Agilent Technologies
E1470A
Cascade RF Switch Module
User's Manual

Agilent Technologies

Manual Part Number: E1470-90002
Printed in U.S.A. E1100

AGILENT TECHNOLOGIES WARRANTY STATEMENT

AGILENT PRODUCT: E1470A Cascade RF Switch Module **DURATION OF WARRANTY:** 3 years

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

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

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

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

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

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

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

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

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

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

U.S. Government Restricted Rights

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

Agilent Technologies

E1470A Cascade RF Switch Module User's Manual
Edition 2

Documentation History

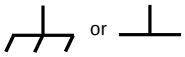
All Editions and Updates of this manual and their creation date are listed below. The first Edition of the manual is Edition 1. The Edition number increments by 1 whenever the manual is revised. Updates, which are issued between Editions, contain replacement pages to correct or add additional information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

Edition 1 . . . . . September, 1995
Edition 2 . . . . . November, 2000

Safety Symbols

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

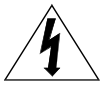
Indicates the field wiring terminal that must be connected to earth ground before operating the equipment — protects against electrical shock in case of fault.



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

Alternating current (AC)

Direct current (DC).



Warning. Risk of electrical shock.

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

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

WARNINGS

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

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

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

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

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

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

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

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

Agilent Technologies	**DECLARATION OF CONFORMITY** **According to ISO/IEC Guide 22 and CEN/CENELEC EN 45014**	

Manufacturer's Name: Agilent Technologies, Inc.
Manufacturer's Address: *Measurement Products Unit*
815 14th Street S.W.
Loveland, CO 80537 USA

Declares, that the product

Product Name: Cascade RF Switch
Model Number: E1470A
Product Options: *This declaration includes all options of the above product(s).*

Conforms with the following European Directives:

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

Conforms with the following product standards:

	Standard	**Limit**
EMC	*IEC 61326-1:1997 + A1:1998 / EN 61326-1:1997 + A1:1998*	
	CISPR 11:1997 + A1:1997 / EN 55011-1991	*Group 1, Class A [1]*
	IEC 61000-4-2:1995+A1998 / EN 61000-4-2:1995	*4 kV CD, 8 kV AD*
	IEC 61000-4-3:1995 / EN 61000-4-3:1995	*3 V/m, 80-1000 MHz*
	IEC 61000-4-4:1995 / EN 61000-4-4:1995	*0.5 kV signal lines, 1 kV power lines*
	IEC 61000-4-5:1995 / EN 61000-4-5:1995	*0.5 kV line-line, 1 kV line-ground*
	IEC 61000-4-6:1996 / EN 61000-4-6:1996	*3 V, 0.15-80 MHz*
	IEC 61000-4-11:1994 / EN 61000-4-11:1994	*1 cycle, 100%*
	Canada: ICES-001:1998	
	Australia/New Zealand: AS/NZS 2064.1	
Safety	*IEC 61010-1:1990+A1:1992+A2:1995 / EN 61010-1:1993+A2:1995*	
	Canada: CSA C22.2 No. 1010.1:1992	
	UL 3111-1	

Supplemental Information:

[1] The product was tested in a typical configuration with Agilent Technologies test systems.

September 5, 2000
Date

Name

Quality Manager
Title

For further information, please contact your local Agilent Technologies sales office, agent or distributor.
Authorized EU-representative: Agilent Technologies Deutschland GmbH, Herrenberger Straβe 130, D 71034 Böblingen, Germany

Revision: A.03 Issue Date: 09/05/00

Notes:

Chapter 1
Configuring the RF Switch

Using This Chapter

This chapter gives guidelines to use the Cascade RF Switch module (RF Switch) including:

Switching Diagram

The E1470A Cascade RF Switch module consists of a series of twenty 3-to-1 multiplexers. Each 3-to-1 multiplexer can be programatically cascaded with other 3-to-1 multiplexers to form larger multiplexers. For example, combining two adjacent multiplexers (cascading) forms a 6-to-1 multiplexer, cascading three forms a 9-to-1 multiplexer, or cascading four forms a 12-to-1 multiplexer, etc. Cascading all twenty 3-to-1 multiplexers forms one 60-to-1 multiplexer.

Multiple combinations are simultaneously allowed on the module. User connections to the module are to SMB connectors on the faceplate. Figure 1-2 shows the switching diagram of the Cascade RF Switch module with the switches shown in the power-on/reset state.

Since the relays on the switch are Form C, the relays are considered to be **reset** (or **opened**) when the COMMON terminal is connected to the NC terminal (the power-on/reset state). Relays are considered to be **set** (or **closed**) when the COMMON terminal is connected to the NO terminal. See Figure 1-1.

