zooming, filtering, and the final decimation to quickly verify those operations.
- Noise: does a quick baseband measurement with the input signal disconnected, and verifies that the front-end noise is within specification.
- Bump: Verifies some front-end levels associated with the analog-to-digital converter.
- Memory: fills the entire DRAM then verifies that all the data is correct.
- Analog: verifies that autozero adjust is working and that the input is triggering.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

testResult contains the instrument numeric error code.

testMessage contains the self test status message string up to 80 characters long.

NOTE

The self-test takes about the following amount of time to complete:

Memory size (MBytes)	Time (min.)
8	1.0
16	1.5
32	2.5
64	4.5

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init

hpe1437_status_get

Reads Status Register information for the module.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_status_get**(ViSession ***id***, ViPInt16 ***statusPtr***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

statusPtr contains the status word. The bits are defined below:

1-0 State: These two bits indicate the current state of the measurement loop as shown in the table below. See the Measurement Loop section for more information about the states.

```
Bits     State
==============
 11     Trigger
 10     Measure
 01       Arm
 00      Idle
```

2 Passed: This bit is always set to 1.

3 Ready: This bit is set whenever the module is operating as a message-based device and is set for Normal operation. See the VXIbus Specifications for more information on the Normal configuration sub-state.

4 ADC Error: This bit is set whenever a hardware error is detected in the ADC. The bit is cleared when the Status register is read.

5 Ext Clk Speed: This bit is set when a measurement has been aborted because the external clock is too fast (over 20.48 MHz) with respect to the DSP clock. This situation only occurs when a fast external ADC clock is used with an internal oscillator DSP clock. This bit is cleared with the first subsequent read.

6 Setup error: An invalid parameter value was requested. If an invalid block size was requested, the closest valid block size is used until a change to an interrelated parameter makes the requested block size valid. If a data resolution, data type, filter bandwidth, or filter decimation parameter was requested which would result in an inability to make a measurement, the previous valid parameter is used until a change to an interrelated parameter makes the requested parameter valid.

7 Sync/Idle Complete: This bit is set when the most recent user-initiated SYNC or IDLE change has propagated through to all modules in a system. The change is a result of asserting SYNC or forcing IDLE via the Control Register or issuing a meas_control command or function.

8 Read Valid: This flag is set whenever there is at least one valid 16-bit data word available to be read via the Data register.

9 Measure Done: This bit is set in continuous mode whenever the size of the data in the FIFO is equal to or greater than the block size register. Check this bit before reading data to insure that a block of data may be transferred without fear of running out of data, thereby holding up the Local bus or VME bus. This bit is set in block mode whenever the module has successfully taken a block size number of samples since the most recent trigger

10 Armed: This bit is set whenever the module is in the Trigger state, or is in the Arm state and has satisfied its pre-trigger requirements. When this bit is set, the module releases the VXI SYNC line. Once all modules release the SYNC line, then all modules go to the Trigger state.

11 FIFO Overflow: This bit set when the FIFO buffer overflows in continuous mode.

12 Overload: This bit is set whenever the ADC converts a sample that exceeds the range of the ADC. The bit is cleared when the Status register is read. Repeated ADC errors may indicate that the module should be recalibrated.

13 Error: This bit is set whenever there is an error in the error queue. It is cleared when the error queue is empty.

14 ModID*: A (1) in this field indicates that the module is not selected via the P2 MODID line. A (0) indicates that the module is selected by a high state on the P2 MODID line.

15 Hardware Set: This bit is set when all commands are complete and the hardware has been set.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init

hpe1437_trigger_delay_actual_get

Returns the actual trigger delay from the most recent trigger event.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_trigger_delay_actual_get**(ViSession ***id***, ViPReal64 ***actualDelayPtr***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

actualDelayPtr contains the returned actual delay from the most recent trigger event and the resulting first output sample time. This delay value provides more accuracy than the *delay* parameter alone since it includes a measurement of the fractional part of the output sample period between the actual trigger event and the next available output sample. The trigger delay accuracy improves to one ADC sample clock period rather than one output sample period. This can result in a substantial improvement in accuracy when narrow bandwidth decimation filtering is used. The **hpe1437_trigger_delay_actual_get** function must be called for each new trigger event that requires precise delay measurement. The actual delay is still expressed in output sample periods, however, it can take on non-integer values.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_trigger_setup

hpe1437_trigger_phase_actual_get

Returns a representation of the phase value of the LO at the trigger point.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_trigger_phase_actual_get**(ViSession ***id***, ViPReal64 ***actualPhasePtr***);

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

actualPhasePtr contains the returned value interpreted as follows:

0 <= value < 1.0

```
where  0 => 0 degrees
      .25 => 90 degrees
      .5 => 180 degrees
```

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_trigger_setup, hpe1437_trigger_phase_capturePAGE 60

hpe1437_trigger_phase_capture

Prepares for LO phase capture in frequency-synchronized, multiple-module zoom measurements.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_trigger_phase_capture**(ViSession ***id***);

Description

Use this function if you intend to subsequently use **hpe1437_trigger_phase_actual_get** to capture the LO phase on the next SYNC assertion. You should send **hpe1437_trigger_phase_capture** to only one module in the system (typically the master) after you have completed all frequency and filter setup functions since those functions take the module out of the phase_capture mode. Therefore, you should call this function just prior to starting the measurement.

When the **hpe1437_frequency_sync** mode is turned off, the **hpe1437_trigger_phase_capture** function is not needed because the module will revert to the phase_capture mode by default.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_trigger_setup, hpe1437_trigger_phase_actual_getPAGE 59, hpe1437_frequency_sync, hpe1437_trigger_delay_actual_get

hpe1437_trigger_setup

hpe1437_trigger_setup sets all triggering parameters. This description also includes information on the following functions which set or query the trigger parameters individually:

hpe1437_trigger_adclevel specifies the trigger threshold for an ADC trigger

hpe1437_trigger_adclevel_get gets the ADC trigger threshold

hpe1437_trigger_delay specifies a pre- or post-trigger delay time

hpe1437_trigger_delay_get gets the trigger delay time

hpe1437_trigger_gen determines whether a module can generate a trigger

hpe1437_trigger_gen_get gets the trigger generation status

hpe1437_trigger_maglevel specifies the trigger threshold for a magnitude trigger

hpe1437_trigger_maglevel_get gets magnitude trigger threshold

hpe1437_trigger_slope selects a positive or negative trigger

hpe1437_trigger_slope_get gets trigger slope

hpe1437_trigger_type determines the trigger type

hpe1437_trigger_type_get gets trigger type

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_trigger_setup**(ViSession ***id***, ViInt16 ***tType***, ViInt32 ***delay***, ViInt16 ***adcLevel***, ViInt16 ***magLevel***, ViInt16 ***slope***, ViInt16 ***gen***);

ViStatus **hpe1437_trigger_adclevel**(ViSession ***id***, ViInt16 ***adcLevel***);

ViStatus **hpe1437_trigger_adclevel_get**(ViSession ***id***, ViPInt16 ***adcLevelPtr***);

ViStatus **hpe1437_trigger_delay**(ViSession ***id***, ViInt32 ***delay***);

ViStatus **hpe1437_trigger_delay_get**(ViSession ***id***, ViPint32 ***delayPtr***);

ViStatus **hpe1437_trigger_gen**(ViSession ***id***, ViInt16 ***gen***);

ViStatus **hpe1437_trigger_gen_get**(ViSession ***id***, ViPInt16 ***genPtr***);

ViStatus **hpe1437_trigger_maglevel**(ViSession ***id***, ViInt16 ***magLevel***);

ViStatus **hpe1437_trigger_maglevel_get**(ViSession ***id***, ViPInt16 ***magLevelPtr***);

ViStatus **hpe1437_trigger_slope**(ViSession ***id***, ViInt16 ***slope***);

ViStatus **hpe1437_trigger_slope_get**(ViSession ***id***, ViPInt16 ***slopePtr***);

ViStatus **hpe1437_trigger_type**(ViSession ***id***, ViInt16 ***tType***);

ViStatus **hpe1437_trigger_type_get**(ViSession ***id***, ViPInt16 ***tTypePtr***);

Description

An E1437 can be triggered to collect data in a variety of ways. The trigger can be internally generated or can come from an external source. Multiple modules can be triggered synchronously. A variable pre- and post-trigger delay can be programmed for data collection. The slope and level of the trigger point on a signal can be selected. The source of the internal trigger can be either the output of the ADC or the magnitude of the complex output of the decimation filter.

hpe1437_trigger_setup is the function that sets all trigger parameters at once. An E1437 will generate a trigger only when it is in the TRIGGER state and the SYNC line on the VXI backplane is released. When a trigger is generated, the E1437 will release the SYNC line.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

tType determines the trigger source. **HPE1437_ADC** generates a trigger based on the raw data samples from the ADC. **HPE1437_MAG** generates a trigger based on the log magnitude of the signal after it has been filtered to a selectable bandwidth around the center frequency established by the **hpe1437_frequency_setup** function. **HPE1437_EXTERNAL** uses transitions on the signal applied to the BNC external trigger connector on the front panel. **HPE1437_USER** disables the module from any event-driven trigger generation though it is still possible to force the module to trigger a measurement by pulling the SYNC line once the module is in the trigger state. You may do this by calling the **hpe1437_meas_start** function, waiting for the module to reach the trigger state, then triggering the measurement by using **hpe1437_meas_control** to pull the SYNC line. **HPE1437_IMMEDIATE** triggers a measurement immediately upon entering the trigger state.

NOTE

In multi-module systems all modules should be of the same type in order to have the same actual delay.

tTypePtr contains the current value of *tType*.

delay is the time delay, in units of output samples, between when a trigger is received and the first data point in the output data. Negative values indicate a pre-trigger condition, where samples prior to the trigger event are included in the output data. The amount of pre-trigger delay is limited to the number of samples which can be saved in the 8 Mbyte buffer memory. See the **hpe1437_data_setup** function description for the number of bytes used per sample. The delay limits depend on the data type as follows:

Trigger Delay (DRAM size in bytes)			
	32 bit complex	32 bit real / 16 bit complex	16 bit real
Post-trigger	16,777,116	33,554,332	67,108,764
Pre-trigger	132-DRAMsize/8	164-DRAMsize/4	228-DRAMsize/2

If *delay* is <132-DRAMsize/8 or >16,777,116 a bad parameter error will be set. However, the delay is still programmed in order to accommodate the valid setups generated by other data types.

delayPtr contains the current value of the of *delay*.

adcLevel is used to set the triggering signal threshold when using the ADC trigger source. This threshold is (full scale ×*adclevel*/256), where –256 ≤ *adclevel* ≤ 255. There is hysteresis around the threshold in order to prevent multiple triggers from a single threshold crossing.

adcLevelPtr contains the current value of the of the *adclevel* parameter.

magLevel is used to set the triggering threshold when using the mag trigger source. The threshold is (+0.3762874 ×*maglevel*)dB relative to full scale signal, where –349 ≤ *maglevel* ≤ 19.

magLevelPtr contains the current value of the *maglevel* parameter.

slope selects the edge of the trigger source on which a trigger occurs. **HPE1437_POSITIVE** sets triggering on the positive slope and **HPE1437_NEGATIVE** on the negative slope.

slopePtr contains the current value of the of the trigger slope.

gen determines whether a module may generate a trigger. **HPE1437_ON** enables triggering. **HPE1437_OFF** disables triggering. This is useful in multi-module systems with the same trigger type where you want only certain module(s) to generate a trigger.

genPtr contains the current value of the of the *gen* parameter.

Reset Values

tType	IMMEDIATE
delay	0
adcLevel	0
magLevel	–128
slope	POSITIVE
gen	ON

Effect on Active Measurement

The commands in this group do not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_frequency_setup, hpe1437_data_, hpe1437_filter_decimate, hpe1437_meas_start hpe1437_meas_control, hpe1437_trigger_delay_actual_get

hpe1437_wait

Facilitates the synchronization and control of multi-module systems.

VXI*plug&play* Syntax

#include "hpe1437.h"

ViStatus **hpe1437_wait**(ViSession ***id***);

Description

This function assures that all slave modules are completely set up before issuing measurement control commands to the master module. Prior to calling **hpe1437_meas_control** for the master module in multi-module systems, you should call **hpe1437_wait** for each other module within the related synchronous group to which you have previously sent commands. The function performs a continuous loop which polls the status register of the indicated module until the *hardware complete* and *sync/idle complete* bits are both true.

CAUTION

This an endless loop which assumes that the firmware will eventually set both bits.

You do not need to call **hpe1437_wait** for single modules or non-synchronous groups since the **hpe1437_meas_control** and **hpe1437_meas_start** functions perform an implicit wait.

Parameters

id is the VXI instrument session pointer returned by the **hpe1437_init** function.

Effect on Active Measurement

This command does not abort any measurement in progress.

See Also

hpe1437_init, hpe1437_meas_start hpe1437_meas_controlPAGE 47

VXI*plug&play* Quick Reference

ViStatus **hpe1437_attrib_get**(ViSession ***id***, ViInt16 ***attrib***, ViPint32 ***value***)

ViStatus **hpe1437_clock_setup**(ViSession ***id***, ViInt16 ***sync***, ViInt16 ***source***, ViInt16 ***dsp***, ViInt16 ***master***, ViReal64 ***fs***);

ViStatus **hpe1437_clock_dsp**(ViSession ***id***, ViInt16 ***dsp***);

ViStatus **hpe1437_clock_dsp_get**(ViSession ***id***, ViPInt16 ***dspPtr***);

ViStatus **hpe1437_clock_fs**(ViSession ***id***, ViReal64 ***fs***);

ViStatus **hpe1437_clock_fs_get(**ViSession ***id***, ViPReal64 ***fsPtr***);

ViStatus **hpe1437_clock_master**(ViSession ***id***, ViInt16 ***master***);

ViStatus **hpe1437_clock_master_get**(ViSession ***id***, ViPInt16 ***masterPtr***);

ViStatus **hpe1437_clock_multi_sync**(ViSession ***id***, ViInt16 ***sync***);

ViStatus **hpe1437_clock_multi_sync_get**(ViSession ***id***, ViPInt16 ***syncPtr***);

ViStatus **hpe1437_clock_source**(ViSession ***id***, ViInt16 ***source***);

ViStatus **hpe1437_clock_source_get**(ViSession ***id***, ViPInt16 ***sourcePtr***);

ViStatus **hpe1437_close**(ViSession ***id***);

ViStatus **hpe1437_data_memsize_get**(ViSession ***id***, ViPInt16 ***memSizePtr***);

ViStatus **hpe1437_data_scale_get**(ViSession ***id***, ViPReal64 ***scalePtr***);

ViStatus **hpe1437_data_setup**(ViSession ***id***, ViInt16 ***dType***, ViInt16 ***resolution***, ViInt16 ***mode***, ViInt32 ***blocksize***, ViInt16 ***appendStatus***, ViInt16 ***port***);

ViStatus **hpe1437_data_append_status**(ViSession ***id***, ViInt16 ***appendStatus***);

ViStatus **hpe1437_data_append_status_get**(ViSession ***id***, ViPInt16 ***appendStatusPtr***);

ViStatus **hpe1437_data_blocksize**(ViSession ***id***, ViInt32 ***blocksize***);

ViStatus **hpe1437_data_blocksize_get**(ViSession ***id***, ViPint32 ***blocksizePtr***);

ViStatus **hpe1437_data_mode**(ViSession ***id***, ViInt16 ***mode***);

ViStatus **hpe1437_data_mode_get**(ViSession ***id***, ViPInt16 ***modePtr***);

ViStatus **hpe1437_data_port**(ViSession ***id***, ViInt16 ***port***);

ViStatus **hpe1437_data_port_get**(ViSession ***id***, ViPInt16 ***portPtr***);

ViStatus **hpe1437_data_resolution**(ViSession ***id***, ViInt16 ***resolution***);

ViStatus **hpe1437_data_resolution_get**(ViSession ***id***, ViPInt16 ***resolutionPtr***);

ViStatus **hpe1437_data_type**(ViSession ***id***, ViInt16 ***dType***);

ViStatus **hpe1437_data_type_get**(ViSession ***id***, ViPInt16 ***dTypePtr***);

ViStatus **hpe1437_error_message**(ViSession ***id***, ViStatus ***errNum***, ViPString ***errMessage***);

ViStatus **hpe1437_error_query**(ViSession ***id***, ViPint32 ***errNumPtr***, ViPString ***errMessage***);

ViStatus **hpe1437_filter_resp_get**(ViSession ***id***, ViReal64 ***resp***[ ], ViInt32 ***n***, ViReal64 ***fmin***, ViReal64 ***fmax***);

ViStatus **hpe1437_filter_setup**(ViSession ***id***, ViInt16 ***sigBw***, ViInt16 ***decimate***);

ViStatus **hpe1437_filter_decimate** (ViSession ***id***, ViInt16 ***decimate***);

ViStatus **hpe1437_filter_decimate_get**(ViSession ***id***, ViPInt16 ***decimatePtr***);

ViStatus **hpe1437_filter_bw** (ViSession ***id***, ViInt16 ***sigBw***);

ViStatus **hpe1437_filter_bw_get**(ViSession ***id***, ViPInt16 ***sigBwPtr***);

ViStatus **hpe1437_filter_sync**(ViSession ***id***);

ViStatus **hpe1437_frequency_center_raw**(ViSession ***id***, ViInt16 ***coarse***, ViInt32 ***fine***);

ViStatus **hpe1437_frequency_setup**(ViSession ***id***, ViInt16 ***cmplxDc***, ViInt16 ***sync***, ViReal64 ***freq***);

ViStatus **hpe1437_frequency_cmplxdc**(ViSession ***id***, ViInt16 ***cmplxDc***);

ViStatus **hpe1437_frequency_cmplxdc_get**(ViSession ***id***, ViPInt16 ***cmplxDcPtr***);

ViStatus **hpe1437_frequency_sync**(ViSession ***id***, ViInt16 ***sync***);

ViStatus **hpe1437_frequency_sync_get**(ViSession ***id***, ViPInt16 ***syncPtr***);

ViStatus **hpe1437_frequency_center**(ViSession ***id***, ViReal64 ***freq***);

ViStatus **hpe1437_frequency_center_get**(ViSession ***id***, ViPReal64 ***freqPtr***);

ViStatus **hpe1437_init**(ViRsrc ***instrDesc***, ViBoolean ***idQuery***, ViBoolean ***rst***, ViPSession ***id***);

ViStatus **hpe1437_input_autozero**(ViSession ***id***);

ViStatus **hpe1437_input_setup**(ViSession ***id***, ViInt16 ***range***, ViInt16 ***coupling***, ViInt16 ***antiAlias***, ViInt16 ***signal***, ViInt16 ***floatIn***);

ViStatus **hpe1437_input_alias_filter**(ViSession ***id***, ViInt16 ***antiAlias***);