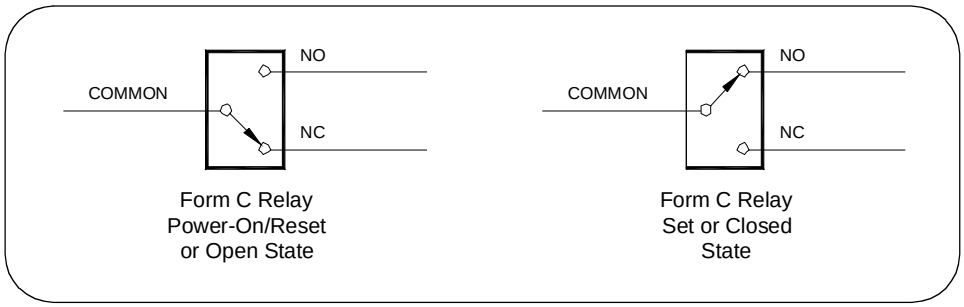


Figure 1-1. Form C Relays States

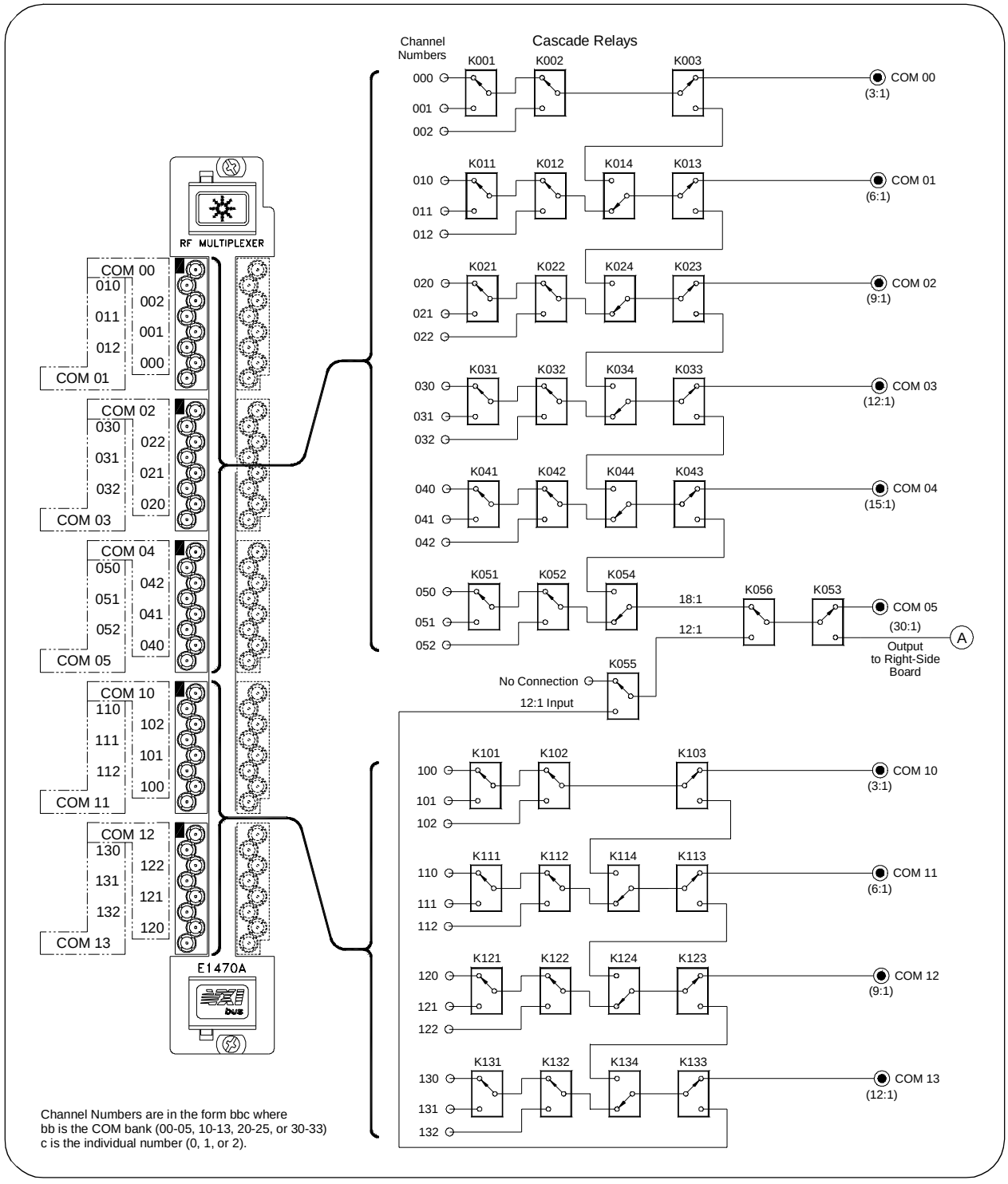


Figure 1-2. Cascade RF Switch Switching Diagram (continued on next page)

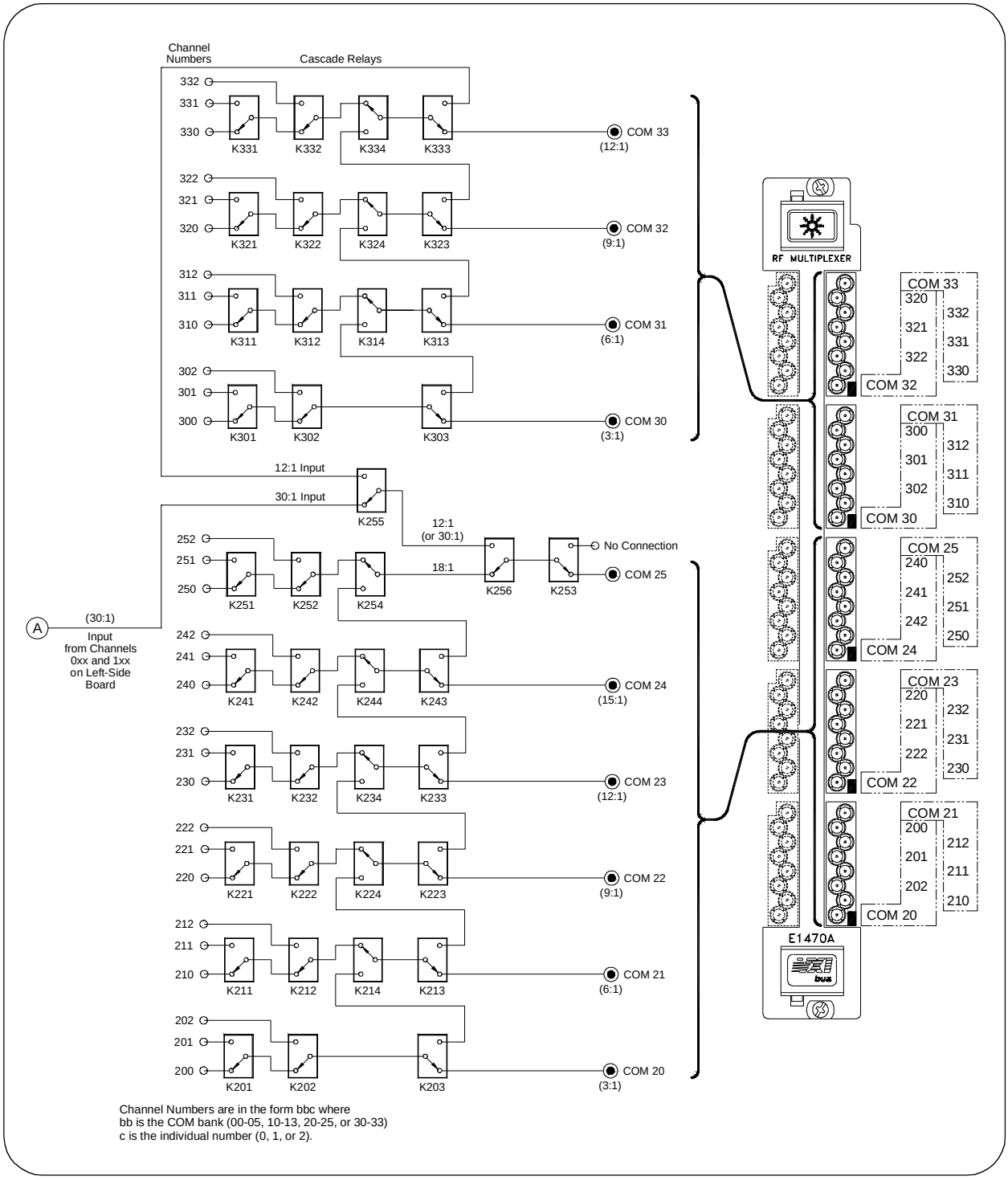


Figure 1-2. Cascade RF Switch Switching Diagram (continued)

Creating Multiple Multiplexers

You can configure the Cascade RF Switch module to create multiple multiplexers of varying sizes. In its power-on/reset state, the switch is configured as 20 independent 3-to-1 multiplexers. By specifying a valid path from a COM terminal to a channel in a different bank (functionally cascading contiguous 3-to-1 multiplexers) other multiplexer sizes can be configured.

Figure 1-3 shows typical 3-to-1, 6-to-1, 9-to-1, and 12-to-1 multiplexers. Other sizes can be configured by specifying valid ROUTe:PATH statements (see Chapter 2 for details). See Figure 1-2 for channel and COM numbering information.

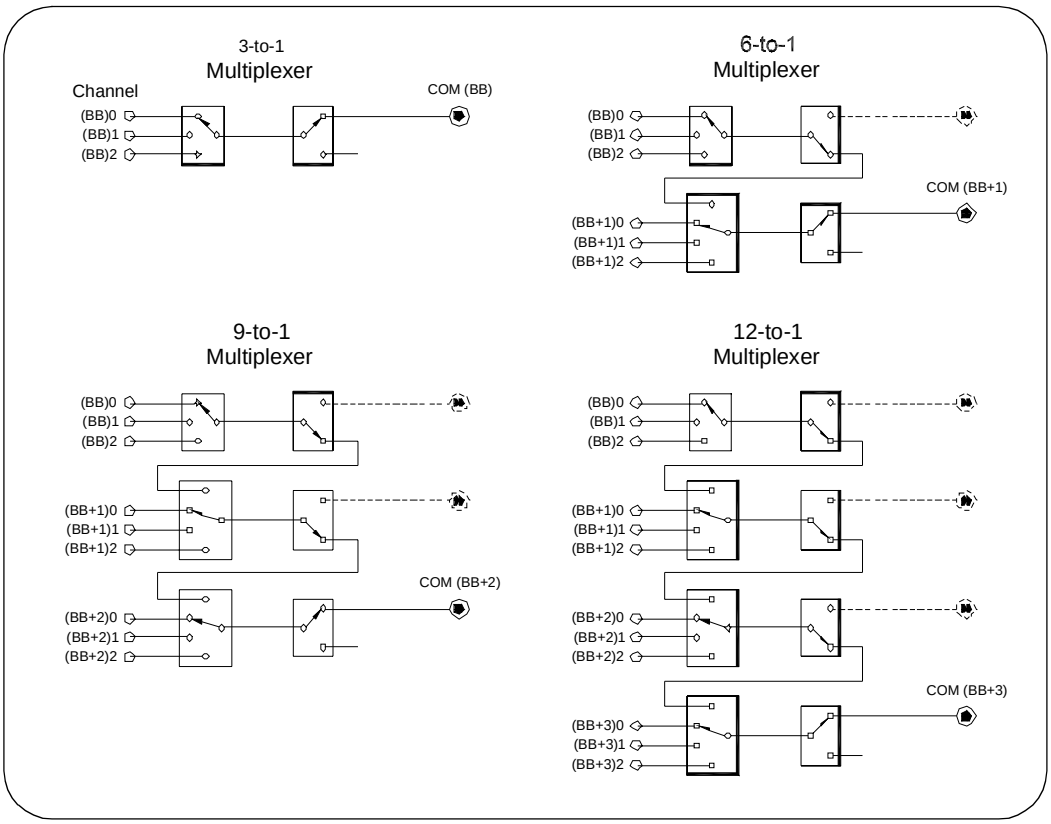


Figure 1-3. Creating Multiple Multiplexers

NOTE *Generally, the COM terminal is on the highest-numbered bank. Exceptions are that channels 100 through 132 can go to COM 05 as well as to COM 13 and channels 300 through 332 can go to COM 25 as well as to COM 33.*

For example, COM 01 can be used as the common for channels 000 - 002 and 010 - 012 creating a 6-to-1 multiplexer. COM 11 can be the common for channels 100 - 102 and 110 - 112 for another 6-to-1 multiplexer. COM 02 can be common for channels 000 - 002, 010 - 012, and 020 - 022 for a 9-to-1 multiplexer. COM 03 can be the common for channels 000 - 002, 010 - 012, 020 - 022, and 030 - 032 for a 12-to-1 multiplexer.

COM 04 can be used for a 15-to-1 multiplexer for all channels between 000 and 042. COM 05 can be the common for all channels from 000 through 052 creating an 18-to-1 multiplexer. Multiplexers of 21-to-1, 24-to-1, and 27-to-1 can also be configured. Two 30-to-1 multiplexers can be created using channels 00 through 132 to COM 05 and channels 200 through 332 to COM 25. One 60-to-1 multiplexer can be created using all the channels to COM 25.

RF Switch Configuration

This section gives guidelines to configure the RF Switch module, including:

- Warnings and Cautions
- Selecting the Logical Address
- Setting the Interrupt Request Level
- Connecting User Wiring

Warnings and Cautions

WARNING **SHOCK HAZARD. Only service-trained personnel who are aware of the hazards involved should install, remove, or configure the module. Before you remove any installed module, disconnect AC power from the mainframe and from other modules that may be connected to the module.**

WARNING **CHANNEL WIRING INSULATION. All channels that have a common connection must be insulated so that the user is protected from electrical shock in the event that two or more channels are connected together. This means wiring for all channels must be insulated as though each channel carries the voltage of the highest voltage channel.**

CAUTION **MAXIMUM POWER.** The maximum RF power that can be applied to the module is 10 Watts RF. Do not apply line AC power to any terminal on this module.

CAUTION **STATIC ELECTRICITY.** Static electricity is a major cause of component failure. To prevent damage to the electrical components in the module, observe anti-static techniques whenever removing a module from the mainframe or working on a module.

Setting the Logical Address

The logical address of the Cascade RF Switch module is set with the Logical Address (LADDR) switch on the module. The logical address is factory-set to 120. Valid addresses are from 1 to 256. See Figure 1-4 for address switch settings.

The logical address is the sum of the values of the switches set to the CLOSED position. In Figure 1-4, switches 3 through 6 are CLOSED and the associated values of these switches are 8, 16, 32, and 64. Thus, the logical address = 8 + 16 + 32 + 64 = 128.

NOTE *When using the Cascade RF Switch module with an E1406 Command Module, the address must be a multiple of 8 (for example, 8, 16, 24,... 112, 120, 128,... 240, 248). The module cannot be configured as part of a multiple-module switchbox instrument.*

If the Logical Address Switches are set for 255, the System Resource Manager automatically assigns a Logical Address to the module. You can poll the Resource Manager to determine the logical address assigned to the module.

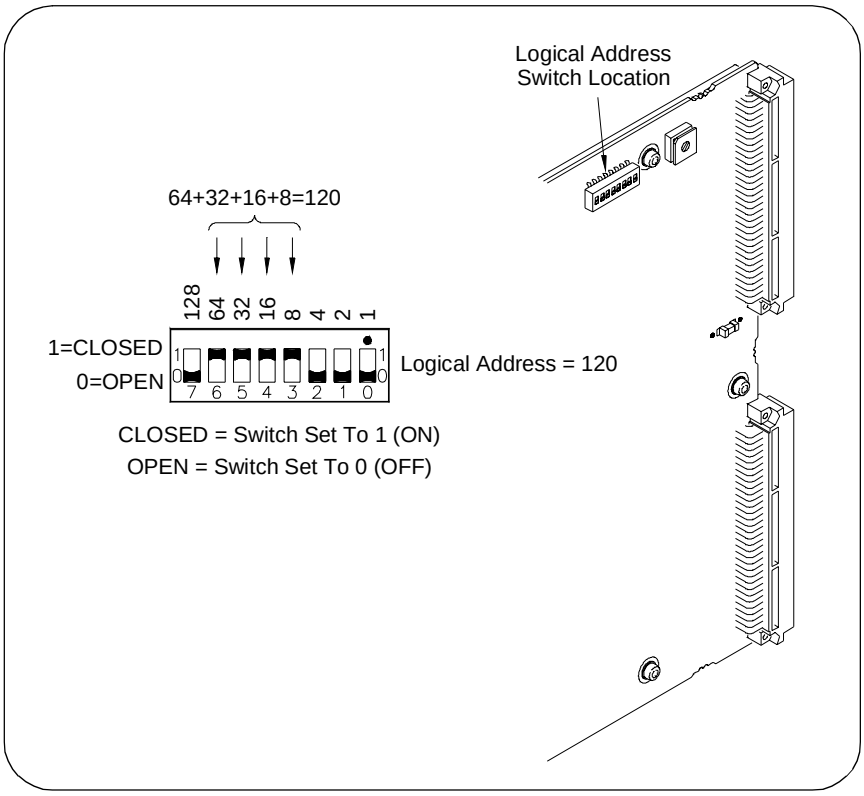


Figure 1-4. Setting the Logical Address Switch

Setting the Interrupt Request Level

Interrupts are enabled at power-up, after a SYSRESET, or after resetting the module via the Control Register (see Appendix B). If interrupts are enabled, the system generates an interrupt after writing to any relay control register. The interrupt is generated approximately 13 msec after writing to the register to indicate the end of relay closure/settling time.