ViStatus **hpe1437_input_alias_filter_get**(ViSession ***id***, ViPInt16 ***antiAliasPtr***);

ViStatus **hpe1437_input_coupling**(ViSession ***id***, ViInt16 ***coupling***);

ViStatus **hpe1437_input_coupling_get**(ViSession ***id***, ViPInt16 ***couplingPtr***);

ViStatus **hpe1437_input_float**(ViSession ***id***, ViInt16 ***floatIn***);

ViStatus **hpe1437_input_float_get**(ViSession ***id***, ViPInt16 ***floatInPtr***);

ViStatus **hpe1437_input_range**(ViSession ***id***, ViInt16 ***range***);

ViStatus **hpe1437_input_range_get**(ViSession ***id***, ViPInt16 ***rangePtr***);

ViStatus **hpe1437_input_signal**(ViSession ***id***, ViInt16 ***signal***);

ViStatus **hpe1437_input_signal_get**(ViSession ***id***, ViPInt16 ***signalPtr***);

ViStatus **hpe1437_input_range_auto**(ViSession ***id***, ViReal64 ***sec***);

ViStatus **hpe1437_interrupt_restore**(ViSession ***id***);

ViStatus **hpe1437_interrupt_setup**(ViSession ***id***, ViInt16 ***intrNum***, ViInt16 ***priority***, ViInt16 ***mask***);

ViStatus **hpe1437_interrupt_mask_get**(ViSession ***id***, ViInt16 ***intrNum***, ViPInt16 ***maskPtr***);

ViStatus **hpe1437_interrupt_priority_get**(ViSession ***id***, ViInt16 ***intrNum***, ViPInt16 ***priorityPtr***);

ViStatus **hpe1437_lbus_mode**(ViSession ***id***, ViInt16 ***lbusMode***);

ViStatus **hpe1437_lbus_mode_get**(ViSession ***id***, ViPInt16 ***lbusModePtr***);

ViStatus **hpe1437_lbus_reset**(ViSession ***id***, ViInt16 ***lbusReset***);

ViStatus **hpe1437_lbus_reset_get**(ViSession ***id***, ViPInt16 ***lbusResetPtr***);

ViStatus **hpe1437_meas_control**(ViSession ***id***, ViInt16 ***idle***, ViInt16 ***sync***);

ViStatus **hpe1437_meas_start**(ViSession ***id***);

ViStatus **hpe1437_read**(ViSession ***id***, ViReal32 ***rec***[ ], ViInt32 ***sampleCount***, ViPInt16 ***overloadPtr***);

ViStatus **hpe1437_read64**(ViSession ***id***, ViReal64 ***rec***[ ], ViInt32 ***sampleCount***, ViPInt16 ***overloadPtr***);

ViStatus **hpe1437_read_raw**(ViSession ***id***, ViInt16 ***rec***[ ], ViInt32 ***wordCount***);

ViStatus **hpe1437_reset**(ViSession ***id***);

ViStatus **hpe1437_revision_query**(ViSession ***id***, ViString ***driverRev***, ViString ***instRev***);

ViStatus **hpe1437_self_test**(ViSession ***id***, ViPInt16 ***testResultPtr***, ViString ***testMessage***);

ViStatus **hpe1437_status_get**(ViSession ***id***, ViPInt16 ***statusPtr***);

ViStatus **hpe1437_trigger_delay_actual_get**(ViSession ***id***, ViPReal64 ***actualDelayPtr***);

ViStatus **hpe1437_trigger_phase_actual_get**(ViSession ***id***, ViPReal64 ***actualPhasePtr***);

ViStatus **hpe1437_trigger_phase_capture**(ViSession ***id***);

ViStatus **hpe1437_trigger_setup**(ViSession ***id***, ViInt16 ***tType***, ViInt32 ***delay***, ViInt16 ***adcLevel***, ViInt16 ***magLevel***, ViInt16 ***slope***, ViInt16 ***gen***);

ViStatus **hpe1437_trigger_adclevel**(ViSession ***id***, ViInt16 ***adcLevel***);

ViStatus **hpe1437_trigger_adclevel_get**(ViSession ***id***, ViPInt16 ***adcLevelPtr***);

ViStatus **hpe1437_trigger_delay**(ViSession ***id***, ViInt32 ***delay***);

ViStatus **hpe1437_trigger_delay_get**(ViSession ***id***, ViPint32 ***delayPtr***);

ViStatus **hpe1437_trigger_gen**(ViSession ***id***, ViInt16 ***gen***);

ViStatus **hpe1437_trigger_gen_get**(ViSession ***id***, ViPInt16 ***genPtr***);

ViStatus **hpe1437_trigger_maglevel**(ViSession ***id***, ViInt16 ***magLevel***);

ViStatus **hpe1437_trigger_maglevel_get**(ViSession ***id***, ViPInt16 ***magLevelPtr***);

ViStatus **hpe1437_trigger_slope**(ViSession ***id***, ViInt16 ***slope***);

ViStatus **hpe1437_trigger_slope_get**(ViSession ***id***, ViPInt16 ***slopePtr***);

ViStatus **hpe1437_trigger_type**(ViSession ***id***, ViInt16 ***tType***);

ViStatus **hpe1437_trigger_type_get**(ViSession ***id***, ViPInt16 ***tTypePtr***);

ViStatus **hpe1437_wait**(ViSession ***id***);

Visual Basic Quick Reference

Return& = **hpe1437_attrib_get**(***id***&, ***attrib***%, ***value***&)

Return& = **hpe1437_clock_setup**(***id***&, ***sync***%, ***source***%, ***dsp***%, ***master***%,***fs***#)

Return& = **hpe1437_clock_dsp**(***id***&, ***dsp***%)

Return& = **hpe1437_clock_dsp_get**(***id***&, ***dspPtr***%)

Return& = **hpe1437_clock_fs**(***id***&,***fs***#)

Return& = **hpe1437_clock_fs_get(*id*&,*fsPtr*#)**

Return& = **hpe1437_clock_master**(***id***&, ***master***%)

Return& = **hpe1437_clock_master_get**(***id***&, ***masterPtr***%)

Return& = **hpe1437_clock_multi_sync**(***id***&, ***sync***%)

Return& = **hpe1437_clock_multi_sync_get**(***id***&, ***syncPtr***%)

Return& = **hpe1437_clock_source**(***id***&, ***source***%)

Return& = **hpe1437_clock_source_get**(***id***&, ***sourcePtr***%)

Return& = **hpe1437_close**(***id***&)

Return& = **hpe1437_data_memsize_get**(***id***&, ***memSizePtr***%)

Return& = **hpe1437_data_scale_get**(***id***&, ***scalePtr***#)

Return& = **hpe1437_data_setup**(***id***&, ***dType***%, ***resolution***%, ***mode***%, ***blocksize***&, ***appendStatus***%, ***port***%)

Return& = **hpe1437_data_append_status**(***id***&, ***appendStatus***%)

Return& = **hpe1437_data_append_status_get**(***id***&, ***appendStatusPtr***%)

Return& = **hpe1437_data_blocksize**(***id***&, ***blocksize***&)

Return& = **hpe1437_data_blocksize_get**(***id***&, ***blocksizePtr***&)

Return& = **hpe1437_data_mode**(***id***&, ***mode***%)

Return& = **hpe1437_data_mode_get**(***id***&, ***modePtr***%)

Return& = **hpe1437_data_port**(***id***&, ***port***%)

Return& = **hpe1437_data_port_get**(***id***&, ***portPtr***%)

Return& = **hpe1437_data_resolution**(***id***&, ***resolution***%)

Return& = **hpe1437_data_resolution_get**(***id***&, ***resolutionPtr***%)

Return& = **hpe1437_data_type**(***id***&, ***dType***%)

Return& = **hpe1437_data_type_get**(***id***&, ***dTypePtr***%)

Return& = **hpe1437_error_message**(***id***&, ***errNum***&, ***errMessage***$)

Return& = **hpe1437_error_query**(***id***&, ***errNumPtr***&, ***errMessage***$)

Return& = **hpe1437_filter_resp_get**(***id***&, ***resp***#[ ], ***n***&,***fmin***#,***fmax***#)

Return& = **hpe1437_filter_setup**(***id***&, ***sigBw***%, ***decimate***%)

Return& = **hpe1437_filter_decimate** (***id***&, ***decimate***%)

Return& = **hpe1437_filter_decimate_get**(***id***&, ***decimatePtr***%)

Return& = **hpe1437_filter_bw** (***id***&, ***sigBw***%)

Return& = **hpe1437_filter_bw_get**(***id***&, ***sigBwPtr***%)

Return& = **hpe1437_filter_sync**(***id***&)

Return& = **hpe1437_frequency_center_raw**(***id***&, ***coarse***%,***fine***&)

Return& = **hpe1437_frequency_setup**(***id***&, ***cmplxDc***%, ***sync***%,***freq***#)

Return& = **hpe1437_frequency_cmplxdc**(***id***&, ***cmplxDc***%)

Return& = **hpe1437_frequency_cmplxdc_get**(***id***&, ***cmplxDcPtr***%)

Return& = **hpe1437_frequency_sync**(***id***&, ***sync***%)

Return& = **hpe1437_frequency_sync_get**(***id***&, ***syncPtr***%)

Return& = **hpe1437_frequency_center**(***id***&,***freq***#)

Return& = **hpe1437_frequency_center_get**(***id***&,***freqPtr***#)

Return& = **hpe1437_init**(***instrDesc***$, ***idQuery***%, ***rst***%, ViPSession ***id***)

Return& = **hpe1437_input_autozero**(***id***&)

Return& = **hpe1437_input_setup**(***id***&, ***range***%, ***coupling***%, ***antiAlias***%, ***signal***%, ***floatIn***%)

Return& = **hpe1437_input_alias_filter**(***id***&, ***antiAlias***%)

Return& = **hpe1437_input_alias_filter_get**(***id***&, ***antiAliasPtr***%)

Return& = **hpe1437_input_coupling**(***id***&, ***coupling***%)

Return& = **hpe1437_input_coupling_get**(***id***&, ***couplingPtr***%)

Return& = **hpe1437_input_float**(***id***&,***floatIn***%)

Return& = **hpe1437_input_float_get**(***id***&,***floatInPtr***%)

Return& = **hpe1437_input_range**(***id***&, ***range***%)

Return& = **hpe1437_input_range_get**(***id***&, ***rangePtr***%)

Return& = **hpe1437_input_signal**(***id***&, ***signal***%)

Return& = **hpe1437_input_signal_get**(***id***&, ***signalPtr***%)

Return& = **hpe1437_input_range_auto**(***id***&, ***sec***#)

Return& = **hpe1437_interrupt_restore**(***id***&)

Return& = **hpe1437_interrupt_setup**(***id***&, ***intrNum***%, ***priority***%, ***mask***%)

Return& = **hpe1437_interrupt_mask_get**(***id***&, ***intrNum***%, ***maskPtr***%)

Return& = **hpe1437_interrupt_priority_get**(***id***&, ***intrNum***%, ***priorityPtr***%)

Return& = **hpe1437_lbus_mode**(***id***&, ***lbusMode***%)

Return& = **hpe1437_lbus_mode_get**(***id***&, ***lbusModePtr***%)

Return& = **hpe1437_lbus_reset**(***id***&, ***lbusReset***%)

Return& = **hpe1437_lbus_reset_get**(***id***&, ***lbusResetPtr***%)

Return& = **hpe1437_meas_control**(***id***&, ***idle***%, ***sync***%)

Return& = **hpe1437_meas_start**(***id***&)

Return& = **hpe1437_read**(***id***&, ***rec***&[ ], ***sampleCount***&, ***overloadPtr***%)

Return& = **hpe1437_read64**(***id***&, ***rec***#[ ], ***sampleCount***&, ***overloadPtr***%)

Return& = **hpe1437_read_raw**(***id***&, ***rec***%[ ], ***wordCount***&)

Return& = **hpe1437_reset**(***id***&)

Return& = **hpe1437_revision_query**(***id***&, ***driverRev***$, ***instRev***$)

Return& = **hpe1437_self_test**(***id***&, ***testResultPtr***%, ***testMessage***$)

Return& = **hpe1437_status_get**(***id***&, ***statusPtr***%)

Return& = **hpe1437_trigger_delay_actual_get**(***id***&, ***actualDelayPtr***#)

Return& = **hpe1437_trigger_phase_actual_get**(***id***&, ***actualPhasePtr***#)

Return& = **hpe1437_trigger_phase_capture**(***id***&)

Return& = **hpe1437_trigger_setup**(***id***&, ***tType***%, ***delay***&, ***adcLevel***%, ***magLevel***%, ***slope***%, ***gen***%)

Return& = **hpe1437_trigger_adclevel**(***id***&, ***adcLevel***%)

Return& = **hpe1437_trigger_adclevel_get**(***id***&, ***adcLevelPtr***%)

Return& = **hpe1437_trigger_delay**(***id***&, ***delay***&)

Return& = **hpe1437_trigger_delay_get**(***id***&, ***delayPtr***&)

Return& = **hpe1437_trigger_gen**(***id***&, ***gen***%)

Return& = **hpe1437_trigger_gen_get**(***id***&, ***genPtr***%)

Return& = **hpe1437_trigger_maglevel**(***id***&, ***magLevel***%)

Return& = **hpe1437_trigger_maglevel_get**(***id***&, ***magLevelPtr***%)

Return& = **hpe1437_trigger_slope**(***id***&, ***slope***%)

Return& = **hpe1437_trigger_slope_get**(***id***&, ***slopePtr***%)

Return& = **hpe1437_trigger_type**(***id***&, ***tType***%)

Return& = **hpe1437_trigger_type_get**(***id***&, ***tTypePtr***%)

Return& = **hpe1437_wait**(***id***&)

Parameter numeric equivalents

Numeric equivalents may be used in place of alphanumeric variables in function calls. These numeric equivalents are also available as popups within online function parameter descriptions.

HPE1437_16BIT	1
HPE1437_32BIT	0
HPE1437_20000KHZ	1
HPE1437_20480KHZ	0
HPE1437_AC	1
HPE1437_ADC	1
HPE1437_APPEND	2
HPE1437_ASSERT	1
HPE1437_BLOCK	0
HPE1437_BUFFER	2
HPE1437_COMPLEX	1
HPE1437_CONTINUOUS	1
HPE1437_DATA_REGISTER	3
HPE1437_DC	0
HPE1437_EXT_PLL_REF	3
HPE1437_EXTEND	3
HPE1437_EXTERNAL	2
HPE1437_FRONT	1
HPE1437_GENERATE	1
HPE1437_IMMEDIATE	4
HPE1437_INSERT	3
HPE1437_INTEL	1
HPE1437_IO_ADDRESS	1
HPE1437_IO_HANDLE	0
HPE1437_LBUS	1
HPE1437_MAG	3
HPE1437_MOTOROLA	0
HPE1437_NEGATIVE	1

HPE1437_OFF	0
HPE1437_ON	1
HPE1437_OSCILLATOR	0
HPE1437_PIPELINE	0
HPE1437_POSITIVE	0
HPE1437_REAL	0
HPE1437_REAR	2
HPE1437_RELEASE	0
HPE1437_RM_HANDLE	2
HPE1437_USER	0
HPE1437_VME	0

Errors

The following errors are generated by library calls:

0000 HPE1437_SUCCESS "No error."

0001 HPE1437_NO_DATA_MEASUREMENT_IN_PROGRESS "No data available, a measurement is in progress."

0002 HPE1437_NO_DATA_MEASUREMENT_PAUSED "No data available, the measurement is paused."

0003 HPE1437_NO_DATA_WAITING_FOR_TRIGGER "No data available, trigger has not occurred."

0004 HPE1437_NO_DATA_WAITING_FOR_ARM "No data available, acquiring pre-trigger data."

0005 HPE1437_BAD_RESOURCE_DESCRIPTOR "The resource descriptor string is not valid."

0006 HPE1437_NO_E1437_FOUND "No E1437 found at specified logical address."

0007 HPE1437_PROC_READY_TIMEOUT "Timeout is waiting for E1437 command processor."

0008 HPE1437_MEMORY_ALLOCATION_ERROR "Memory allocation error."

0009 HPE1437_CAPABILITY_NOT_SUPPORTED "Capability not supported."

0010 HPE1437_BAD_ERR_NO "The returned error number does not exist."

0011 HPE1437_UNSUPPORTED_HARDWARE_CONFIG "Unsupported hardware configuration."

0012 HPE1437_CAN'T_START "Unable to start measurement."

0013 HPE1437_NULL_ID "Hardware addressed does not exist."

0014 HPE1437_RESOURCE_MANAGER_ERROR "Resource Manager could not be executed successfully; possible installation error."

The following errors are generated by firmware:

0097 HPE1437_BAD_COMMAND "Invalid command code."

0098 HPE1437_PARM_ERROR "Invalid command parameter."

0100 HPE1437_CAL_SAVE_ERROR "Error in saving calibration constants."

0101 HPE1437_DOWNLOAD_ERROR "Error while downloading new firmware."

0102 HPE1437_DSPCLOCK_TOO_SLOW_ERROR "DSP clock slower than minimum specification."

0103 HPE1437_AUTOZERO_ERROR "Autozero error, hardware problem."

0104 HPE1437_MODE_ERROR "Invalid mode requested."

0105 HPE1437_START_ERROR "Unable to start measurement."

0106 HPE1437_SELFTEST_ERROR "Error occurred during self test."

0107 HPE1437_INTERNAL_ERROR "Internal software error occurred."

0108 HPE1437_AUTORANGE_ERROR "Error occurred during autoranging, hardware problem."

0127 HPE1437_BYTE_SWAP_ERROR "Invalid command code, possible byte order error."

Functions Which Abort Measurements

The following functions abort any measurement in progress:

hpe1437_clock_dsp
hpe1437_clock_master.
hpe1437_clock_multi_sync
hpe1437_clock_source
hpe1437_data_append_status
hpe1437_data_blocksize
hpe1437_data_mode
hpe1437_data_port
hpe1437_data_resolution
hpe1437_data_type
hpe1437_filter_decimate
hpe1437_filter_bw
hpe1437_filter_sync
hpe1437_init
hpe1437_input_autozero
hpe1437_lbus_mode
hpe1437_meas_control (depending on write value)
hpe1437_meas_start
hpe1437_reset

5

ASCII Overview and Commands

Introduction

ASCII commands allow you to communicate with the E1437A without using the libraries, although most users will find it easier and faster to use libraries than these ASCII commands. The ASCII commands in this chapter are provided mainly to accommodate users who have previously used SCPI (Standard Commands for Programming Instruments) with the HP/Agilent E1406 Command Module. You will note the similarities in command structure between these ASCII commands and SCPI.

Command Syntax

This section describes the syntax elements used in the ASCII command reference.