As shown in Figure 1-5, the Interrupt Request Level switch selects the priority level that will be asserted. The Interrupt Request Level switch is set in position 1 as shipped from the factory. For most applications this priority level should not be changed. The interrupts are disabled when set to position 'X'. To change the setting, set the switch to the level required.

NOTE *Interrupts can also be disabled using the Control Register (see Appendix B). Also, consult your mainframe manual to make sure backplane jumpers/switches are configured correctly.*

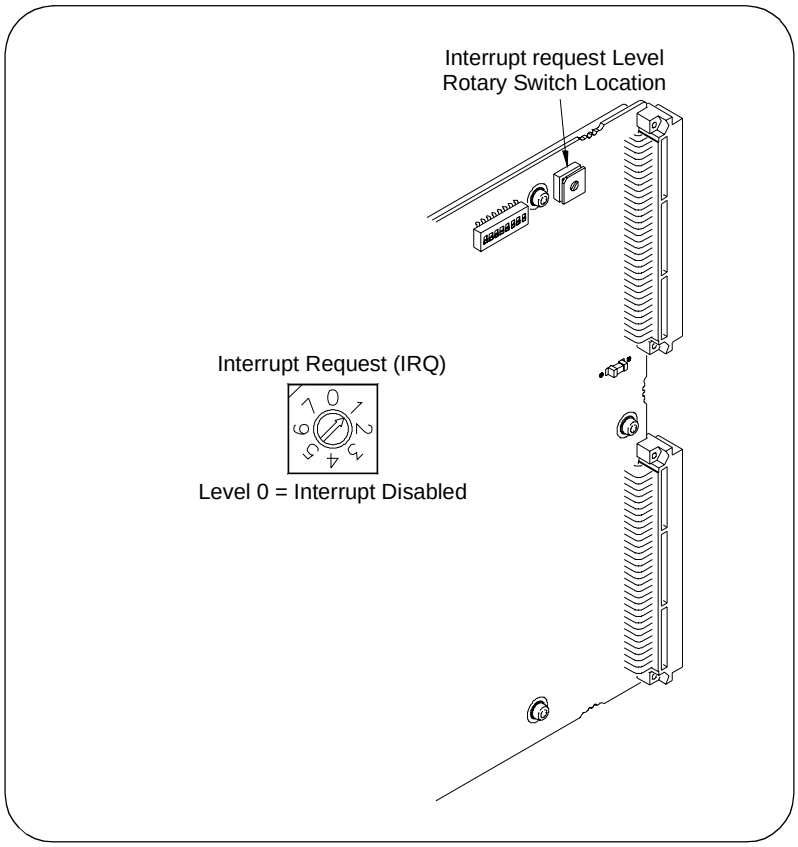


Figure 1-5. Setting the Interrupt Request Level Switch

Connecting User Wiring

User wiring connections to the module are via multiple connector blocks (part number 1250-2563). Figure 1-6 shows how to wire and assemble the connector housing. See “Cables and Connectors” for guidelines to assemble SMB jacks and connectors. See Table 1-2 in “User Wiring Log” for a log to record your wiring configuration.

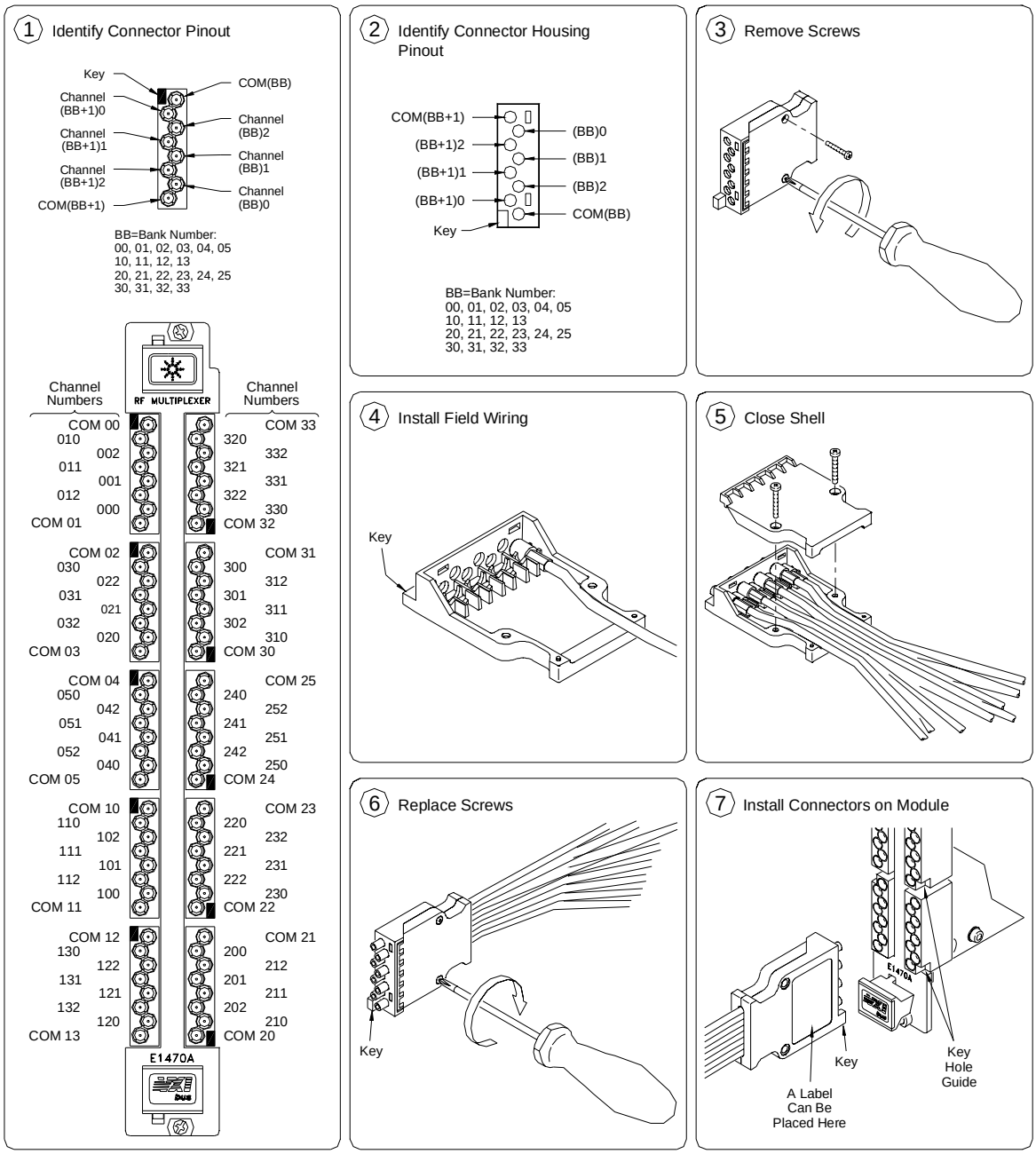


Figure 1-6. Installing User Wiring

Cables and Connectors

The Cascade RF Switch module is shipped with a kit of 85 SMB connector jacks and 10 connector housings. You must supply your own 50Ω double-shielded cable (single-shielded cable can also be used). Agilent recommends RG188DS or RS316DS double-shielded cables or triple-shielded cable (part number 8120-0552).