Special Syntactic Elements

Some syntactic elements have special meanings:

- colon (:) — The colon is a part of the program header (command or query) and does not imply a heirarchy such as that which exists with SCPI commands for other instruments.
- comma (,) — A comma separates the data sent with a command or returned with a response. For example the FILTER:SETUP command requires two values: one to select the filter signal bandwidth and one to select extra decimation. A message to select 460 kHz bandwidth and a decreased sample rate of 1.28 MHz would be:

```
FILTER:SETUP 4,1
```

- <WSP> — One white space is required to separate a program headers (the command or query) from its parameters. For example the command "FILTER:SETUP 4,1" contains a white space between the program header (FILTER:SETUP) and the parameters (4,1). White space characters are not allowed within the program header.

Conventions

Syntax and return format description use the following conventions:

- < > Angle brackets enclose the names of items that need further definition. The definition will be included in accompanying text.
- ::= "is defined as" When two items are separated by this synbol, the second item replaces the first in any statement that contains the first item. For example, A::=B indicates that B replaces A in any statment that contains A.
- | "or" When items in a list are separated by this symbol one and only one of the items can be chosen from the list For example, A|B indicates that A or B can be chosen, but not both.
- … an ellipsis (trailing dots) is used to indicate that the preceding element may be repeated one or more time.

The command interpreter is not case sensitive. No short forms for keywords are allowed

Using ASCII Commands in Your Environment

ASCII commands require no drivers or other special downloadable files. They may be sent from the host computer through an GPIB/HPIB interface to a HP/Agilent E1406 Command Module in a VXI mainframe containing the E1437A.

Using ASCII commands with HP BASIC

In order to address the module you must know the addressing information about your GPIB/HPIB interface, your command module, and the E1437A. The addressing format is as follows:

```
HCCMM

where H=the HP-IB interface select code
           CC=the command module's HP-IB address
           MM=the E1437A module's logical address divided by 8.
```

For example if your HPIB/GPIB interface is at select code 7, the HP/Agilent E1406 command module is at HPIB/GPIB address 9, and the E1437A's logical address is 192, the address you use for ASCII commands is 70924.

Example statements in the ASCII Command Reference represent this environment.

Using ASCII commands with VISA

It is possible to send ASCII commands through the VISA interface, although using the C function library provides more capability and greater ease of use.

Before using ASCII in this environment be sure that all standard VISA files are installed and that the interface is properly configured.

The following is an example of sending ASCII commands to the E1437A through the VISA interface:

```
Declare Function viReadbin Lib "VISA32.DLL" Alias "#256" (ByVal vi
As Long, Buffer As Any, ByVal count As Long, retCount As Long)
As Long
Dim rec(1024) As Long

er = viOpenDefaultRM(rm)
er = viOpen(rm, "VXI::192", 0, 0, id) output id, "MEAS:START"
output id, "READ 32"
er = viReadbin(id, rec(0), 4096, retCount&)
REM <The data in rec() is available for use here.> er = viClose(id)
er = viClose(rm)

Sub output(id, a$)
er = viWrite(id, a$, Len(a$), retCount&)
End Sub
```

ASCII Programming Reference

*IDN? query

Returns a string that identifies the E1437A.

Query syntax: *IDN?

Example Statement:
```
OUTPUT 70924;"*Idn?"
ENTER 70924;identity$
```

Return Format: HEWLETT-PACKARD, E1437A, <serial number>, <swrev0:swrev1:hwrev3>

Description: The response to this query uniquely identifies your module and the version of the module's firmware and hardware.

*RST command

Executes a device reset..

Command syntax: *RST

Example Statement: OUTPUT 70924;"*rst"

Description: This command returns the module to a reset state.

The following are not affected by this command:

- Calibration constants

*TST? query

Tests the module's hardware and returns the result..

Query syntax: *TST?

Example Statement: OUTPUT 70924;"*TST?"

Description: The module's selt-test performs the E1437A diagnostic tests. If the results are within specified limits, the module returns 0. If the results exceed the secified limits, the module returns 1 and an error message is placed in the error queue. The length of the self-test is approximately as follows:

Memory SIze (MBytes)	Time (min)
8	1
16	1.5
32	2.5
48	4.5

The query accesses the error queue.

The following tests are performed:

- Digital: rails the front end to a full scale value then turns on zooming, filtering, and the final decimation to quickly verify those operations.
- Noise: does a quick baseband measurement with the input signal disconnected, and verifies that the front-end noise is within specification.
- Bump: Verifies some front-end levels associated with the analog-to-digital converter.
- Memory: fills the entire DRAM then verifies that all the data is correct.
- Analog: verifies that autozero adjust is working and that the input is triggering.

CLOCK:SETUP command/query

Sets all timing parameters. This description also includes information on the following commands which set or query the timing parameters individually:

CLOCK:DSP selects the clock used to drive the decimation/zoom section.

CLOCK:FS provides the frequency of an external sample clock.

CLOCK:MASTER determines whether a module shares its ADC clock.

CLOCK:MULTI:SYNC specifies whether the module uses a shared clock and sync.

CLOCK:SOURCE selects the source of the ADC clock.

Command syntax:

CLOCK:SETUP <multisync>,<source>,<dsp>,<master>,<fs>

multisync::= 0 | 1 | 2

source::= 0 | 1 | 2 | 3

dsp::= 0 | 1

master::= 0 | 1 | 2

fs <numeric>

numeric::=>0-20600000

CLOCK:MULTI:SYNC 0 | 1 | 2

CLOCK:SOURCE 0 | 1 | 2 | 3

CLOCK:DSP 0 | 1

CLOCK:MASTER 0 | 1 | 2

CLOCK:FS <numeric>

numeric::=100000-20600000

Query syntax:

CLOCK:DSP?

CLOCK:FS?

CLOCK:MASTER?

CLOCK:MULTI:SYNC?

CLOCK:SOURCE?

Example Statement:

OUTPUT 70924;"Clock:setup 1,2,0,2,10000000"

OUTPUT 70924;"Clock:Multi:Sync 2"

Description:

CLOCK:SETUP is used to configure all timing parameters used for sampling (ADC clock) and decimation/zoom (DSP clock). This command, as well as the other CLOCK commands covered in this description, is used to select the source and distribution of clocking and synchronization signals used by the E1437 module. The primary clock signal used by the module is the ADC clock, for which the rising edges indicate the time for each sample of the analog-to-digital converter. Another clock signal is the DSP clock, which drives the digital signal processing and memory sections of the module. Normally the DSP clock is the same as the ADC clock, and data is transferred synchronously from the ADC to the DSP portion of the module. However, in certain situations the two clocks may be independent, requiring asynchronous data transfers from the ADC to the DSP. The remaining CLOCK commands and queries listed above set or query the parameters individually.

Parameter definitions:

is used to specify whether the module uses a shared ADC clock and SYNC signal. Modules in multi-module systems must all have the same *sync* parameter setting.

parameter value	***multisync*** **parameter definition**
0	OFF. The ADC clock and SYNC are generated locally
1	FRONT. The module uses the shared clock and SYNC provided on the front panel distribution connectors
2	REAR. The module uses the shared ADC clock and SYNC signals which are distributed on the VXI backplane using the ECL trigger lines

selects the clock source that is used to drive the analog to digital converter (ADC) for single module operation or when a module is used as the master ADC clock source for a multi-module system. In multi-module systems the source parameter is ignored for all but the master module.

parameter value	***source*** **parameter definition**
0	20.48 MHz internal oscillator
1	20 MHz internal oscillator
2	EXT. TTL, ECL, or sine signal on the external, BNC, front panel clock input connector
3	EXT:PLL. Takes a 10 MHz reference from another instrument on the external, BNC, front panel clock input connector and uses a PLL to convert it to a 20 MHz reference

selects the clock used to drive the decimation/zoom section within the E1437. Normally, the DSP clock should be coupled to the ADC clock whenever possible since the spurious performance specification is degraded when the clocks are independent. However, when a slow or intermittent ADC clock results in greater than 1 μs between clock edges, the DSP clock must be generated from the internal oscillator to avoid data loss in the dynamic RAM.

parameter value	***dsp*** **parameter definition**
0	OSC. Causes the DSP clock to be the internally generated 20.48 MHz oscillator.
1	ADC. Forces the DSP clock to be driven by the ADC clock

determines whether an E1437 makes its local ADC clock available to other modules as a shared clock. Multi-module synchronization requires that one and only one of the modules to be identified as the master, the source of the shared ADC clock.

parameter value	***master*** **parameter definition**
0	OFF. The module is driving neither the front panel nor the back plane. This is the correct variable to use for all non-master modules in a system.
1	ON. When *multisync* = 1 (front panel) the E1437 drives the front panel ADC clock. If *multisync* = 2 (back plane) the module uses its ADC clock to drive the VXI backplane in the mainframe in which it resides.
2*	BUFFER. Allows the ADC clock and SYNC lines from the module's front panel connectors to drive the backplane of a mainframe not containing the master.

* Only one module per mainframe may be set to 1 or to 2. In multi-mainframe systems using backplane clock and sync distribution only one module per any mainframe not containing the master may be set to 2.

provides the module with the frequency of an external sample clock connected to the Ext Clk TTL connector. When using an external clock or when a module is a non-master in a multi-module group, the frequency of the ADC clock is unknown by the module. It is the responsibility of the programmer to provide the correct frequency so that commands dependent on fs will operate properly. This value has no effect if the module is set up to use the internal ADC clock.

Comments:

For more details on the interaction among source, master and sync with multiple modules and multiple mainframes see Managing multiple modules.

The master, multisync, source, and dsp parameters are interdependent with legitimate combinations being as follows (along with the resultant DSP clock rates):

MASTER	SYNC	SOURCE	DSP	DSP CLOCK RATE
N/A	OFF	20.x (internal)	N/A	Internal source
N/A	OFF	EXT	ADC	External source
N/A	OFF	EXT	OSC	20.48
N/A	OFF	EXT:PLL	N/A	20
OFF | BUFFER	FRONT	N/A	ADC	Master ADC
OFF | BUFFER	FRONT	N/A	OSC	20.48
OFF	REAR	N/A	ADC	Master ADC
OFF	REAR	N/A	OSC	20.48
ON	FRONT	20.x	N/A	Source
ON	FRONT	EXT	ADC	External
ON	FRONT	EXT	OSC	20.48
ON	FRONT	EXT:PLL	N/A	20
ON	REAR	20.x	N/A	Source
ON	REAR	EXT	ADC	External
ON	REAR	EXT	OSC	20.48
ON	REAR	EXT:PLL	N/A	20
BUFFER	REAR	N/A	ADC	Master ADC
BUFFER	REAR	N/A	OSC	20.48

If *fs*>20,480,000 then dsp must = ADC

The maximum rate at which data may be transferred to memory is determined by the DSP clock rate: Max bytes/s. = 4 * DSP clock rate. In continuous mode the maximum rate is limited to (4 * DSP clock rate)/2. However, you may successfully perform this type of measurement by adding a level of decimation to reduce the sample rate.

Example:

The correct method to set up a synchronous multi-module group that insures that all modules share the same ADC clock is:

```
! First, insure that one module is putting its clock on the backplane
OUTPUT <addrMaster>;" CLOCK:Master  1"
! Put each module into multi-sync mode with internal clock!  (unless external
clock is connected to
! master HP E1437  through Ext Clk TTL connector).
! For each module address (except master):
OUTPUT <addrAll>;"Clock:Setup 2,0,1,0,20480000"
```

Reset State:

multisync=OFF, *source*=20480000, *dsp*=ADC, *master*=OFF,*fs*=20480000

See Also:

FILTER:SETUP, DATA:SETUP

DATA:SETUP command/query

Sets all format and data output flow parameters. This description also includes information on the following commands which set or query the format and flow parameters individually:

DATA:APPEND:STATUS appends status information to a data block.

DATA:APPEND:STATUS? gets the append status state

DATA:BLOCKSIZE determines the size of the output data block.

DATA:BLOCKSIZE? gets the output data block size

DATA:MODE selects block mode or continuous mode.

DATA:MODE? gets the data mode

DATA:PORT selects VME bus or local bus output port.

DATA:PORT? gets the output port designation

DATA:RESOLUTION selects 16 or 32 bits data resolution.

DATA:RESOLUTION? gets the data resolution

DATA:TYPE selects real or complex output data.

DATA:TYPE? gets output data type

Command syntax:

DATA:SETUP <type>,<resolution>,<mode>,<blocksize>,<append>,<port>

type::=0 | 1

resolution::=0 | 1

mode::=0 | 1

blocksize <numeric>

numeric::= 1 to memorysize/2

append::=0 | 1

port::=0 | 1

DATA:APPEND:STATUS 0

DATA:BLOCKSIZE <numeric>

numeric::= 1 to memorysize/2

DATA:MODE 0 | 1

DATA:PORT 0 | 1

DATA:RESOLUTION 0 | 1

DATA:TYPE 0 | 1

Query syntax:

DATA:APPEND:STATUS?

DATA:BLOCKSIZE?

DATA:MODE?

DATA:PORT?

DATA:RESOLUTION?

DATA:TYPE?

Example Statement:

OUTPUT 70924;"DATA:setup 1,1000000,0,2,0,1"

OUTPUT 70924;"Data:mode 2"

Parameter definitions:

determines whether the E1437 collects and returns real or complex data. Normally, if the frequency set with the FREQUENCY:SETUP command is zero, the type should be set to real since the imaginary component of each sample is zero anyway. When non-zero center frequencies are used the type should normally be set to complex. Otherwise the imaginary component of the signal will be lost.

parameter value	***type*** parameter definition
0	REAL. Causes only the real part of the data to be returned for each sample.
1	COMPLEX. Causes the real data followed by the imaginary data to be returned in each sample.

selects the data resolution. Choosing 16-bit precision allows for more samples in the FIFO memory. Choosing 32 bits allows more dynamic range. Because of the broadband white noise present on the input of the analog-to-digital converter, it is normally sufficient to use 16 bit resolution whenever the FILTER:SETUP command specifies a signal bandwidth greater than 250 kHz. For narrower bandwidths much of the broadband white noise is filtered out, resulting in lower noise in the output data. To take advantage of this lower noise, the 32-bit data resolution should be used.

parameter value	***resoultion*** parameter definition
0	32 BIT. Selects data resolution of 32 bits.
1	16 BIT. Selects data resolution of 16 bits.

selects whether the E1437's data collection operates in block mode or continuous mode. Block mode is used whenever each block of data is to be associated with an individual trigger "event". The continuous mode is useful for continuous signal processing applications where data gaps are unacceptable.As long as the data is read out fast enough to prevent overflow in the output FIFO, the measurement will continue.

parameter value	***mode*** parameter definition
0	BLOCK. Selects block transfer mode in which the measurement is halted after each block of data. To start collection of the next data block the module must be armed and triggered again
1	CONTINUOUS. Means that a single arm and trigger event starts a measurement which runs continuously with no gaps between output data blocks.

determines the number of sample points in each output data block. The range of available block sizes depends on the number of bytes required for each sample. The command accepts any number between 1 and memory size (in bytes)/2. The actual number used is the first integer power of 2 equal to or larger than the requested blocksize. If the requested block size falls outside the range shown in the table the closest valid value will be used and a status register flag (bit 6) will be set indicating a setup error. If a subsequent change in another parameter permits a block size closer to the originally requested value, the module will adjust the block size to that value.

The following table summarizes the available block sizes for each setting of the dType and resolution parameters.

Data port	Data type	Resolution	Bytes per sample	Block size (with standard 8 Bytes memory) *	
				Min	Max
VME	REAL	16	2	3	4,194,304
VME	REAL	32	4	2	2,097,152
VME	COMPLEX	16	4	2	2,097,152
VME	COMPLEX	32	8	1	1,048,576
LBUS	REAL	16	2	6	4,194,304
LBUS	REAL	32	4	3	2,097,152
LBUS	COMPLEX	16	4	3	2,097,152
LBUS	COMPLEX	32	8	2	1,048,576

*For optional additional memory, multiply by the appropriate memory size multiplier. For example, for 32 MByte memory option multiply max block size by 4.

Note

Block size does not need to be a power of two. Considerably more samples may need to be taken in order to set the block available status bit.

selects whether or not status information is appended to a data block. In this status byte, Bit 0 will be set if an ADC overload occurred and bit 1 will be set for an ADC error. The other bits are undefined. When the appended byte is transferred via the VME backplane, the byte is located in the lower 8 bits of the 16 bit word after the end of the sampled data block. The upper 8 bits are undefined. When the appended byte is output via the local bus (as a 32-bit word), it is marked as the last byte of a transfer block. This status byte should be read separately from any block read operations in order to not affect the alignment of subsequent elements.

parameter value	*append* parameter definition
0	OFF. Disables the status append feature.
1	ON. Means that an extra byte of status information is appended to the end of each data block to indicate whether an ADC overload or error occurred during the collection of that block of data.

determines which output port is used to take data from the E1437 module.

parameter value	*port* parameter definition
0	VME. Means the data is to be output using standard VME register reads
1	LBUS. Means the data is to be output as a byte-serial data stream via the VXI local bus. When using the local bus port the module immediately to the right of the E1437 must be capable of receiving the local bus byte sequence.

The following table summarizes the output word or byte sequence for each combination of type, resolution, and port parameters:

Type	Resolution	Port	Sequence
REAL	16BIT	VME	R0[15:0],R1[15:0],...
COMPLEX	16BIT	VME	R0[15:0],Q0[15:0],R1[15:0],Q1[15:0],...
REAL	32BIT	VME	R0[31:16],R0[15:0],R1[31:16],R1[15:0],...
COMPLEX	32BIT	VME	R0[31:16],R0[15:0],Q0[31:16],Q0[15:0],R1[31:16]...
REAL	16BIT	LBUS	R0[15:8],R0[7:0],R1[15:8],R1[7:0],...
COMPLEX	16BIT	LBUS	R0[15:8],R0[7:0],Q0[15:8],Q0[7:0],R1[15:8]...
REAL	32BIT	LBUS	R0[31:24],R0[23:16],R0[15:8],R0[7:0],R1[31:24],...
COMPLEX	32BIT	LBUS	R0[31:24],R0[23:16],R0[15:8],R0[7:0],Q0[31:24],Q0[23:16],Q0[15:8],Q0[7:0], R1[31:24],...

Comments

The maximum rate at which data may be transferred to memory is determined by the DSP clock rate: Max bytes/s. = 4 * DSP clock rate. In continuous mode the maximum rate is limited to (4 * DSP clock rate) / 2. However, you may successfully perform this type of measurement by adding a level of decimation to reduce the sample rate.