Standard SMB connector jacks will fit into the Cascade RF Switch module connector sockets and may be used if adjacent sockets on the module are NOT used. However, the outside diameter of the standard SMB jacks prohibits using them on the closely spaced, adjacent sockets on the module and they will not fit in the connector housing. Special jacks with a smaller shoulder must be used if adjacent sockets on the module are used. See Table 1-1.

Table 1-1. SMB Connectors and Connector Housings

Description	Quantity	Part Number
SMB Jacks*	Package of 8	E1470-22101
Connector Housing	Individual	1250-2563

* Single SMB jacks are available from E. F. Johnson Co. by part number 131-4304-011/020.

Assembling SMB Connector Jacks

Figure 1-7 shows how to assemble the SMB connector jacks. Jacks for double-shielded cable require a 0.151 hex crimp about 0.260 wide. Individual jacks for single-shielded RG188 and RG316 cable are available from E. F. Johnson Co (part number 131-4303-011/020) and require a hex crimp size of 0.128.

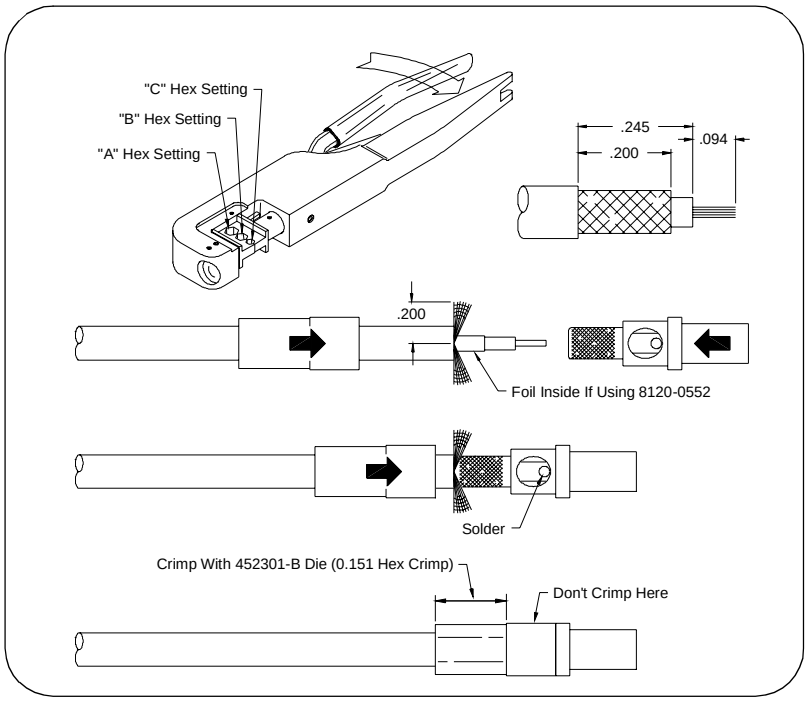


Figure 1-7. Assembling SMB Jacks and Cables

User Wiring Table

Table 1-2 provides a log for you to document wiring to the Cascade RF Switch module. See Figure 1-1 for terminal identification. See Figure 1-6 for guidelines to connect user wiring. You can copy the table as desired.

Table 1-2. User Connections Wiring

Term	Connected to:	Term	Connected to:	Term	Connected to:
COM 00		**COM 11**		**COM 24**	
CH 002		CH 110		CH 240	
CH 001		CH 111		CH 241	
CH 000		CH 112		CH 240	
COM 01		**COM 12**		**COM 25**	
CH 010		CH 122		CH 252	
CH 011		CH 121		CH 251	
CH 012		CH 120		CH 250	
COM 02		**COM 13**		**COM 30**	
CH 022		CH 130		CH 300	
CH 021		CH 131		CH 301	
CH 020		CH 132		CH 302	
COM 03		**COM 20**		**COM 31**	
CH 030		CH 200		CH 312	
CH 031		CH 201		CH 311	
CH 032		CH 202		CH 310	
COM 04		**COM 21**		**COM 32**	
CH 042		CH 212		CH 320	
CH 041		CH 211		CH 321	
CH 040		CH 210		CH 320	
COM 05		**COM 22**		**COM 33**	
CH 050		CH 220		CH 332	
CH 051		CH 221		CH 331	
CH 052		CH 222		CH 330	
COM 10		**COM 23**			
CH 102		CH 232			
CH 101		CH 231			
CH 100		CH 230			

Chapter 2
Programming the RF Switch

Using This Chapter

This chapter gives guidelines to program the Cascade RF Switch module (RF Switch) including:

Installing Device Drivers

Before you can use the Cascade RF Switch module, you may need to install device drivers. The type of driver(s) to be installed depend on whether you use an E1406 Command Module or another type of command module. The two types of drivers applicable to the RF Switch module are VXI*plug&play* Instrument Drivers (installed on your PC) and SCPI Instrument Drivers (downloaded into the E1406 Command Module).

NOTE *It is highly recommended the SCPI Instrument driver be installed whether the VXI instrument is programmed using its VXIplug&play driver or using SCPI commands embedded in an I/O language. For the latest information on drivers, see the Agilent Web Site:*

http://www.agilent.com/find/inst_drivers

To download the SCPI Instrument Driver into the E1406A Command Module, you will need to use the *VXI Installation Consultant (VIC)* contained on the *Agilent Technologies Universal Instrument Drivers* CD. To download the driver, install the CD in your CD-ROM drive and follow the installation instructions. The setup program should run automatically. If it does not, click **Start | Run** and type <*drive*>:SETUP.EXE in the command line, where <*drive*> is the letter for your CD-ROM drive.

NOTE *To download a driver, the ROM version number of the E1406 Command Module must be A.06.00 or above. To determine the version number, send the IEEE 488.2 common command *IDN?. A typical return value follows, where A.06.01 is the version number.*

HEWLETT-PACKARD,E1406A,0,A.06.01

Addressing the Switch

By specifying a path *destination* (a COM number) and a *source* (a channel number), a channel is connected to a COM terminal. The format for addressing the switch is [ROUTe:]PATH[:COMMon] <*comm*>, <*channel*> where <*comm*> is a 2-digit number specifying the bank for the COM terminal and <*channel*> is a 3-digit number specifying a channel number. (Leading 0s can be omitted.) See the [ROUTe:]PATH[:COMMon] command in Chapter 3 for valid <*comm*> and <*channel*> numbers.

You can use [ROUTe:]PATH[:COMMon]? <*comm*>, <*channel*> to indicate whether a path is closed (returns a 1) or is open (returns a 0). You can use the PATH statement to create multiple 3-to-1 multiplexers, 6-to-1 multiplexers, 9-to-1 multiplexers, 12-to-1 multiplexers, etc. Up to two 30-to-1 multiplexers or one 60-to-1 multiplexer can be configured. For example, the following statements each connect a COM terminal to a channel.