A limitation also applies to 32-bit, complex data transfers. Because this type of transfer cannot be made at the full sample rate, a level of decimation must be added in order to reduce the sample rate

The following table summarizes under what data parameter combinations decimation must be used:

Resolution	Type	Decimation	Filter BW	Block	Continuous	Sample Rate (MBytes/sec)
16	Complex	False	0 or 1	Yes	No	80
32	Real	False	0 or 1	Yes	No	40
32	Complex	True	0 or 1	Yes	No	40
32	Complex	False	2	Yes	No	40
32	Complex	False	0 or 1	No	No	40
All other combinations				Yes	Yes	< 40

Reset Values

type=REAL, *resolution*=32BIT, *mod*e=BLOCK, *blocksize=1024, append*=OFF, *port*=VME

See Also

FREQUENCY:SETUP, FILTER:DECIMATE, MEAS:CONTROL, CLOCK:DSP

DATA:VME:ORDER command/query

Selects the 16-bit word ordering out of the VME port when the data resolution is 32 bits.

Command syntax; DATA:VME:ORDER <order>

order::=0 | 1

Query syntax: DATA:VME:ORDER?

Example Statement: OUTPUT 70924;"DATA:VME:Order 1"

Parameters:

parameter value	***order*** **parameter definition**
0	MOTOROLA. High word is output first
1	INTEL. Low word is output first

Reset Values *WordOrder*=MOTOROLA

ERROR query

Returns the error number for the oldest error in the queue.

Query syntax: ERROR?

Example Statement: OUTPUT 70924;"ERROR?"

FILTER:SETUP command/query

Sets the digital filter bandwidth and decimation filter parameters. This description also includes information on the following commands which set or query the decimation filter parameters individually:

FILTER:DECIMATE selects an extra factor of 2 decimation.

FILTER:DECIMATE? gets current state of extra decimation

FILTER:BW selects a signal filter bandwidth.

FILTER:BW? gets the signal filter bandwidth

Command Syntax:

FILTER:SETUP <sigBw>,<decimate>

sigBbw::=<numeric>

numeric::=0 to 24

decimate::=0|1

FILTER:BW <numeric>

<numeric>::=0 to 24

FILTER:DECIMATE::=0|1

Query Syntax:

FILTER:BW?

FILTER:DECIMATE?

Example Statements:

```
OUTPUT 70924;"FILTER:SETUP 12,0"
OUTPUT 70924;"FILTER:BW?"
ENTER 70924;Response$
```

Parameter Definitions:

selects an alias protected signal filter bandwidth that is roughly ±fs/(2.56 * 2^(sigBw)) where fs is the ADC sample frequency. In zoom applications, where the center frequency is generally not zero, the zoom filter bandwidth is centered on the frequency programmed with the frequency:setup command. For baseband measurements the filter may equivalently be considered as a low pass filter of approximately bandwidth fs/(2.56 * 2^(sigBw)) since the negative frequencies are generally of no interest.The valid range of sigBw is 0 through 24. When sigBw = 0, no digital filtering is applied to the signal and the module relies on the analog anti-alias filter to limit the signal bandwidth to fs/2.56.

To more accurately calculate the bandwidth use the calculation ±fs * k/2^(sigBw) where:

```
k=.36 for .25 dB bandwidth
k=.44 for 3 dB bandwidth
k=.5 for 15 dB bandwidth
k=.62 for 110 dB bandwidth
```

selects the data output sample rate. You would normally want to add the extra level of decimation in order to increase the displayed span.

parameter value	*decimate* parameter definition
0	OFF. The output sample rate is: *fs* when *bw* = 0 or fs/ 2^(*bw*-1) when *bw* > 0.
1	ON. The output sample rate is reduced by an additional factor of two by discarding alternate samples

Comments

To ensure full alias-free operation the analog anti-alias filter (set by the INPUT:ALIAS:FILTER command) should be ON unless the application inherently bandlimits the input signal to less than fs/2. The analog anti-alias filter has a fixed bandwidth and thus is fully effective only when fs>=20 MHz. If a slower external ADC clock is used, an additional analog filter of the appropriate bandwidth may be required for full alias protection.

The decimation process used to reduce the output sample rate is driven from a "decimation counter" which keeps track of which samples to save and which ones to discard for each of the octave bandwidth reduction filter stages. In multi-module systems where synchronous sampling is required, the decimation counters in all the modules must be synchronous with each other. This condition can be forced by using the FILTER:SYNC command.

The following table summarizes the relationship between data parameter combinations, decimation, filter bandwidth, and whether the particular combination permits block or continuous measurements:

Resolution	Type	Decimation	Filter BW	Block	Continuous	Sample Rate (MBytes/sec)
16	Complex	False	0 or 1	Yes	No	80
32	Real	False	0 or 1	Yes	No	40
32	Complex	True	0 or 1	Yes	No	40
32	Complex	False	2	Yes	No	40
32	Complex	False	0 or 1	No	No	40
All other combinations				Yes	Yes	< 40

Example: Here are some bandwidth and sample rate results using the "k" calculation for bandwidth:

Fs = 20.48 MHz default internal ADC clock

	Signal Bandwidth		Sample Rate	
sigBw	**25 Db**	**15 Db**	**Decimation OFF**	**Decimation ON**
0	±7.37	±10.24	20.48	10.24 (see CAUTION)
1	±3.69	±5.12	20.48	10.24
2	±1.84	±2.56	10.24	5.12
3	±0.92	±1.28	5.12	2.56
4	±0.46	±0.64	2.56	1.28
... Continue to decrease by factors of two ...				

CAUTION Turning decimation ON when *bw*=0 results in aliasing (garbage data) due to upper limit of the sampling frequency.

Reset Values *sigBbw*=0, *decimate*=OFF

See Also CLOCK:FS?, FREQUENCY:SETUP, FILTER:SYNC, INPUT:ALIAS:FILTER, DATA:MODE

FILTER:SYNC command

Synchronizes the decimation counter.

Command Syntax: FILTER:SYNC

Description: This command causes the digital decimation counter to be reset by the next SYNC line rising transition. Any measurement in progress is terminated and the module is placed in the idle state. By calling FILTER:SYNC for every E1437 module using a shared ADC clock, and then calling MEAS:CONTROL to cause a SYNC transition, the decimation counters will be started at the same time. Once this is done the decimation counters will stay synchronized as long as the same ADC clock is used. It is not necessary to resynchronize the decimation counters when the digital filter bandwidths are changed.

Comments: If you also want to synchronize frequency or phase, see FREQUENCY:SETUP and multi-module information .

Example: The following example shows how to use this command while avoiding potential conflicts and undefined conditions.

```
! Force all modules to Idle state
OUTPUT <addrAll>; "MEAS:CONTROL 1,0"
! Hold in IDLE to avoid undesired SYNC release */
! Release forced idle on all modules
OUTPUT <addrAll>;"MEAS:CONTROL 0,0"
!
! Wait for last module Sync/Idle Complete bit 7
REPEAT
  OUTPUT <addrAll>;"STATUS?"
  ENTER <addrAll>;Oper_status
UNTIL BIT (Oper_status,7)

! Put all modules into filter Sync mode
 OUTPUT <addrAll>;"FILTER:SYNC"
!
!Assert & release sync to syncronize all modules
OUTPUT <addrMaster>;"MEAS:CONTROL 0,1"
OUTPUT <addrMaster>;"MEAS:CONTROL 0,0"

!Verify Sync Valid on Master
REPEAT
  OUTPUT <addrMaster>;"STATUS?"
  ENTER <addrMaster>;Oper_status
UNTIL BIT (Oper_status,7)
!
! Toggle SYNC line to arm all modules
OUTPUT <addrMaster>;"MEAS:CONTROL 0,1"
OUTPUT <addrMaster>;"MEAS:CONTROL 0,0"
!
!Allow trigger
```

NOTE Resetting the decimation counter causes a transient in the digital filters. The transient takes about 30 output sample periods to decay 120 dB. See the impulse response graphs in the specification section for more detail.

See Also: FILTER:SETUP, MEAS:CONTROL, FREQUENCY:CMPLXDC

FREQUENCY:CENTER:RAW command/query

Provides a fast way to set the center frequency.

Command Syntax: FREQUENCY:CENTER:RAW <coarse>, <fine>

coarse::=0 to 2047

fine::=0 to 499999999

Query Syntax: FREQUENCY:CENTER:RAW?

Example Statements: OUTPUT 70924;"FREQUENCY:CENTER:RAW 1024,1000000

Description: This command sets the center frequency without relying on the internal E1437 microprocessor to do any floating point computations, since the internal microprocessor does not have a floating point co-processor. The resulting center frequency is:

fs*((coarse/2048)+(fine/1.024*10^12))

Parameter Definitions:

sets high frequencies or a low resolution frequency component.

sets very low frequencies or a high resolution frequency component.

See Also: FREQUENCY:SETUP, CLOCK:FS:GET, DATA:TYPE, MEAS:CONTROL

FREQUENCY:SETUP command/query

Sets all the zoom center frequency parameters. This description also includes information on the following commands which set or get frequency parameters individually:

FREQUENCY:CMPLXDC selects a complex baseband measurement

FREQUENCY:CMPLXDC? gets the state of the baseband measurement mode

FREQUENCY:SYNC prepares the module for a synchronous frequency change

FREQUENCY:SYNC? gets the state of the synchronous change mode

FREQUENCY:CENTER sets the center frequency

FREQUENCY:CENTER? gets the current center frequency

Command Syntax:

FREQUENCY:SETUP <cmplxdc>,<sync>,<frequency>

cmplxdc::=0 | 1

sync::=0 | 1

frequency <numeric>

numeric::= –0.5 - +0.5

FREQUENCY:CMPLXDC 0 | 1

FREQUENCY:SYNC 0 | 1

FREQUENCY:CENTER <numeric>

<numeric>::= –0.5 - +0.5

Query Syntax:

FREQUENCY:CMPLXDC?

FREQUENCY:SYNC?

FREQUENCY:CENTER?

Example statements:

```
OUTPUT 70924;"FREQUENCY:SETUP 1,0, 0.25"
OUTPUT 70924;"FREQUENCY:CENTER?"
ENTER 70924;Response$
```

Description:

FREQUENCY:SETUP sets the center frequency of a zoomed measurement. The center of a frequency band of interest is converted to DC with this command. The frequency transition is phase continuous unless the center frequency is set to zero in which case the transition may be selected either to be phase continuous or phase reset. This command may also be used to synchronously change frequency in multiple-module systems.

Parameter Definitions:

selects either a phase continuous or phase reset transition when the freq = 0. . The state of this parameter does not affect any transition where freq ≠0. Whether the real or complex data is saved and ultimately sent to the output port is determined by the DATA:TYPE command.

parameter value	***cmplxdc*** **parameter definition**
0	OFF causes phase to be reset to zero when combined with a frequency change to zero
1	ON combined with a frequency change to zero does not reset the phase, thereby generating a complex DC measurement at baseband.

controls when a frequency transition is implemented.

parameter value	***sync*** **parameter definition**
0	OFF allows an immediate frequency change.
1	ON. In multiple-module systems, setting this parameter ON prepares the modules for a frequency change, but does not actually bring about the change until the next ADC clock corresponding to the next assertion of the shared SYNC signal. The SYNC transition is generated by calling the MEAS:CONTROL command. Note that returning sync to OFF before the SYNC signal transition has occurred forces an immediate asynchronous frequency change.

is a number between -0.5 and +0.5, which will be interpreted as a fraction of the sample frequency. *freq* is the desired center frequency divided by the ADC sample frequency. For example, selecting .25 with a sample clock frequency of 20 MHz will yield a center frequency of 5.0 MHz. The ADC sample frequency is returned by the CLOCK:FS? command. Negative frequencies select the negative image of the signal, which is spectrally inverted from the input signal.

Comments:

Although the *freq* parameter is a double floating point number, its effective resolution is 1/(1024*10^9) or 20 μHz when *fs*=20.48 MHz. The actual frequency will be set to the nearest available value. This value is returned by the FREQUENCY:CENTER? command. In multi-module systems this value represents the pending value rather than the current value when a frequency change is incomplete due to a pending SYNC signal transition.

In multiple-module systems it is often desirable to force the frequency change to occur synchronously in order to preserve the phase relationship of the LOs. This is accomplished by setting the sync parameter to ON for all the modules which are to be changed. See the first example below.

In configurations involving synchronous operation of multiple E1437 modules, the FREQUENCY:SETUP command provides a mechanism to force all LOs to the same phase. This can be done by first setting the frequency to zero. See the second example below.

Example:

The following example shows how to synchronously change the center frequency and maintain the phase relationship between modules in a multi-module system without stopping a measurement in progress.

```
! For all ids, check status bits 0 and 1 to assure that all modules are in
MEASURE or IDLE
! state. Changing frequency on modules in TRIGGER or ARM states may risk
unintended
! frequency changes.
!
OUTPUT <addrAll>;"status?"
ENTER  <addrAll>;Response$
!        ...
!for all ids, prepare all modules for a frequency change.
OUTPUT <addrAll>;"FREQUENCY:SETUP 0,1,0.25"
! Master module asserts and releases SYNC line to move all modules to the new
! center frequency
OUTPUT <addrMaster>;"MEAS:CONTROL 0,1"
OUTPUT <addrMaster>;"MEAS:CONTROL 0,0"
```

The following example shows how to synchronously change the center frequency and reset the phase for all modules in a multi-module system without stopping a measurement in progress.

```
! For all ids, check status bits 0 and 1to assure that all modules are in
MEASURE or IDLE
! state. Changing frequency on modules in TRIGGER or ARM states is invalid.
!
OUTPUT <addrAll>;"status?"
ENTER  <addrAll>;Response$
!        ...
! Prepare all modules to change to zero frequency and phase.
OUTPUT <addrAll>;"FREQUENCY:SETUP 0,1,0.0"
 ! Master module asserts SYNC line to move all modules to the zero center
frequency and phase */
OUTPUT <addrMaster>;"MEAS:CONTROL 0,1"
OUTPUT <addrMaster>;"MEAS:CONTROL 0,0"
!        ...
!Master module asserts SYNC line to move all modules to the zero center
frequency and phase */
       ...
! Prepare all modules for a frequency change
OUTPUT <addrAll>;"FREQUENCY:SETUP 0,1,0.25"
! Master module asserts and releases SYNC line to move all modules to the new
center frequency
! while maintaining the phase
!
!Verify Sync Valid on Master
REPEAT
  OUTPUT <addrMaster>;"STATUS?"
  ENTER <addrMaster>;Oper_status
UNTIL BIT (Oper_status,7)
!
OUTPUT <addrMaster>;"MEAS:CONTROL 0,1"
OUTPUT <addrMaster>;"MEAS:CONTROL 0,0"
```

Reset Values: *cmplxdc*=OFF, *sync*=OFF, *freq*=0

See Also CLOCK:FS?, DATA:TYPE, CLOCK:MULTI:SYNC, MEAS:CONTROL

INPUT:AUTOZERO command

Nulls out the input DC offset voltage.

Command Syntax: INPUT:AUTOZERO

Description: INPUT:AUTOZERO updates a table of DC offset corrections to be used with each input setup condition. The applicable correction from this table is automatically added to the input offset parameter to achieve the correct DC offset value. Because of the length of time needed to execute this command, it is not automatically called when the module is reset. Thus, the user program is responsible for explicitly initiating the autozero. This command should be called at least once after the temperature of the module has stabilized. The interval between calls after that depends on the importance of DC accuracy in the user application. It is not necessary to call the autozero command for every change of input setup parameters since the correction table maintains values for all setup conditions.

NOTE Calling INPUT:AUTOZERO aborts any measurement already in progress and eliminates LO phase coherence and filter synchronization in a synchronous multi-module system. See the FREQUENCY:SYNC and FILTER:SYNC commands for details on how to re-establish LO phase and filter synchronization.

See Also INPUT:SETUP, FREQUENCY:SYNC, FILTER:SYNC

INPUT:RANGE:AUTO command

Performs auto-ranging.

Command Syntax: INPUT:RANGE:AUTO <sec>

sec::=<numeric>

numeric::=≥0 seconds

Description: This command sets the range of a E1437 to the lowest value that will not cause an ADC overload to occur. The algorithm will start at the lowest range and move up until there is no ADC overload.

Parameter definitions: is the time in seconds to take data at each range to insure that an overload is detected. Setting this parameter to 0.0 will result in this time being set automatically according to an algorithm that depends on block size and filter bandwidth.

NOTE An autorange that is pending or in progress will be aborted if an INPUT:RANGE or another INPUT:RANGE:AUTO command is received.

See Also INPUT:SETUP

INPUT:SETUP command/query

Sets all the analog input parameters. This description also includes information on the following commands which set or query the input parameters individually:

INPUT:ALIAS:FILTER selects the built-in analog anti-alias filter.

INPUT:ALIAS:FILTER? gets the anti-alias filter state

INPUT:COUPLING selects AC or DC input coupling.

INPUT:COUPLING? get the input coupling type

INPUT:FLOAT selects floating the input connector.

INPUT:FLOAT? gets the input connector state

INPUT:RANGE sets the full scale range.

INPUT:RANGE? gets the input range

INPUT:SIGNAL selects the input buffer amplifier.

INPUT:SIGNAL? gets the input buffer amplifier state

Command Syntax:

INPUT:SETUP <range>,<coupling>,<alias>,<signal>,<float>

range::=<numeric>

numeric::= INTEGERS 0 to 9

coupling::=0 | 1

alias::=0 | 1

signal::=0 | 1

float::=0 | 1

INPUT:ALIAS 0 | 1

INPUT:COUPLING 0 | 1

INPUT:FLOAT 0 | 1

INPUT:RANGE <numeric>

<numeric>::=0 to 9 (integer)

INPUT:SIGNAL 0 | 1

Query Syntax:

INPUT:ALIAS?

INPUT:COUPLING?

INPUT:FLOAT?

INPUT:RANGE?

INPUT:SIGNAL?

Example Statements:

```
OUTPUT 70924;"Input:setup 5,1,1,1,0"
OUTPUT 70924;"input:signal?"
```

Parameter Definitions:

determines whether or not to use the built-in analog anti-alias filter. It is recommended that the filter is always on to insure bandlimited, anti-aliased data.

parameter value	***alias*** **parameter definition**
0	OFF disables the anti-alias filter
1	ON inserts a sharp-cutoff (11-pole) 8 MHz lowpass filter ahead of the analog-to-digital converter.

specifies the AC or DC coupling mode of the input. Using DC will connect the input directly to the 50 Ohm buffer amplifier. AC inserts a 0.2 mF capacitor between the input connector and the 50 Ohm buffer amplifier.

parameter value	***coupling*** **parameter definition**
0	DC connects the input directly to the 50 Ohm buffer amplifier.
1	AC inserts a 0.2 μF capacitor between the input connector and the 50 Ohm buffer amplifier.

determines whether or not to allow the outer shield of the input connector to float relative to chassis ground. Using ON allows the connector to float in order to reduce potential ground loop induced pick-up at low frequencies. Using OFF disables floating by attaching the outer shield of the input connector directly to chassis ground. See the specifications section for more details.

parameter value	***float*** **parameter definition**
0	OFF disables floating by attaching the outer shield of the input connector directly to chassis ground. See the specifications section for more details.
1	ON allows the connector to float in order to reduce potential ground loop induced pick-up at low frequencies.

is a range index number between 0 and 9 which is transformed to a full scale voltage value. The corresponding discrete legal values of full scale vary from 0.02 volt to 10.24 volts with factor-of-two steps (.02 * 2^range). If range is greater than 9 the full scale value used is 10.24 volts. Non-integer values result in the next higher range. Signal inputs with an absolute value larger than full scale generate an ADC overflow error.