PATH COMM 00,001	*!Connects COM 00 to Channel 001, !COM 00 is common to channels 000, !001, 002; forming a 3-to-1 mux.*
PATH COMM 04,020	*!Connects COM 04 to Channel 020, !COM 04 is common to channels 020 !through 042; forming a 9-to-1 mux.*
PATH COMM 05,002	*!Connects COM 05 to Channel 002, !COM 05 is common to channels 000 !through 052 forming an 18-to-1 mux.*

Using invalid numbers for <*comm*> and <*channel*> will generate an error. When switching a signal path, only the relays necessary to complete the path are switched. All other relays remain in their current state. This prevents unexpected switching results. However, when closing one signal path, another signal path might open. For example:

PATH COMM 01,010	*!Closes a signal path from COM 01 !to Channel 010.*
PATH COMM? 01,010	*!Returns "1" indicating the path is !closed.*
PATH COMM 02,002	*!Closes a signal path from COM 02 to !Channel 002 and changes the state !of the cascade relay, opening the !prior signal path.*
PATH COMM? 01,010	*!Returns "0" indicating the path is !open.*

Programming Examples

The following C-language programs show one way to verify initial operation for the Cascade RF Switch module, to close signal paths, and to save and recall module states. To run these programs, you must have installed the E1470A SCPI Device Driver, *Agilent IO Libraries for Windows*, and a GPIB module in your PC.

Example: Module Self-Test

This program:

- Identifies the module and device driver
- Resets the module
- Closes a path (source/destination)
- Verifies that the path is closed
- Executes the module self-test

The *RST command performs a device reset on the module and sets it to its power-on state. (Saved module states and status information are not affected by *RST.) The *TST? command verifies that the relay positions match the configurations programmed using the ROUT:PATH commands.

NOTE **TST? results are unpredictable if you use register -based programming or DIAG:CLOS or DIAG:OPEN to control individual relays. The value returned should be a "0". Any other value indicates the actual state of the relays do not match the configuration programmed by the ROUT:PATH command. See Chapter 3 for details.*

```
/* Self -Test.
This program resets the E1470A, reads the ID string, performs a
self-test, reads any self-test error messages, and closes and verifies
a signal path */

#include <visa.h>
#include <stdio.h>
#include <stdlib.h>

void err_handler();
void main()
{

char buff[256] = {0};
int err_no, ch_closed;

   /* Create and Open a Device Session. E1470 is at logical address 120 */

   ViStatus err;
   ViSession defaultRM,rf_mux;
   viOpenDefaultRM (&defaultRM);
   viOpen (defaultRM,"GPIB-VXI0::9::120",VI_NULL,VI_NULL,&rf_mux);
```

```
    /*Reset the E1470A */

    err= viPrintf (rf_mux,"*RST;*CLS;*OPC?\n');
        if (err < VI SUCCESS) err_handler (rf_mux,err);
    err= viScanf (rf_mux,"%s",&buf);
        if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Read and display the ID String. Should return
       HEWLETT-PACKARD,E1470A,0,A.01.00 */

    err = viPrintf (rf_mux "*IDN?\n');
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err= viScanf (rf_mux,"%s",&buf);
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    printf ("Module Identification String: %s\n",buf);

    / * Do the Self Test */

    printf ("Performing the Self Test\n");
    err= viPrintf (rf_mux,"*TST?\n');
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err = viScanf (rf_mux,"%d",&err_no);
        while (err < VI_SUCCESS) err = viScanf (rf_mux,"%d",&err_no);
    if (err no != 0) printf ("\nSelf Test Error: %d\n",err_no);
        else printf ("\nNo Self Test Errors");

    /* Close a signal path from COM 02 to Channel 002 */

    err= ViPrintf (rf_mux,"PATH:COMM 02,002\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Verify the path is closed */

    err = viPrintf (rf_mux,"PATH: COMM? 02,002\n');
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err= viScanf (rf_mux,"%d",&ch_closed);
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    if (ch_closed ==1) printf ("Signal path is closed");
        else printf ("Signal path is NOT closed");

    /* Close Session */

    viClose (rf_mux);
    viClose (defaultRM);
}
void err_handler()     /* Error handling routine */
    {
    ViStatus err;
    char err_msg[1024]={0};
    viStatusDesc(rf_mux,err,err_msg);
    printf ("Error = %s\n",err_msg);
    return;
    }
```

Example: Closing a Signal Path

This program example closes a signal path from COM 01 to channel 010 and verifies that the path is closed.

```
#include <visa.h>
#include <stdio.h>
#include <stdlib.h>

void err_handler();
void main()
    {
    int ch_closed;

    /* Create and open a device session, E1470 is at logical address 120 */

    ViStatus err;
    ViSession defaultRM,rf_mux;
    ViOpenDefaultRM (&defaultRM);
    viOpen (defaultRM,"GPIB-VXI0::9::120",VI_NULL,VI_NULL,&rf_mux);

    /* Close a path from COM 01 to channel 010 */

    err= viPrintf (rf_mux,"PATH:COMM 01,011\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Verify the path closure */

    err = viPrintf (rf_mux,"PATH:COMM? 01,011\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err = viScanf (rf_mux,"%d",&ch_closed);
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    if (ch_closed == 1) printf ("Signal path is closed");
        else printf ("Signal path is NOT closed");

    /* Close the session */

    viClose (rf_mux);
    viClose (defaultRM);
}

void err_handler()     /* Error handling routine */
    {
    ViStatus err;
    char err_msg[1024]={0};
    viStatusDesc(rf_mux,err,err_msg);
    printf ("Error = %s\n",err_msg);
    return;
    }
```

Example: Opening and Closing Signal Paths

This program first closes a signal path from COM 01 to channel 011 and verifies that the path is closed. Next, the program closes a signal path from COM 02 to channel 010 (which opens the COM 01 to channel 011 path). Then, the program verifies that the COM 02 to channel 010 path is closed and the COM 01 to channel 011 path is open.

```
#include <visa.h>
#include <stdio.h>
#include <stdlib.h>

void err_handler();
void main()
    {
    int ch_closed;

    /* Create and open a device session. E1470 is at logical address 120 */

    ViStatus err;
    ViSession defaultRM,rf_mux;
    viOpenDefaultRM (&defaultRM);
    viOpen (defaultRM,"GPIB-VXI0::9::120",VI_NULL,VI_NULL,&rf_mux);

    /* Close a path from COM 01 to channel 011 */

    err= viPrintf (rf_mux,"PATH: COMM 01,011\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Verify path closure */

    err= viPrintf (rf_mux,"PATH:COMM? 01,011\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err= viScanf (rf_mux,"%d",&ch_closed);
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    if (ch_closed == 1) printf ("Signal path 01,011 is closed");
    else printf ("Signal path 01,011 is NOT closed");

    /* Close a second signal path COM02 to channel 010 */

    err = viPrintf (rf_mux,"PATH:COMM 02,01\n");
    if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Verify the path closure */

    err= viPrintf (rf_mux,"PATH:COMM? 01,011\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err= viScanf (rf_mux,"%d",&ch_closed);
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    if (ch_closed == 1) printf ("Signal path 01,011 is closed");
    else printf ("Signal path 01,011 is NOT closed");
```

```
    err= viPrintf (rf_mux,"PATH:COMM? 02,01\n");
        if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err = viScanf (rf_mux,"%d",&ch_closed);
    if (err < VI_SUCCESS) err_handler (rf_mux,err);
    if (ch_closed == 1) printf ("Signal path 02,010 is closed");
    else printf ("Signal path 02,010 is NOT closed");

    /* Close the session */

    viClose (rf_mux);
    viClose (defaultRM);
}
void err_handler()          /* Error handling routine */
    {
    ViStatus err;
    char err_msg[1024]={0};
    viStatusDesc(rf_mux,err,err_msg);
    printf ("Error = %s\n",err_msg);
    return;
    }
```

Example: Saving and Recalling Module States

The *SAV command saves the current state of all relays on the Cascade RF Switch module and thus all the signal path connections. You can use *SAV to save up to ten module states and then use the *RCL command to return to a specific saved state.