Range	Full scale voltage	Full Scale dBm
0	.02	–24
1	.04	–18
2	.08	–12
3	.16	–6
4	.32	0
5	.64	6
6	1.28	12
7	2.56	18
8	5.12	24
9	10.24	30

NOTE If an INPUT:RANGE:AUTO command is pending or in progress it is aborted when an INPUT:RANGE or INPUT_RANGE? command is received. INPUT_RANGE? also returns an error if an autorange is pending or in progress.

determines whether or not the input signal is sent to the buffer amplifier.

parameter value	***signal*** **parameter definition**
0	OFF redirects the input signal to a dummy 50 Ohm load, and feeds the buffer amplifier from an internally grounded 50 Ohm source resistance. The signal OFF setting is useful for making reference measurements without the signal applied. When using AC coupling the 0.2 μF capacitor remains between the input connector and its 50 Ohm termination.
1	ON attaches the input signal to the 50 Ohm buffer amplifier.

Comments:

To ensure full alias-free operation the analog anti-alias filter should be ON unless the application inherently bandlimits the input signal to less than fs/2. The analog anti-alias filter has a fixed bandwidth and thus is fully effective only when fs≥20 MHz. If a slower external ADC clock is used, an additional analog filter of the appropriate bandwidth may be required for full alias protection.

When using the analog anti-alias filter, the range parameter may need to be set higher than the actual range of the input signal. The reason for this is that step changes of input voltage cause an overshoot and ringing response at the output of the anti-alias filter. The peak overshoot will actually exceed the input voltage step by about 20%. The range setting must accommodate this overshoot to avoid an ADC overflow.

Reset Values: *range*=10.24, *coupling*=DC, *alias*=ON, *signal*=ON, *float*=OFF

See Also INPUT:RANGE:AUTO

INTERRUPT:RESTORE command

Restores the interrupt masks to the setting last programmed with INTERRUPT:SETUP.

Command Syntax: INTERRUPT:RESTORE

Example Statements: OUTPUT 70924;"Interrupt:restore"

Description: The interrupt masks set by the INTERRUPT:SETUP function are cleared during the interrupt acknowledge cycle. This function restores the cleared interrupt masks.

See Also: INTERRUPT:SETUP

INTERRUPT:SETUP command/query

Sets all interrupt parameters. This description also includes information on the following commands which query the interrupt parameters individually:

INTERRUPT:MASK? gets the interrupt event mask.

INTERRUPT:PRIORITY? gets the VME interrupt line.

Command Syntax: INTERRUPT:SETUP <intrNum>,<priority>,<mask>

IntrNum::=0 | 1

priority::=0 to 7

mask::=0 to 255

Query Syntax: INTERRUPT:MASK?

INTERRUPT:PRIORITY?

Example Statements: OUTPUT 70924;"Interrupt:setup 0,5,24"

OUTPUT 70924;"INTERRUPT:MASK?"

Description: An E1437 has two independent interrupt generators, each capable of interrupting on one of the seven VME interrupt lines when a status condition specified by a mask occurs.

INTERRUPT:SETUP sets the interrupt mask, priority and which of the two interrupt generators on the E1437 is to be used. The remaining INTERRUPT commands set or query the mask and priority individually.

Parameter Definitions: is the number of the interrupt generator. The only values accepted are 0 and 1.

specifies the mask of events on which to interrupt. This mask is created by ORing together the bits defined in bits 8 through 15 of the status register. The mask parameter format is 0xMM00 where MM represents the maskable upper 8 bits. The lower 8 bits cannot be used for generating interrupts, and therefore must be set to zero in the function call.

specifies which of the seven VME interrupt lines to use. The only legal values are 0 through 7. Specifying 0 turns the interrupt off, while 7 is the highest priority.

Comments: The mask is cleared during the interrupt acknowledge cycle. Therefore, the command must be sent again in order to generate further interrupts.

Reset Values *priority*=0, *mask*=0

See Also: STATUS?

LBUS:MODE command/query

Set and query local bus mode.

Command Syntax: LBUS:MODE <mode>

mode::=0|1|2|3

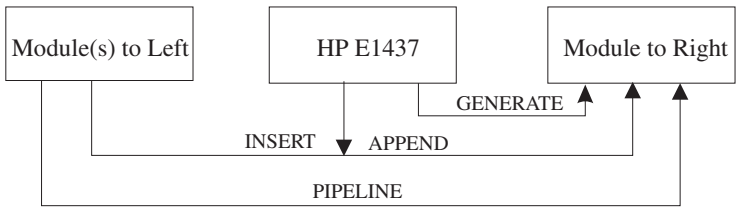
Query Syntax: LBUS:MODE?

Example Statements: OUTPUT 70924;"Lbus:Mode 2"

Description: LBUS:MODE sets the local bus to either generate, append, insert or pipeline data. The data port must be set to the local bus with the DATA:PORT command before these modes take effect.

Parameter Definitions: selects the transmission mode of the local bus when it is enabled by the DATA:PORT command. The state of this parameter is unaffected by switching back and forth between the local bus and the VME backplane with the DATA:PORT command.

parameter value	***mode*** **parameter definition**
0	PIPELINE causes the E1437 to pipe data through from modules on its left without appending or inserting its own data.
1	GENERATE forces the module addressed to generate data only, not passing through data from other modules on the local bus
2	APPEND causes the E1437 to pass through data from modules on its left and append its data to the end
3	INSERT causes the E1437 to place its data on the local bus and then pass through data from modules on its left.



Reset Values: *lbusMode*=PIPELINE

See Also: DATA:PORT

LBUS:RESET command/query

Resets local bus. Gets the current local bus reset state.

Command Syntax: LBUS:RESET <reset>

reset::=0 | 1

Query Syntax: LBUS:RESET ?

Example Statements: OUTPUT 70924;"Lbus:reset 1"

Description: In order to avoid glitches in the local bus data, the local bus interface has strict requirements as to the order in which modules in a VXI mainframe have their local bus interface reset. Upon powerup or whenever any single module in the mainframe is put into a reset state, all modules should be placed into the reset state from left to right. Then all modules can be take out of reset from left to right.

Parameter Definitions: puts the E1437's local bus into reset or takes it out of reset.

parameter value	***reset*** **parameter definition**
0	OFF takes the E1437 out of reset
1	ON puts the E1437's local bus into reset.

Example: When E1437s are used with the E1485 measurement controller, the E1485 must be reset while all of the E1437s are being held in reset to avoid initial glitches in the local bus data. The E1437s should be taken out of reset only after the first MEAS:CONTROL release is issued. The correct way to reset the local bus is as follows:

```
! For all modules hold HP E1437s in reset
      OUTPUT <addrAll>;"Lbus:Reset 1"
! Reset the E1485 lbus
      OUTPUT <id1485>;"LBUS:CONTROL 1,0"
! Set desired LBUS mode for all modules
!       .......
!
! For all id first arming
     OUTPUT <addrAll>;"Meas:control 0,1"
! Remove reset from HP E1437s, has no effect after first time
     OUTPUT <addrAll>;"Lbus:Reset 0"
```

Reset Values *reset*=ON

MEAS:CONTROL command

Initiates and controls measurements in a multi-module system.

Command Syntax: MEAS:CONTROL <idle>,<sync>

idle::=0 | 1

sync::=0 | 1

Example Statements: OUTPUT 70924;"Meas:Control 1,0"

Description: MEAS:CONTROL explicitly controls the measurement state.

Parameter Definitions: selects the condition of the IDLE state.

parameter value	*idle* parameter definition
0	RELEASE reverses a previous HPE1437_ASSERT or ensures that no forced IDLE is active.
1	ASSERT holds the module in the IDLE state.

MEAS:CONTROL also changes the state of the SYNC signal, which is used to arm or trigger an E1437 module. In systems containing multiple E1437 modules the SYNC signal is used to arm or trigger all modules simultaneously, and also to synchronize decimation counters and local oscillators among the E1437 modules.

selects the state of the sync signal. ASSERT causes the module to assert the SYNC signal. RELEASE causes the module to release the SYNC signal. When the sync parameter of the CLOCK:SETUP command is set to FRONT or REAR, the SYNC signal is shared with other E1437 modules. If any one of these modules asserts this shared SYNC signal then it becomes asserted for all of them. All modules must release it before the shared SYNC signal is released. Asserting then releasing the SYNC line is used to start a measurement, load local oscillator values, or take a digital filter out of reset. These situations require a SYNC line transition but do not require that the SYNC line be held in a asserted state.

parameter value	*sync* parameter definition
0	RELEASE causes the module to release the SYNC signal.
1	ASSERT causes the module to assert the SYNC signal.

NOTE

When the SYNC line is asserted, it will remain asserted for an adequate number of ADC clock cycles to ensure that the signal effect will have propagated to all the modules in the system. You can determine when the command is completed by looking as the Sync/Idle Complete bit in the Status Register.

Comments:

See The Measurement Loop section for details on how a measurement progresses through the four states.

Special conditions prevail during the Measure state. If programmed for block mode operation in the Measure state, the module will assert the SYNC signal (regardless of the MEAS:CONTROL *sync* parameter setting) until a complete block of data has been collected and is available to the I/O port. When the shared SYNC signal is released, indicating that all block mode data collection is finished, all block mode modules move synchronously to the idle state. In continuous mode the module releases the SYNC signal immediately after moving into the measure state. This allows the MEAS:CONTROL command to manipulate the SYNC signal to cause synchronous changes to LO frequency while a continuous measurement is in progress. In continuous mode a module moves to the idle state only if explicitly programmed to do so or whenever the FIFO data buffer overflows.

In addition to controlling the progression through the four module states, the SYNC signal is used to allow for synchronizing the decimation counters and local oscillators of multiple E1437 modules. This is done by calling FILTER:SYNC and/or FREQUENCY:SYNC prior to asserting SYNC with MEAS:CONTROL. This is normally done with the module in the IDLE state; however, the center frequency can also be changed in the Measure state with FREQUENCY:SYNC if the modules are all programmed for continuous (non-block mode) data collection.

If all modules in a multi-module system are in the idle state when the MEAS:CONTROL *sync* parameter is asserted, the LO frequency will be updated and the next measurement will be armed. If all modules are in the measurement state in continuous mode, the LO frequency will be synchronously updated, and the measurement will continue. In continuous mode care must be taken to ensure that all modules are in the same state, either the idle state or the measure state, before using MEAS:CONTROL to assert SYNC. Otherwise some modules will re-arm while others will continue the current measurement. In block mode the sync assertion will be ignored unless all modules are currently in the idle state.

In the case of systems made up of multiple mainframes you must be aware that only modules in mainframe A may assert sync. Any sync asserted in other mainframes is ignored.

Example: The following example shows how to initiate a measurement in a typical multi-module system

```
! Place all HP E1437s in IDLE
    OUTPUT <addrAll>;"MEAS:CONTROL 1,0"
!
! Take all HP E1437s out of IDLE
    OUTPUT <addrAll>;"MEAS:CONTROL 0,0"
!
! Check for Sync/Idle complete on last module (if decimation is synchronous);
! Check all modules if decimation is not synchronous.
    OUTPUT <addrAll>;"Status?
    ENTER <addrAll> Result$
          .....
! Assert SYNC on master module to arm all modules
    OUTPUT <addrMaster>;"MEAS:CONTROL 0,1"
!
!Release SYNC to allow triggering by any module
    OUTPUT <addrMaster>;"MEAS:CONTROL 0,0"
```

Reset Values: *idle*=RELEASE, *sync*=RELEASE

See Also: STATUS?, DATA:SETUP, FILTER:SYNC, FREQUENCY:SYNC, CLOCK:SETUP

MEAS:START command

Initiates a measurement in single-module systems.

Command Syntax: MEAS:START

Example Statements: OUTPUT 70924;"meas:start"

Description: MEAS:START provides an easy way to initiate a measurement in a single module system. This command moves the module through the IDLE state and the SYNC state while checking the status to assure a valid state.

Comments: See The Measurement Loop section for details on how a measurement progresses through the four states.

The meas:start command also checks status to assure that the module is in a valid state

Example: This example illustrates a simple measurement in a single module system

```
! Start a measurement
   OUTPUT <addr>;"MEAS:START"
! Read data
    OUTPUT <addr>;"READ"
    ENTER <addr>;Result$
```

See Also: STATUS?, CLOCK:SETUP

READ? query

Reads scaled data from FIFO

Query Syntax: READ?<samples>

samples::=1 to 8

Example Statements: OUTPUT 70924;"READ? 4"

Description: This command returns a block of floating point data from the E1437 that has been scaled to be in volts. The number of samples designated to be read must account for variations in blocksize, data type and resolution.

Data is returned as an ASCII string with points separated by commas. You can read up to 4 complex points or 8 real points per read command.

This command can only read data from the VME backplane register. The data port of the E1437 must be set to VME by the DATA:PORT command for this command to be effective. To read data using the local bus in an E1485 environment, see the documentation for local bus data transfers in the E1485 documentation package.

See Also: DATA:PORT, DATA:BLOCKSIZE

RESET command

Places the module in a known state.

Command Syntax: RESET

Example Statements: OUTPUT 70924;"Reset"

Description: This command returns the module and its internal data structures to the power-up state.

The reset values are listed with each command description.

The following are not affected by this command:

- Calibration constants

REVISION? query

Returns strings that identify the date of the firmware revision

Query Syntax: REVISION?

Example Statements:
```
OUTPUT 70924;"revision?"
ENTER 70924;rev$
```

Parameter Definitions: This command returns the date, time, and board number of the module's firmware revision

Return Format: <swrev0:swrev1:board#>

See Also *IDN?

STATUS? query

Reads Status Register information for the module.

Query Syntax: STATUS?

Example Statements: OUTPUT 70924;"Status?

ENTER 70924;Result$

Parameter Definitions: *Result$* contains the status word. The bits are defined below:

1-0 State: These two bits indicate the current state of the measurement loop as shown in the table below. See the Measurement Loop section for more information about the states

Bits	State
11	Trigger
10	Measure
01	Arm
00	Idle

2 Passed: This bit is always set to 1.

3 Ready: This bit is set whenever the module is operating as a message-based device and is set for Normal operation. See the VXIbus Specifications for more information on the Normal configuration sub-state.

4 ADC Error: This bit is set whenever a hardware error is detected in the ADC. The bit is cleared when the Status register is read.

5 Ext Clk Speed: This bit is set when a measurement has been aborted because the external clock is too fast (over 20.48 MHz) with respect to the DSP clock. This situation only occurs when a fast external ADC clock is used with an internal oscillator DSP clock. This bit is cleared with the first subsequent read.

6 Setup error: An invalid parameter value was requested. If an invalid block size was requested, the closest valid block size is used until a change to an interrelated parameter makes the requested block size valid. If a data resolution, data type, filter bandwidth, or filter decimation parameter was requested which would result in an inability to make a measurement, the previous valid parameter is used until a change to an interrelated parameter makes the requested parameter valid.

7 Sync/Idle Complete: This bit is set when the most recent user-initiated SYNC or IDLE change has propagated through to all modules in a system. The change is a result of asserting SYNC or forcing IDLE via the Control Register or issuing a MEAS:CONTROL command.

8 Read Valid: This flag is set whenever there is at least one valid 16-bit data word available to be read via the Data register.

9 Measure Done: This bit is set in continuous mode whenever the size of the data in the FIFO is equal to or greater than the block size register. Check this bit before reading data to insure that a block of data may be transferred without fear of running out of data, thereby holding up the Local bus or VME bus. This bit is set in block mode whenever the module has successfully taken a block size number of samples since the most recent trigger

10 Armed: This bit is set whenever the module is in the Trigger state, or is in the Arm state and has satisfied its pre-trigger requirements. When this bit is set, the module releases the VXI SYNC line. Once all modules release the SYNC line, then all modules go to the Trigger state.

11 FIFO Overflow: This bit set when the FIFO buffer overflows in continuous mode.

12 Overload: This bit is set whenever the ADC converts a sample that exceeds the range of the ADC. The bit is cleared when the Status register is read. Repeated ADC errors may indicate that the module should be recalibrated.

13 Error: This bit is set whenever there is an error in the error queue. It is cleared when the error queue is empty.

14 ModID*: A (1) in this field indicates that the module is not selected via the P2 MODID line. A (0) indicates that the module is selected by a high state on the P2 MODID line.

15 Hardware Set: This bit is set when all commands are complete and the hardware has been set.

TRIGGER:DELAY:ACTUAL? query

Returns the actual trigger delay from the most recent trigger event.

Query Syntax: TRIGGER:DELAY:ACTUAL?

Example Statements:

```
OUTPUT 70924;"trigger:delay:actual?
ENTER 70924;Result$
```

Parameter Definitions: *Result$* contains the returned actual delay from the most recent trigger event and the resulting first output sample time. This delay value provides more accuracy than the delay parameter alone since it includes a measurement of the fractional part of the output sample period between the actual trigger event and the next available output sample. The trigger delay accuracy improves to one ADC sample clock period rather than one output sample period. This can result in a substantial improvement in accuracy when narrow bandwidth decimation filtering is used. The this command must be sent for each new trigger event that requires precise delay measurement. The actual delay is still expressed in output sample periods, however, it can take on non-integer values.

See Also: TRIGGER:SETUP

TRIGGER:PHASE:ACTUAL? query

Returns a representation of the phase value of the LO at the trigger point.

Query Syntax: TRIGGER:PHASE:ACTUAL?

Example Statements: OUTPUT 70924;"trigger:phase:actual?

ENTER 70924;Result$

Parameter Definitions: *Result$* contains the returned value interpreted as follows:

0 <= value < 1.0

```
where  0 => 0 degrees
      .25 => 90 degrees
      .5 => 180 degrees
```

See Also: TRIGGER:SETUP, TRIGGER:PHASE:CAPTURE

TRIGGER:PHASE:CAPTURE command

Prepares for LO phase capture in frequency-synchronized, multiple-module zoom measurements.