The commands have the form *SAV<*n*> and *RCL<*n*> where <*n*> has a range of 0 to 9. Error -222, “Data out of range” results if a value other than 0 through 9 is used for <*n*>.

This example program first creates several PATH configurations and saves that module state as state number 1. Next, the program creates additional paths (while the previous paths remain closed) and saves that state as state 2. Then, the program resets the module and recalls module state number 1.

```
#include <visa.h>
#include <stdio.h>
#include <stdlib.h>

void err_handler();
void main()
    {
    int ch_closed;

/* Create and open a device session. E1470 is at logical address 120 */

    ViStatus err;
    ViSession defaultRM,rf_mux;
    viOpenDefaultRM(&defaultRM);
    viOpen (defaultRM,"GPIB-VXI0::9::120",VI_NULL,VI_NULL,&rf_mux);
```

```
    /* Close multiple signal paths and save as state number 1 */

    err = viPrintf (rf_mux,"PATH:COMM 01,011;:PATH:COMM
          13,100;:PATH:COMM 31,301\n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);

    err = viPrintf (rf_mux,"*SAV 1 \n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Close additional signal paths and save as state number 2 */

    err= viPrintf (rf_mux,"PATH:COMM 02,010;:PATH:COMM 22,202;:
          PATH:COMM 24,232\n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);

    err = viPrintf (rf_mux,"*SAV 2\n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Reset the module */

    err = viPrintf (rf_mux,"*RST;*CLS:OPC?\n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err = viScanf (rf_mux,"%d",&ch_closed);
       if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Recall state number 1 */

    err= viPrintf (rf_mux,"*RCL 1\n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);

    /* Verify that a signal path from state number 1 is closed */

    err = viPrintf (rf_mux,"PATH:COMM? 01,011\n");
       if (err < VI_SUCCESS) err_handler (rf_mux,err);
    err = viScanf (rf_mux,"%d",&ch_closed);
       if (err < VI_SUCCESS) err_handler (rf_mux,err);
    if (ch_closed == 1) printf ("Signal path 01,011 is closed");
    else printf ("Signal path 01,011 is NOT closed");

    /* Close session */

    viClose (rf_mux);
    viClose (defaultRM);
{
void err_handler()          /* Error handling routine */
    {
    ViStatus err;
    char err_msg[1024]={0};
    viStatusDesc(rf_mux, err, err_msg);
    printf ("Error = %s\n",err_msg);
    return;
    }
```

Chapter 3
RF Switch Command Reference

This chapter describes Standard Commands for Programmable Instruments (SCPI) and summarizes IEEE 488.2 Common (*) Commands applicable to the E1470A Cascade RF Switch Module.

Command Types

Commands are separated into two types: IEEE 488.2 Common Commands and SCPI Commands.

Common Commands Format

The IEEE 488.2 standard defines the Common Commands that perform functions like reset, self-test, status byte query, etc. Common commands are four or five characters in length, always begin with the asterisk character (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of Common Commands are:

```
*RST    *ESR 32    *STB?
```

SCPI Commands Format

SCPI commands perform functions like closing switches, querying instrument states, or retrieving data. A subsystem command structure is a hierarchical structure that usually consists of a top level (or root) command, one or more lower level commands, and their parameters. The following example shows part of a typical subsystem:

```
[ROUTe:]PATH[:COMMon] <comm>,<channel>
```

[ROUTe:] is the (optional) root command, PATH is the second level command, and [:COMMon] is a third level (optional) command. <*comm*>,<*channel*> are command parameters.

Command Separator

A colon (**:**) always separates one command from the next lower level command as shown below. Colons separate the root command from the second level command (ROUTe:PATH) and the second level from the third level (PATH:COMMon).

```
ROUTe:PATH:COMMon
```

Abbreviated Commands

The command syntax shows most commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, you may send the entire command. The instrument will accept either the abbreviated form or the entire command.

For example, if the command syntax shows MEASure, then MEAS and MEASURE are both acceptable forms. Other forms of MEASure, such as MEASU or MEASUR will generate an error. You may use upper or lower case letters. Therefore, MEASURE, measure, and MeAsUrE are all acceptable.

Implied Commands

Implied commands are those which appear in square brackets ([ ]) in the command syntax. (The brackets are not part of the command and are not sent to the instrument.) Suppose you send a second level command but do not send the preceding implied command. In this case, the instrument assumes you intend to use the implied command and it responds as if you had sent it. Examine the [ROUTe:] subsystem shown below:

```
[ROUTe:]
    PATH[:COMMon] <comm>,<channel>
    PATH[:COMMon]? <comm>,<channel>
```

The root command ROUTe: is an implied command as is the command: COMMon. To close a signal path, you can send any of the following command statements:

```
PATH 2,1
ROUT:PATH 2,1
PATH:COMM 2,1
ROUT:PATH:COMM 2,1
```

These commands function the same, connecting the COMMON in bank 02 to channel 1 in bank 00. For information on channel and bank numbers, see Chapter 2.

Parameters

Parameter Types. The ROUTe:PATH command accepts only numeric parameters.

Linking Commands

Linking IEEE 488.2 Common Commands with SCPI Commands. Use a semicolon between the commands. For example RST; ROUT:PATH 2 *or* ROUT:PATH 2,1 ;*SAV 1

Linking Multiple SCPI Commands. Use both a semicolon and a colon between the commands. For example, ROUT:PATH 2,1;:PATH 3,32

SCPI Command Reference

This section describes the Standard Commands for Programmable Instruments (SCPI) commands for the E1470A Cascade RF Switch module. Commands are listed alphabetically in by subsystem and within each subsystem.

DIAGnostic

The DIAGnostic subsystem contains instrument-specific commands is are not recommended for general programming. For the E1470A, the DIAG subsystem allows you to open/close individual relays and query individual relays.

Subsystem Syntax

DIAGnostic
:CLOSe <*relay*>{,<*relay*> ...}
:CLOSe? <*relay*>{,<*relay*>...}
:OPEN <*relay*>{,<*relay*>...}
:OPEN? <*relay*>{,<*relay*>...}
:RELAY?

DIAGnostic:CLOSe

DIAGnostic:CLOSe <*relay*>{,<*relay*>...} closes individual relays on the E1470A. Since these are Form C relays, “closed” means the relay is “set” (COMMON to NO).

Parameters

Name	Type	Range of Values
<*relay*>	numeric	001-003|011-014|021-024|031-034|041-044| 051-056|101-103|111-114|121-124|131-134| 201-203|211-214|221-224|231-234|241-244| 251-256|301-303|311-314|321-324|331-334

Comments

Invalid Values. Values other than those listed in the table cause error 2022, “Invalid relay number”.