Command Syntax: TRIGGER:PHASE:CAPTURE

Example Statements: OUTPUT 70924;"trigger:phase:Capture

Description: Use this function if you intend to subsequently use TRIGGER:DELAY:ACTUAL? to capture the LO phase on the next SYNC assertion. You should send TRIGGER:DELAY:CAPTURE to only one module in the system (typically the master) after you have completed all frequency and filter setup functions, since those functions take the module out of the phase_capture mode. Therefore, you should call TRIGGER:DELAY:CAPTURE just prior to starting the measurement.

When the FREQUENCY:SYNC mode is turned off, the TRIGGER:DELAY:CAPTURE function is not needed because the module will revert to the phase:capture mode by default.

See Also: TRIGGER:PHASE:ACTUAL?, TRIGGER_SETUP

TRIGGER:SETUP command/query

Sets all trigger parameters. This description also includes information on the following commands which set or query the trigger parameters individually:

TRIGGER:ADCLEVEL specifies the trigger threshold for an ADC trigger.

TRIGGER:ADCLEVEL? gets the ADC trigger threshold

TRIGGER:DELAY specifies a pre- or post-trigger delay time.

TRIGGER:DELAY? gets the trigger delay time

TRIGGER:GEN determines whether a module can generate a trigger.

TRIGGER:GEN? gets the trigger generation status

TRIGGER:MAGLEVEL specifies the trigger threshold for a magnitude trigger.

TRIGGER:MAGLEVEL? gets magnitude trigger threshold

TRIGGER:SLOPE selects a positive or negative trigger.

TRIGGER:SLOPE? gets trigger slope

TRIGGER:TYPE determines the trigger type.

TRIGGER:TYPE? gets trigger type

Command syntax:

TRIGGER:SETUP <type>,<delay>,<adclevel>,<maglevel>,<slope>,<gen>
  type::= 0 | 1 | 2 | 3 | 4
  delay <numeric>
    numeric::=0 to 6,777,216 sample periods
  adclevel <numeric>
    numeric::= –256 to +255
  maglevel <numeric>
    numeric::= –349 to 19
  slope::=0 | 1
  gen::=0 | 1
TRIGGER:ADCLEVEL <numeric>
  numeric::= –256 to +255
TRIGGER:DELAY <numeric>
  numeric::=0 to 6,777,216 sample periods
TRIGGER:GEN 0 | 1
TRIGGER:MAGLEVEL <numeric>
  numeric::= –349 to 19
TRIGGER:SLOPE 0 | 1
TRIGGER:TYPE 0 | 1 | 2 | 3 | 4

Query syntax: TRIGGER:ADCLEVEL?

TRIGGER:DELAY?

TRIGGER:GEN?

TRIGGER:MAGLEVEL?

TRIGGER:SLOPE?

TRIGGER:TYPE?

Example Statement: OUTPUT 70924;"Trigger:setup 1,256,25.6,0,0,1"

OUTPUT 70924;"trigger:type?"

Description: An E1437 can be triggered to collect data in a variety of ways. The trigger can be internally generated or can come from an external source. Multiple modules can be triggered synchronously. A variable pre- and post-trigger delay can be programmed for data collection. The slope and level of the trigger point on a signal can be selected. The source of the internal trigger can be either the output of the ADC or the magnitude of the complex output of the decimation filter.

TRIGGER:SETUP is the command that sets all trigger parameters at once. An E1437 will generate a trigger only when it is in the TRIGGER state and the SYNC line on the VXI backplane is released. When a trigger is generated, the E1437 will release the SYNC line.

Parameter Definitions: determines the trigger source.

parameter value	*type* parameter definition
0	USER disables the module from any event-driven trigger generation though it is still possible to force the module to trigger a measurement by pulling the SYNC line once the module is in the trigger state. You may do this by calling the MEAS:START function, waiting for the module to reach the trigger state, then triggering the measurement by using MEAS:CONTROL to pull the SYNC line.
1	ADC generates a trigger based on the raw data samples from the ADC
2	EXTERNAL uses transitions on the signal applied to the BNC external trigger connector on the front panel.
3	MAG generates a trigger based on the log magnitude of the signal after it has been filtered to a selectable bandwidth around the center frequency established by the FREQUENCY:SETUP function.
4	IMMEDIATE triggers a measurement immediately upon entering the trigger state.

NOTE In multi-module systems all modules should be of the same type in order to have the same actual delay.

is the time delay, in units of output samples, between when a trigger is received and the first data point in the output data. Negative values indicate a pre-trigger condition, where samples prior to the trigger event are included in the output data. The amount of pre-trigger delay is limited to the number of samples which can be saved in the 8 Mbyte buffer memory. See the DATA:SETUP command description for the number of bytes used per sample. Valid values depend on data type as follows:

Trigger Delay
(DRAM size in bytes)

	32 bit complex	32 bit real 16 bit complex	16 bit real
Post-trigger	16,777,116	33,554,332	67,108,764
Pre-trigger	132 – DRAMsize/8	164 – DRAMsize/4	228 – DRAMsize/2

If *delay* is < 132 – DRAMsize/8 or > 16777116 the software will set a bad parameter error. However, the delay is still programmed in order to accommodate valid setups for other data types for which larger values are valid..

adclevel is used to set the triggering signal threshold when using the ADC trigger source. This threshold is (full scale * adclevel/256), where -256 ≤ adclevel ≤ 255. There is hysteresis around the threshold in order to prevent multiple triggers from a single threshold crossing.

is used to set the triggering threshold when using the mag trigger source. The threshold is (+0.3762874 * maglevel)dB relative to full scale signal, where -349 ≤ maglevel ≤ 19.

selects the edge of the trigger source on which a trigger occurs.

parameter value	***slope*** **parameter definition**
0	POSITIVE sets triggering on the positive slope
1	NEGATIVE sets triggering on the negative slope

determines whether a module may generate a trigger.

parameter value	*gen* parameter definition
0	OFF disables triggering.This is useful in multi-module systems with the same trigger type where you want only certain module(s) to generate a trigger.
1	ON enables triggering

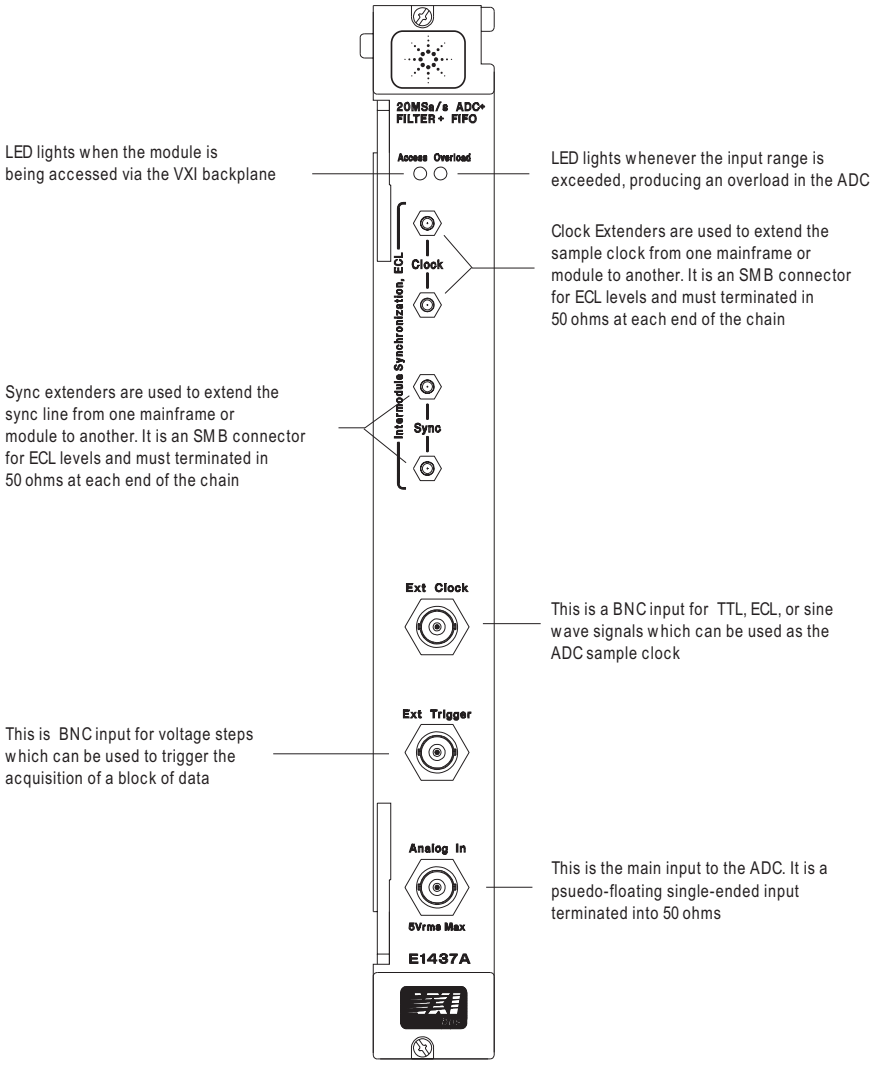
Reset Values: *type*=IMMEDIATE, *delay*=0, *adclevel*=0, *maglevel*=–128, *slope*=POSITIVE, *gen*=ON

See Also: FREQUENCY:SETUP, DATA:SETUP, FILTER:DECIMATE, MEAS:START, MEAS:CONTROL, TRIGGER:DELAY:ACTUAL?

6

Module Description

Front Panel Description



VXI Backplane Connections

Power Supplies and Ground

The E1437A conforms to the VME and VXI specifications for pin assignment. The current drawn from each supply is given in Technical Specifications.

Data Transfer Bus

The E1437A conforms to the VME and VXI specifications for pin assignment and protocol. Only A16/D16 data transfers are supported. Thus the upper address and data bits are ignored.

DTB Arbitration Bus

The E1437A module is not capable of requesting bus control. Thus it does not use the Arbitration bus. To conform to the VME and VXI specifications, it passes the bus lines through.

Priority Interrupt Bus

The E1437A generates interrupts by applying a programmable mask to its status bits. The priority of the interrupt is determined by the interrupt priority setting in the control register.

Utility Bus

The VME specification provides a set of lines collectively called the utility bus. Of these lines, the E1437A only uses the SYSRESET* line.

Pulling the SYSRESET* line low (a hardware reset) has the same effect as setting the reset bit in the Control Register (a software reset), with two exceptions. The exceptions are:

- The Control Register is also reset.
- All logic arrays are reloaded.

Reloading the logic arrays enables the hardware reset to recover from power dropouts which may invalidate the logic setup.

Local Bus

The VXI specification includes a 12-wire local bus between adjacent module slots. Using the local bus, Hewlett-Packard has defined a standard byte-wide ECL protocol that transfers data from left to right at up to 100 Mbyte/s. The E1437A can be programmed to output its data using this high speed port instead of the VME data output register. The Data Port Control register determines which output port is used.

Trigger Lines

The VXI specification provides 8 TTL and 2 ECL trigger lines which can be used for module-specific signaling. When programmed in a multi-input configuration, the E1437A uses the ECL trigger lines, designating ECLTRG0 as the SYNC line and ECLTRG1 as the ADC sample clock (CLOCK). These lines can be extended to other mainframes using the SMB connectors on the front panel. The SMB connectors can also be used for intermodule synchonization within a mainframe, leaving the ECL trigger lines free for other purposes.

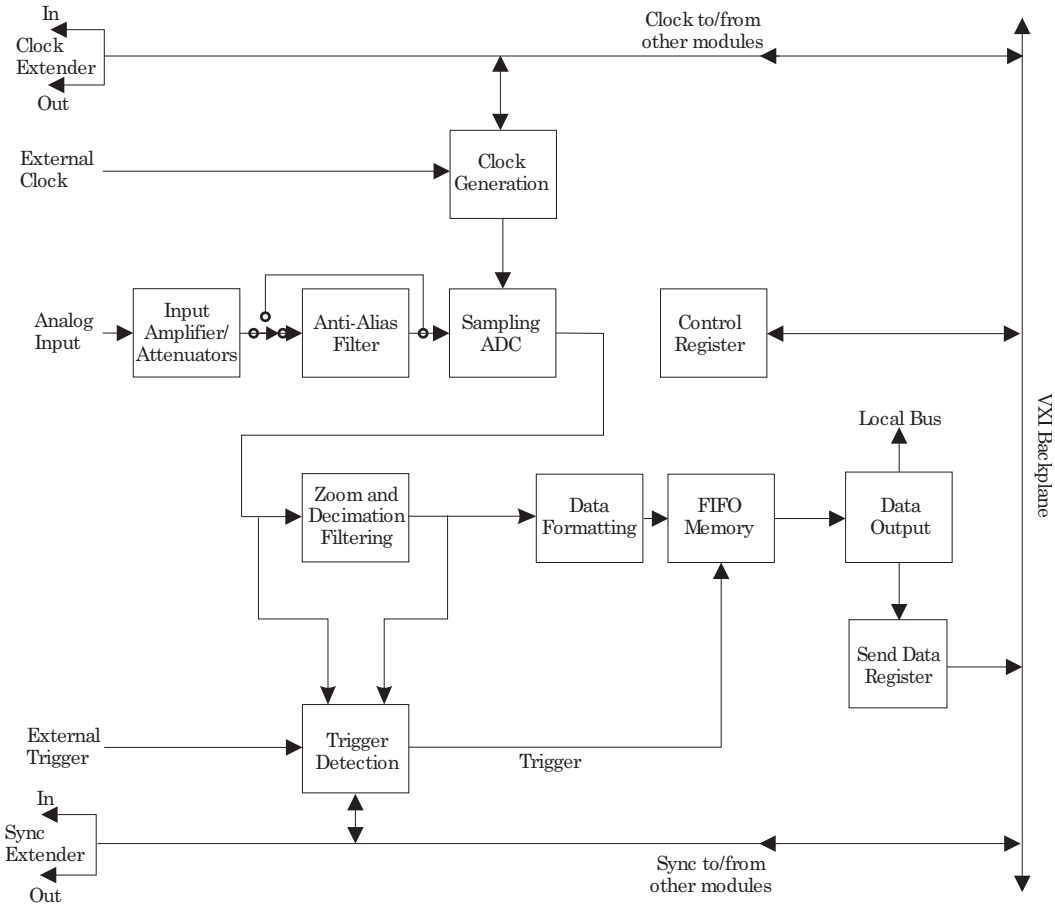
The CLOCK line is the master ADC clock for a synchronous system of multiple E1437A modules. Only one E1437A module in each mainframe is allowed to drive this line.

The SYNC line is used to send timing signals among E1437A modules in a multi-input system. Any module which drives this line must do so synchronously with CLOCK so that transitions on SYNC do not occur near the rising edge of CLOCK. This ensures that all modules with a synchronous state machine clocked on CLOCK will interpret SYNC in a consistent manner for each cycle of the state machine. SYNC is used for synchronizing, arming, and triggering signals between E1437A modules. The interpretation of the SYNC line is dependent on the states of the module described in the Measurement Loop section. The E1437A module is also capable of controlling the SYNC line synchronously via the control register.

Block Diagram and Description

Descriptions of sections in the diagram below appear on the following pages.

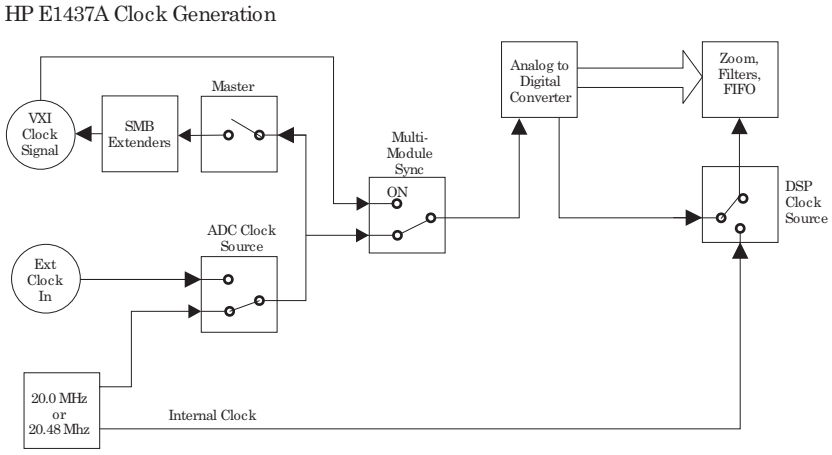
HP E1437 Block Diagram



Clock Generation

The usual source for a clock signal is the 20 MHz or the 20.48 MHz crystal oscillator inside the E1437A. However, the E1437A can also accept an external clock signal through a front-panel BNC (“Ext Clock”). This signal can be TTL, ECL, or sine wave.

In a system using more than one E1437A, the ADCs can be synchronized by programming them to use a common ECL line on the backplane. One of the modules can be the clock master that drives this line. This master clock can be extended to other mainframes by connecting a “Clock” SMB connector to a “Clock” SMB connector on an E1437A in the second mainframe.



Input Amplifier

The input amplifier provides an input termination which maintains good flatness to 8 MHz. The gain/attenuation of the input amplifier is programmable.

Under program control, the input signal can be ac coupled. This allows the system to measure low level ac signals in the presence of a large dc offset. .

Anti-alias Filter

Since the normal ADC sample rate is 20 MHz, a complete representation of the input signal can be achieved only for bandwidths up to 10 MHz. Frequency components above 10 MHz can cause ambiguous results (aliasing).

The anti-alias filter attenuates these high frequency components to reduce aliasing. The anti-alias filter in the E1437A is flat to 8 MHz and rejects signals above 12 MHz by at least 100 dB. Thus the 0-8 MHz frequency range of the sampled signal will be alias free. The filter’s transition band from 8 MHz to 12 MHz will affect flatness and allow some aliasing in the sampled signal frequency range of 8 MHz-10 MHz.

In cases where alias filtering is not necessary the E1437A can be programmed to bypass the anti-alias filter. This allows the system to take advantage of the full 40 MHz sampler bandwidth. To avoid incorrect results, the alias filter bypass mode should be used with caution; it is not recommended for normal operation.

Sampling ADC

The heart of the E1437A is a precision Analog-to-Digital Converter (ADC). The ADC generates 23 bit outputs at a sample rate up to 20.48 MHz. It has very low noise density and very low distortion levels.

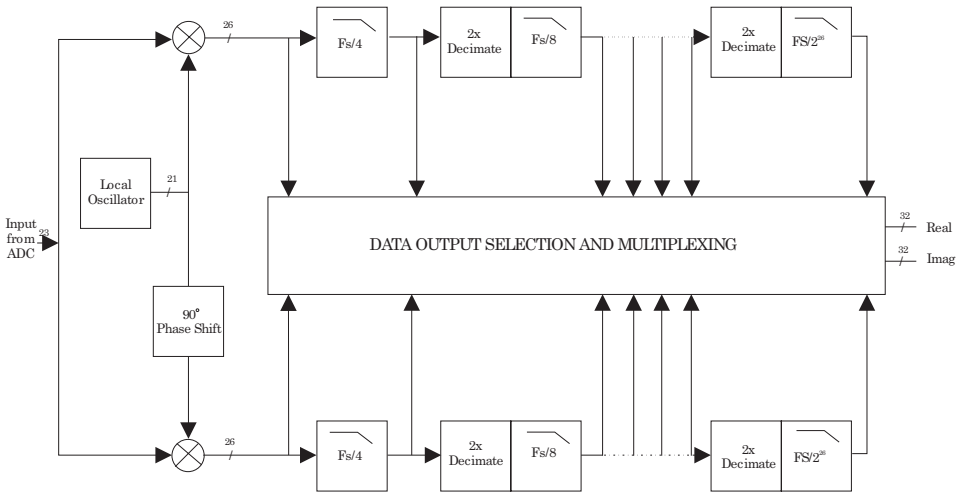
Zoom and Decimation Filtering

This section uses digital circuitry to allow programmable changes in the center frequency and signal bandwidth of the E1437A (zoom). This is done at high speed for real-time operation.

Bandwidth is controlled by a chain of digital low-pass filters (see the diagram below). Each of the filters reduces the bandwidth by a factor of two (decimation). With the ADC sample rate (Fs) set to the standard internal 20.48 MHz rate, the bandwidth choices are 10 MHz, 5 MHz, 2.5 MHz,...0.289 Hz around the programmed local-oscillator (LO) frequency.

Real and imaginary components of the signal are each computed to 32-bit precision, so the complex output of the decimation filtering block contains 64 bits. Whether or not all of these bits are stored in memory is programmable.

Zoom and Decimation Filtering



Data Formatting and FIFO Memory

The E1437A can be programmed to save the real component of the signal or to save the complete complex signal. The data precision can be set to 16 bits or 32 bits. Thus, each sample will occupy from two to eight bytes of memory in the FIFO. The data formatting block packs the selected data into 64-bit words which are stored in the FIFO memory. Since the standard FIFO depth is 1-Mword (8 MByte), it is possible to hold up to 4-Msamples in memory at one time.

The memory may be configured either in block mode or in continuous mode. In block mode, data collection initiated by a trigger will proceed until a specified block length is captured. The measurement is then paused so that the data can be read out. Before a new block can be collected, the module must be re-armed and triggered again. This mode is useful in capturing single transient events or whenever the output data rate is too high to be read and processed in real time.

In continuous mode, data collection is initiated by a trigger and will continue as long as the FIFO does not overflow. Data may be read out of the memory while the measurement is in progress. If the reading of data is sufficiently fast, the FIFO will never overflow and the measurement will continue indefinitely. If the FIFO should ever overflow then the measurement will stop and wait for data to be read out, the measurement to be re-armed, and a new trigger. This mode of operation is useful for real-time applications that employ a high speed signal processor to continuously read and operate on each sample of data. Data can be read from the FIFO in bursts to accommodate pauses for such things as disk access times or block mode computations.

The effective trigger time may be offset from the actual trigger event by programming a trigger timing offset. See the Technical Specifications for the limits of the pre-trigger and post-trigger offset.

Data Output

There are two ways to output data from the E1437A: by way of the VXI backplane or by way of the local bus.

To use the VXI backplane, the E1437A can be programmed so that the output of the FIFO is sent to the Send Data register. Each 64-bit portion of the FIFO memory is sent to the 16-bit register as four separate words. The register can then be read by any controller compatible with the VME standard. Maximum data flow is about 2 MB/s.

The local bus allows data transfers over a high speed 8-bit ECL bus to an adjacent module (to the right) in the VXI mainframe. Multiple adjacent E1437A modules can send data to one signal processor module. The signal processor must be one which supports the Hewlett-Packard ECL local bus protocol, such as the E1485A/B. In addition to higher speed (up to 40 MB/s), the local bus has the advantage that data can be output at the same time that control signals are being sent over the VXI backplane.

In both of the data output modes, the samples must be read out sequentially, offset by the trigger delay.

Trigger Detection

The trigger event used to start a measurement can be generated in five different ways:

- Software trigger
- External
- ADC threshold
- Log-magnitude
- Immediate

All triggering modes support slope selection. In ADC or log-magnitude mode the trigger threshold can be specified with hysteresis to prevent noise-generated triggers of the wrong slope. Log-magnitude triggering is based on the magnitude of the complex signal after zooming and filtering.

For external mode, a trigger signal must be supplied at the “Ext Trigger” connector on the front panel. Any signal with a sharp rising or falling transition greater than 100 mV (i.e. TTL or ECL) can be used as an external trigger source.

Any E1437A module can trigger other E1437A modules using a shared sync line on the VXI backplane. This SYNC line can be extended to other mainframes by connecting a “Sync” SMB connector to a “Sync” SMB connector on a E1437A in the second mainframe. All modules in a synchronous system are triggered on the same ADC sample.

The E1437A hardware samples the trigger source once every sample clock, so the trigger condition must be present for at least one sample clock in order to be recognized.

Control Registers

The E1437A module is controlled by firmware using registers mapped into the 16-bit VXI address space. There are 24 writable and 18 readable registers, each has 16 bits. The control registers are not user accessible.

7

Verifying the E1437A

To verify the E1437A

You may perform a quick verification of the basic functions of the E1437A by performing the built-in self-test function. The self-test verifies the following:

- Digital filtering, zooming, and decimation at full scale voltage range
- Front-end noise specification
- Front-end levels associated with the analog-to-digital converter
- Integrity of the installed memory including all memory options
- Autozero and input triggering

The test is available as:

- the **hpe1437_self_test** function for Windows VXI*plug&play* and HP-UX C language programmers
- the ***TST?** command for ASCII programmers
- a Soft Front Panel selection from the Control menu

See the online help, “E1437A VXI*plug&play* Programmer’s Reference” or “ASCII Overview and Commands” for syntax and details.

8

Replacing Assemblies

Replaceable Parts

For information on upgrading your module or replacing parts, contact your local Agilent Technologies sales and service office. See the Technical Specifications or the Agilent Technologies web site (http://www.agilent.com/find/tmdir) for a list of office locations and addresses.

Ordering Information

To order Agilent Technologies, Inc. parts in the U.S., call Agilent Technologies, Inc. Parts Direct Ordering at (800) 798-5487. Outside the U.S., please contact your local Agfilent Technologies, Inc. parts center.

Caution

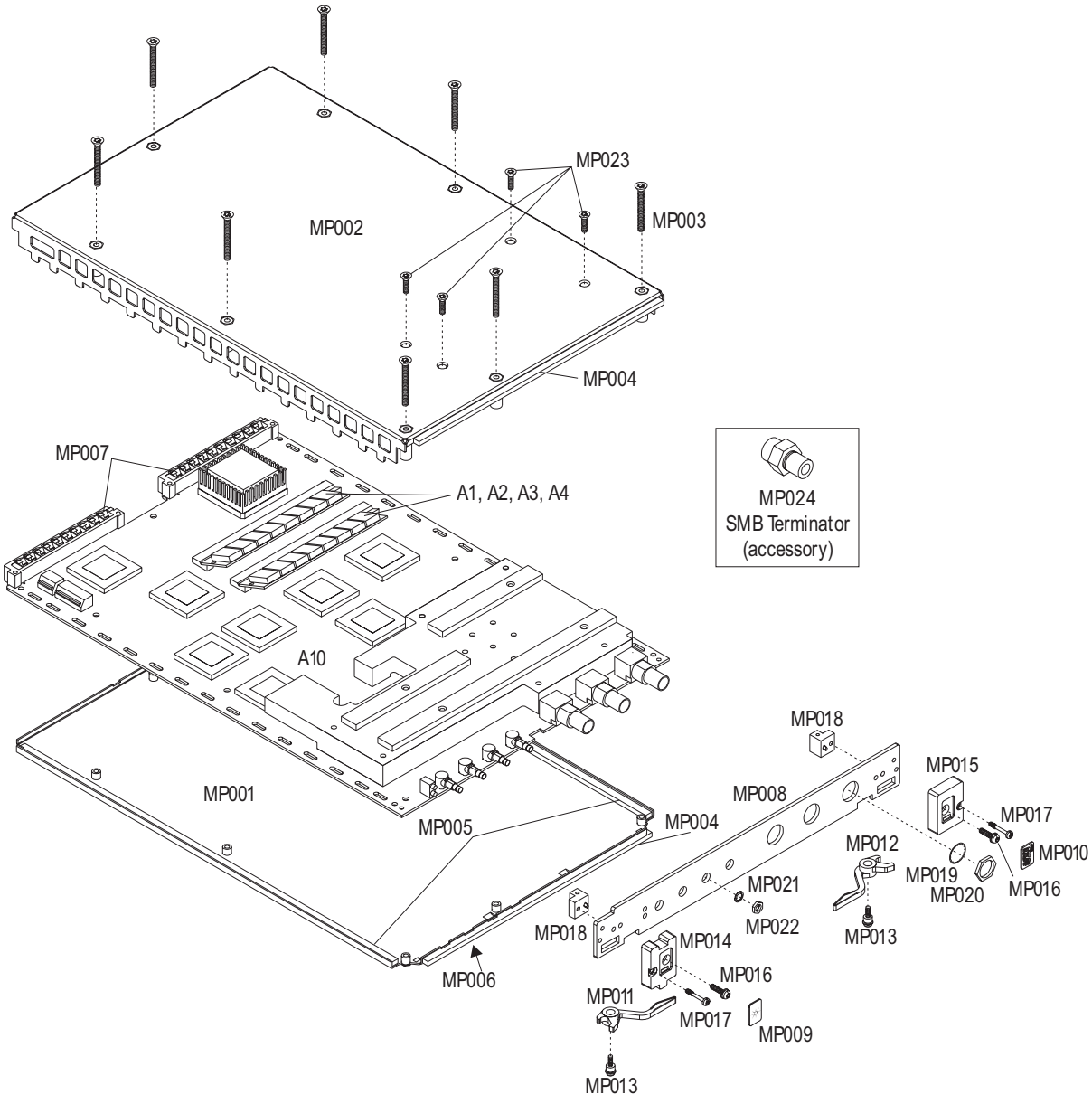
The module is static sensitive. Use the appropriate precautions when removing, handling, and installing to avoid unnecessary damage.

Code Numbers

The following table provides the name and location for the manufacturers' code numbers (Mfr Code) listed in the replaceable parts tables.

Mfr No.	Mfr Name	Location
28480	Agilent Technologies, Inc.	Palo Alto, CA U.S.A.
30817	Instrument Specialties Co. Inc.	Placentia, CA U.S.A.
13940	Smart Modular Technologies	Fremont, CA U.S.A.
02788	M/A-Com Inc.	Burlington, MA U.S.A.
04637	Phelps Dodge Corp.	New York, NY U.S.A.

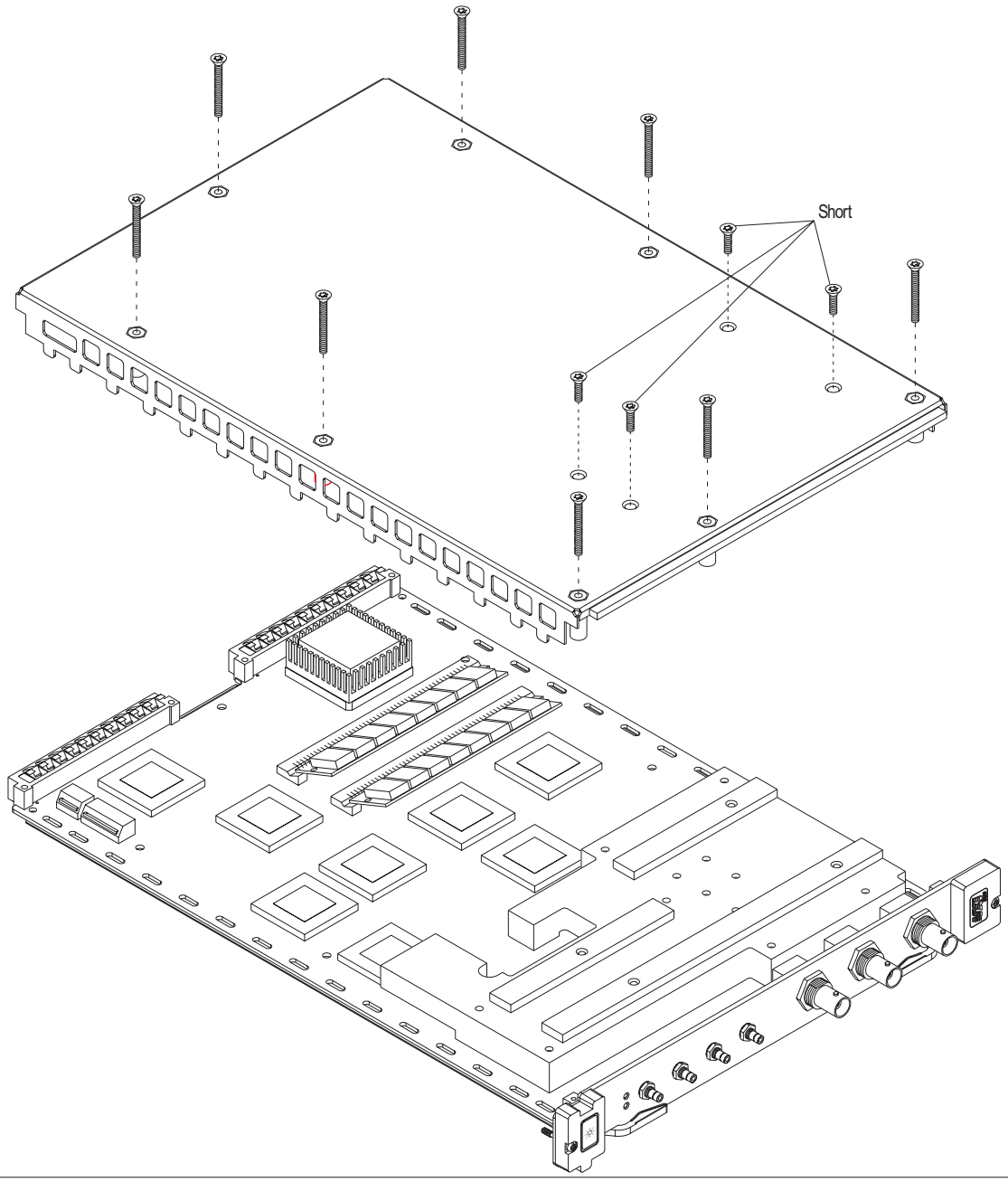
Assemblies


MP023
MP003
MP002
MP004
MP007
A1, A2, A3, A4
MP024
SMB Terminator
(accessory)
A10
MP018
MP015
MP008
MP017
MP012
MP010
MP019
MP020
MP016
MP001
MP005
MP004
MP021
MP018
MP022
MP013
MP014
MP016
MP006
MP011
MP017
MP009
MP013

Ref Des	Part Number	CD	Qty	Description	Mfr Code	Mfr Part Number
A1	1818-6722	1	2	ICM DRAM, 4MB	13940	SM5361000-6
A2	1818-6828	8	2	ICM DRAM, 8MB Opt. UFC	13940	SM536023101P4S6
A3	1818-6728	7	2	ICM DRAM, 16MB Opt. ANC	13940	SM536044002P3S6
A4	1818-6649	1	2	ICM DRAM, 32MB Opt. ANE	13940	SM536084002Q3S6
A10	E1437-69510	3	1	PC ASSEM. EXCHANGE BRD.	28480	E1437-66510
MP001	E1437-00203	7	1	SHTF CVR-BTTM	28480	E1437-00203
MP002	E1437-00202	6	1	SHTF CVR-TOP	28480	E1437-00202
MP003	0515-1135	7	8	SCR-MCH M3.0 25M	28480	0515-1135
MP004	E1485-40602	2	2	GSKT RFI-FRONT PANEL	28480	E1485-40602
MP005	E1485-40601	1	2	GSKT-RFI, BOTTOM COVER	28480	E1485-40601
MP006	8160-0686	6	2	STMP FINGERS-RFI	30817	786-185
MP007	E1450-01202	5	4	STMP SHLD-RFI GRND	28480	E1450-01202
MP008	E1437-00204	8	1	PANEL-FRONT, “E1437A”	28480	E1437-00204
MP009	7121-7964	6	1	LABEL-HP, LOGO	28480	E1400-84308
MP010	7121-7893	5	1	LABEL-VXI, LOGO	28480	E1400-84307
MP011	E1400-45102	6	1	MOLD, HANDLE RIGHT	28480	E1400-45102
MP012	E1400-45101	5	1	MOLD, HANDLE LEFT	28480	E1400-45101
MP013	E1400-00610	7	2	SCR-ASM SHLDR	28480	E1400-00610
MP014	E1400-45011	6	1	MOLD LBUT-ECL	28480	E1400-45011
MP015	E1400-45008	1	1	MOLD BOTTOM-LOGO	28480	E1400-45008
MP016	0515-0664	5	2	SCR-MCH M3.0 12M	28480	0515-0664
MP017	0515-2733	3	2	SCR-MCH M2.5 17	28480	0515-2733
MP018	E1400-40104	8	2	CAST	28480	E1400-40104
MP019	2190-0068	5	3	WSHR-LK .50 NTT	28480	2190-0068
MP020	2950-0154	2	3	NUT-HXP .50-28 .0	28480	2950-0154
MP021	2190-0124	4	4	WSHR-LK #10 NTT	02788	500222
MP022	2950-0078	9	4	NUT-HXP 10-32 .0	04637	500220
MP023	0515-1946	8	4	SCR-MCH M3.0 6MM	28480	0515-1946
MP024	1250-0676	8		TERMN-COAX CONN; 50 Ω	28480	1250-0676

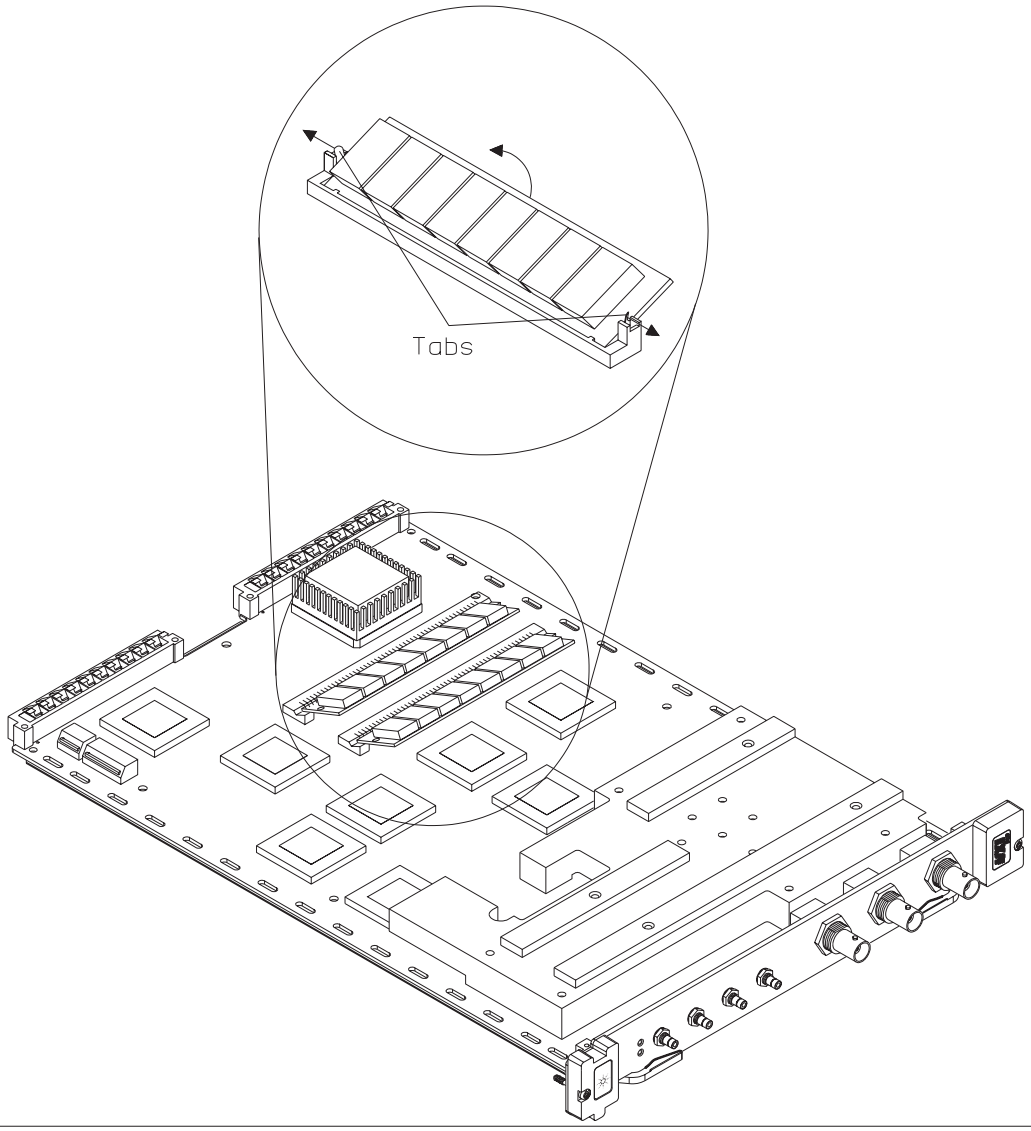
To remove the top and bottom covers

1 Remove the four short and eight long screws using a T-10 torx driver and remove the covers.



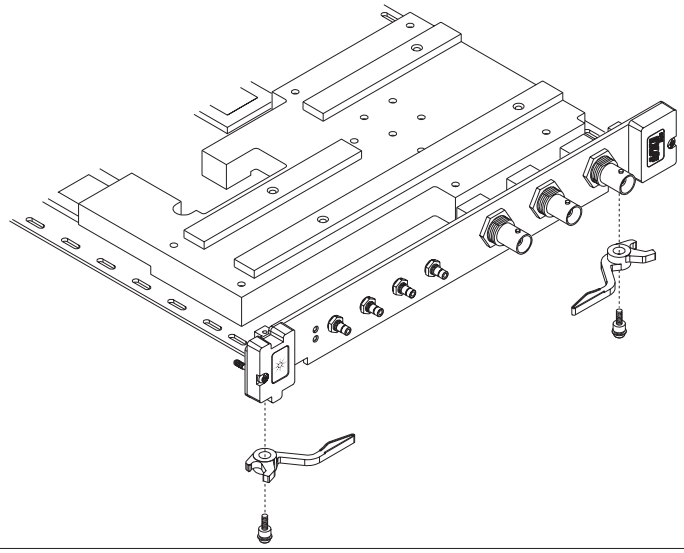
To remove the A1, A2, A3 or the A4 assembly

1 Remove top cover, see “To remove the top and bottom covers.” Gently push the silver tabs outward and tilt the assembly forward releasing it from the connector.

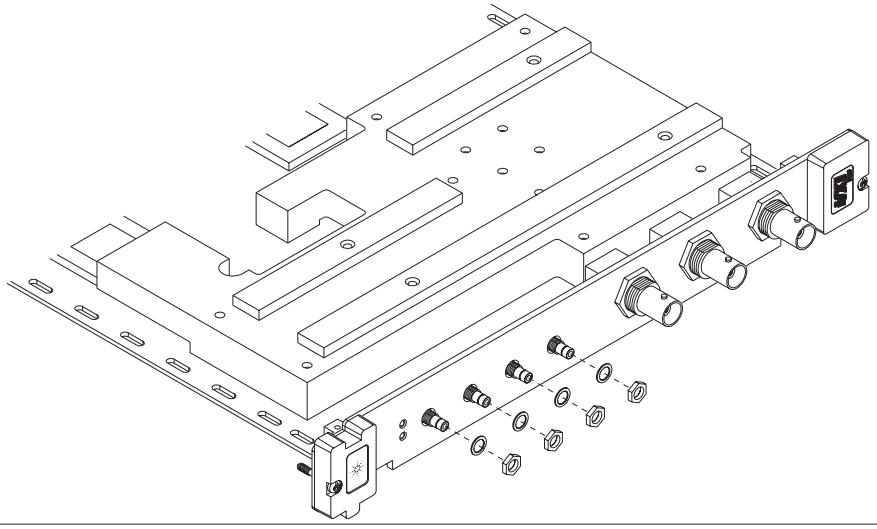


To remove the front panel

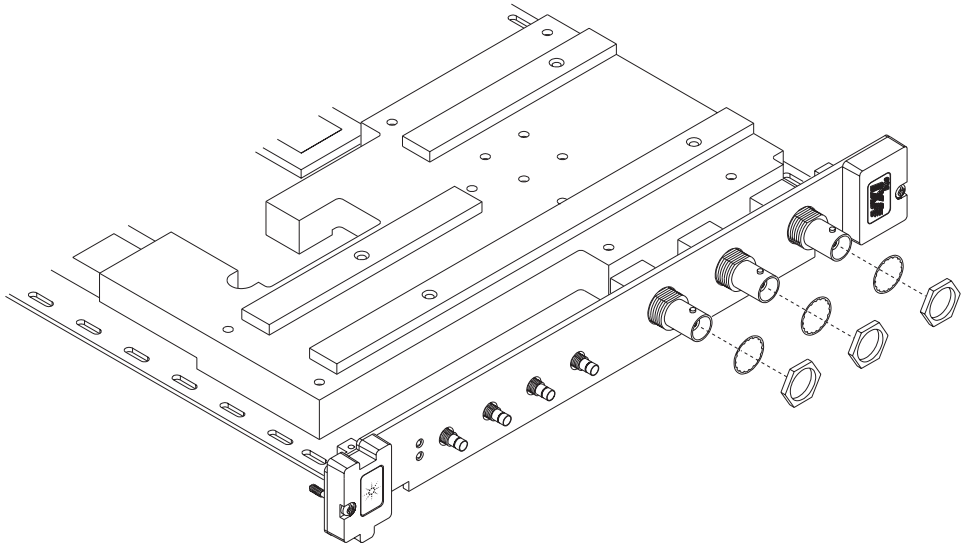
1 Remove covers, see “To remove the top and bottom covers”. Using a T-8 torx driver, remove the two screws that attach the handles to the assembly. *NOTE: be sure to label the two handles, which are different from each other. This will aid you in reassembling the module.*



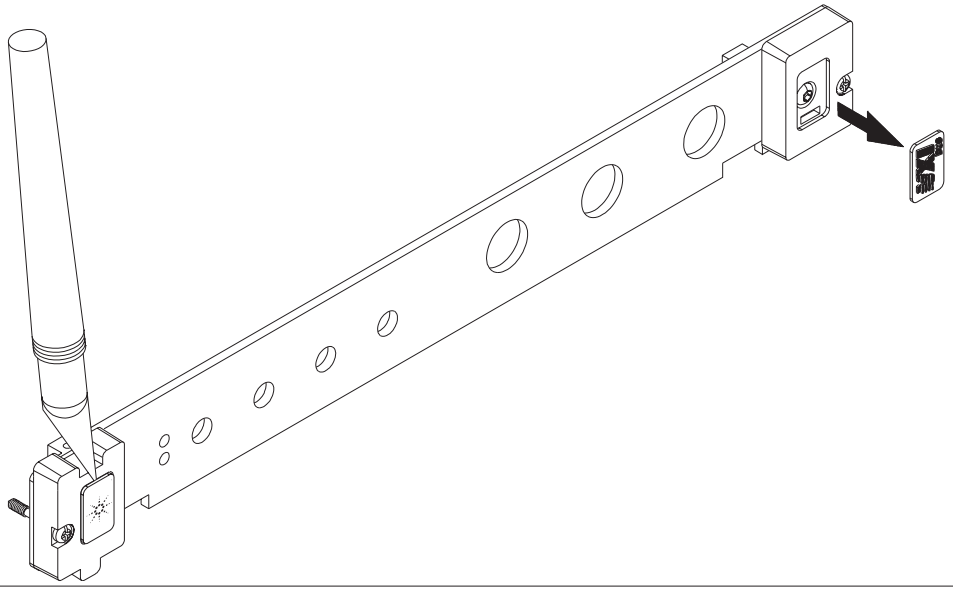
2 Remove the 4 nuts and washers from the gold connectors as shown, using a 1/4" nut driver.



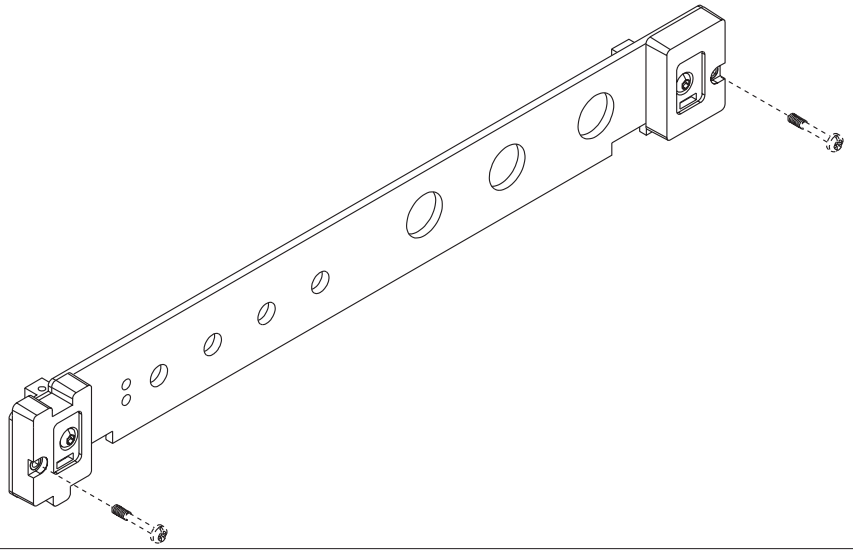
3 Remove the 3 nuts and washers from the BNC connectors as shown, using a 9/16" nut driver. Slide the front panel off the main assembly.



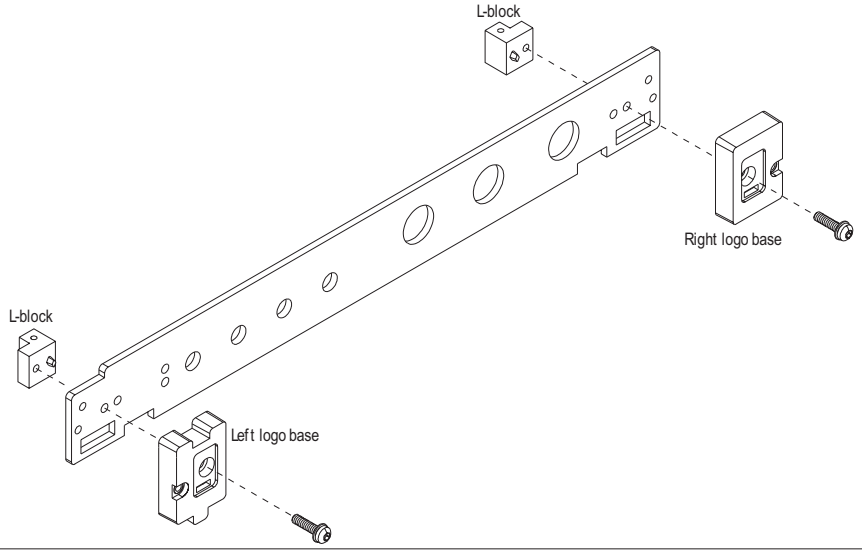
4 *Note: steps 4, 5, and 6 are only necessary if you need to replace the front panel or any of it's components.* Using an X-acto knife, gently pry the labels from the two keys.



5 Using your hand, remove the two captive screws.

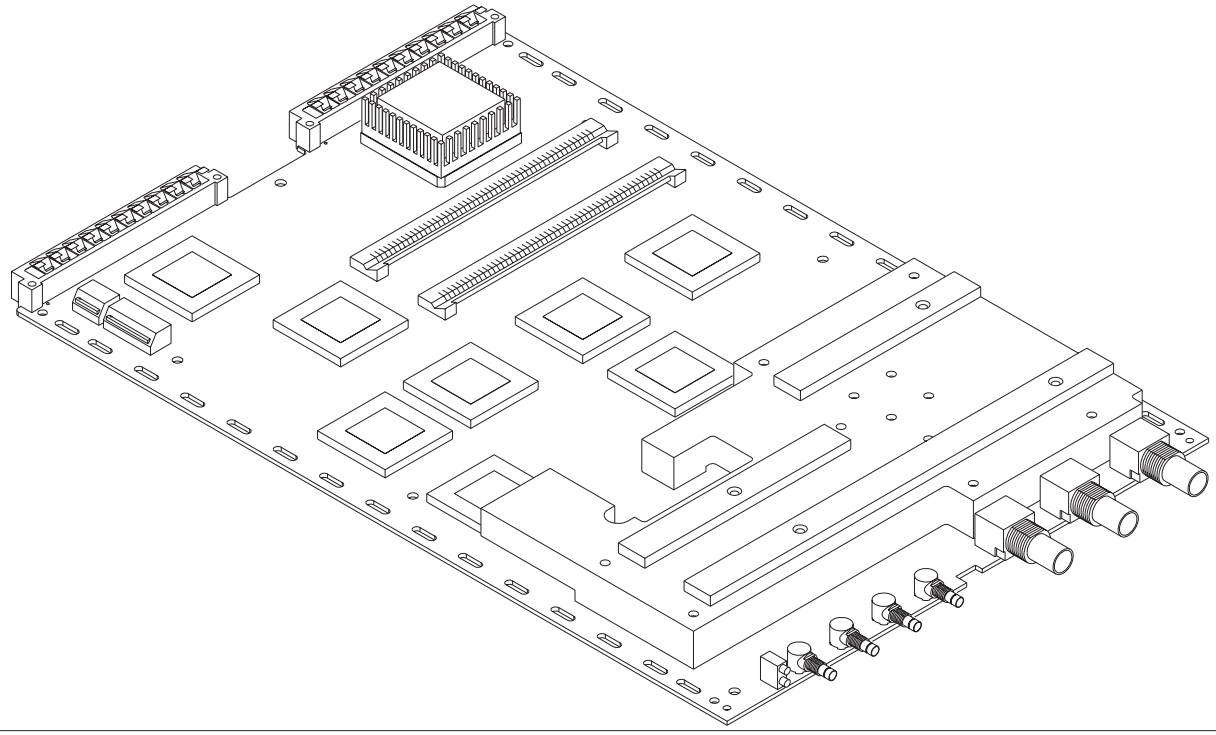


6 Using a T-10 torx driver, remove the two screws that attach the two logo bases and the two L-blocks to the front panel. *Note: there is a left and a right logo base. Also notice the orientation of the two L-blocks. This will be important when you reassemble the front panel.*



To remove the A10 main assembly

1 Remove covers, see “To remove the top and bottom covers”. Remove the SIMMS, see “To remove the A1, A2, A3, or the A4 assembly”. Remove the front panel, see steps 1, 2 and 3 of the “To remove the front panel” section.



9

Backdating

Backdating

This chapter documents modules that differ from those currently being produced. With the information provided in this chapter, this guide can be modified so that it applies to any earlier version or configuration of the module.

Glossary

ADC
Analog to Digital Converter

ASCII
American Standard Code for Information Interchange, a standard format for data or commands.

backplane
A set of lines that connects all the modules in a VXI system.

baseband
A band in the frequency spectrum that begins at zero. In contrast a zoomed band is centered on a specific center frequency.

block mode
A mode in which the HP E1437A stops taking data as soon as a block of data has been collected.

block size
The number of sample points in a block of data.

continuous mode
A mode in which the HP 1437A collects data continuously. It does not stop taking data unless the FIFOFIFO overflows.

decimation filter
A digital filter that simultaneously decreases the bandwidth of the signal and decreases the sample rate. The digital filter provides alias protection and increases frequency resolution. For more information, see Spectrum and Network Measurements available through your Hewlett-Packard Sales Office.

DSP
Digital Signal Processing

FIFO
A First In, First Out buffer and controller used to transmit data.

Fs
Sample Frequency or sample rate

HP-VEE

A Hewlett-Packard program for graphical programming

Local Bus

A high-speed port that Hewlett-Packard has defined as a standard byte-wide ECL protocol which can transfer measurement data from left to right at up to 2.62 Msamples per second on the VXI backplane.

logical address

The VXI logical address identifies where each module is located in the memory map of the VXI system.

VXI

VME Extensions for Instrumentation, a standard specification for instrument systems

VXI*plug&play*

A set of standards which provides VXI users with a level of standardization across different vendors beyond what the VXI standard specifications spell out.

zoom

Selects a frequency span around a specified center frequency. Also known as band selectable operation, this allows you to focus on a specific frequency band.

Need Assistance?

If you need assistance, contact your nearest Agilent Technologies Service Office listed in the Agilent Catalog, or visit our web site: http:// www.agilent.com/find/tmdir for a current sale office listing. If you are contacting Agilent Technologies about a problem with your E1437A 20 MSample/second ADC, please provide the following information:

- Model number: E1437A
- Software version:
- Serial number:
- Options:
- Date the problem was first encountered:
- Circumstances in which the problem was encountered:
- Can you reproduce the problem?
- What effect does this problem have on you?

About this edition

January 1997: First Edition

June 1997: Second Edition.

April 2000: Third Edition - Rebranding, Hewlett-Packard to Agilent Technologies, Inc.

INDEX

E

F

G

H

I

T

U

V

W

Z