Closing Relays. To close single relays, use DIAG:CLOS *abc*. To close multiple relays, use DIAG:CLOS *abc,def,ghi,...* etc.

80 Relays Maximum. The E1470A has 80 relays. Setting more than 80 relay numbers causes error: -108, “Parameter not allowed”.

Example

Closing Relays

```
DIAG:CLOS 001          !Closes relay 001 (connects
                       !CH001 to relay 002 in bank 00)
```

DIAGnostic:CLOSe?

DIAGnostic:CLOSe <*relay*>{,<*relay*>...} returns a number to indicate the closed state of each relay in the list. Since these are Form C relays, “closed” means the relay is “set” (COMMON to NO).

Parameters

Name	Type	Range of Values
<*relay*>	numeric	001-003|011-014|021-024|031-034|041-044| 051-056|101-103|111-114|121-124|131-134| 201-203|211-214|221-224|231-234|241-244| 251-256|301-303|311-314|321-324|331-334

Comments

Relay Closure Results. The output buffer contains an unquoted string containing the result for the relay(s): 0 = Not closed (COMMON to NC) and 1 = Closed (COMMON to NO)

Invalid Values. Values other than those listed in the table cause error 2022, “Invalid relay number”.

Querying Relays. To query single relays, use DIAG:CLOS *abc*. To query multiple relays, use DIAG:CLOS? *abc,def,ghi*,... etc.

80 Relays Maximum. The E1470A has only 80 relays. Setting more than 80 relay numbers causes error: -108, “Parameter not allowed”.

Example

Querying Relay Closures

```
*RST                          !Reset module and open all relays
DIAG:CLOS 002                 !Closes relay 002 (connects
                              !CH002 to relay 003 in bank 00)
DIAG:CLOS? 001,002,003        !Returns 0,1,0
```

DIAGnostic:OPEN

DIAGnostic:OPEN <*relay*>{,<*relay*>...} opens individual relays on the E1470A. Since these are Form C relays, “open” means the relay is “reset” to its power-on state (COMMON to NC).

Parameters

Name	Type	Range of Values
<*relay*>	numeric	001-003|011-014|021-024|031-034|041-044| 051-056|101-103|111-114|121-124|131-134| 201-203|211-214|221-224|231-234|241-244| 251-256|301-303|311-314|321-324|331-334

Comments

Invalid Values. Values other than those listed in the table cause error: 2022, “Invalid relay number”.

Opening Relays. To open single relays, use DIAG:OPEN *abc*. To open multiple relays, use DIAG:OPEN *abc,def,ghi,...* etc.

80 Relays Maximum. The E1470A has only 80 relays. Setting more than 80 relay numbers causes error: -108, “Parameter Not Allowed”.

Example

Opening Relays

```
DIAG:OPEN 333                  !Opens relay 333 (connects
                               !COM333 to relay 334 in bank 33)
```

DIAGnostic:OPEN?

DIAGnostic:OPEN? *<relay>*{,*<relay>*...} returns a number to indicate the open state of each relay in the list. Since these are Form-C relays, “open” means that the relay is “reset” to its power-on state (Common to NC).

Parameters

Name	Type	Range of Values
<relay>	numeric	001-003|011-014|021-024|031-034|041-044| 051-056|101-103|111-114|121-124|131-134| 201-203|211-214|221-224|231-234|241-244| 251-256|301-303|311-314|321-324|331-334

Comments

Relay Open Results. The output buffer contains an unquoted string containing the result for the relay(s): 0 = Not Opened (COMMON to NO) and 1 = Opened (COMMON to NC).

Invalid Values. Values other than those listed in the table cause error: 2022, “Invalid relay number”.

Querying Relays. To query single relays, use DIAG:OPEN? *abc*. To query multiple relays, use DIAG:OPEN? *abc,def,ghi*,... etc.

80 Relays Maximum. The E1470A has only 80 relays. Setting more than 80 relay numbers causes error -108, “Parameter not allowed”.

Example

Querying Relays Opened

```
*RST                           !Reset module and open all relays
DIAG:CLOS 003,014              !Closes relays 003 and 014
                               !(connects relay 002 to relay 013)
DIAG:OPEN? 001, 002, 003, 014  !Returns 1,1,0,0
```

DIAGnostic:RELAY?

DIAGnostic:RELAY? returns the relay numbers of all relays that are closed. Closed is the SET position (COMMON to NO) and is the opposite state of the power-on/reset relay state. The command can be used to determine which relays are closed by a given PATH command.

Comments

Output Buffer Strings. The output buffer contains an unquoted, comma-separated string of numbers where each number is a relay number. If no relay is closed, the output buffer will contain the null string. This is a register readback command that returns the current state of the registers controlling the relays. It does not account for failed relays.

***RST condition.** At power-on or reset (*RST), DIAG:REL? will not return any channel numbers.

Example

Returning Closed Relay Numbers

```
*RST                              !Reset the module
DIAG:CLOS 042,043,053,054,256     !Completes a path from COM25
                                  !to channel 42.This is equivalent
                                  !to PATH 25,42
DIAG:REL?                         !Query the relays
```

This program returns:

```
042,043,053,054,256
```

[ROUTe:]

The ROUTe subsystem automatically connects a specified channel to a specified COMMon terminal on the module.

Subsystem Syntax

[ROUTe:]
PATH[:COMMon] <*comm*>,<*channel*>
PATH[:COMMon]? <*comm*>,<*channel*>

[ROUTe:]PATH[:COMMon]

[ROUTe:]PATH[:COMMon]<*comm*>,<*channel*> closes the E1470A path specified by <*comm*> and <*channel*>. <*comm*> is a 2-digit number and <*channel*> is a 3-digit number. Leading zeros may be omitted.

Parameters

Name	Type	Range of Values
<*comm*>	numeric	00-05, 10-13, 20-25, 30-33
<*channel*>	numeric	000-002, 010-012, 020-022, 030-032, 040-042, 050-052, 100-102, 110-112, 120-122, 130-132, 200-202, 210-212, 220-222, 230-232, 240-242, 250-252, 300-302, 310-312, 320-322, 330-332

Comments

Addressing Signal Paths. A signal path connects a <*channel*> terminal to a COM terminal (specified by <*comm*>. PATH <*comm*>,<*channel*> closes a single path. For multiple paths, use multiple linked commands: PATH <*comm*>,<*channel*>;:PATH <*comm*>,<*channel*>; etc.

Closing may Open Other Paths. Closing one path may open another path if both paths use the same relays. See Chapter 1 to determine if this might happen. Use [ROUTe:]PATH? to determine if a path is closed.

Invalid Values. Invalid <*comm*> and <*channel*> values or combinations may cause one of the following errors:

2001, “Invalid channel number” for invalid <*channel*>
2023, “Invalid common bank number” for invalid <*comm*>.
2024, “Invalid source bank number” for invalid <*channel*>
2025, “Invalid common-source combination” for invalid combination of <*comm*> and <*channel*> parameters.

***RST Condition.** Channel *bb*0 connects to COM *bb* for all 3-to-1 multiplexer banks. This is equivalent to PATH *bb,bb*0 (where *bb* is the <*comm*> number).

Example

Closing Channel Path

```
PATH 2,1                    !Connects COMMON in Bank 02
                            !to channel 1 in bank 00
```

[ROUTe:]PATH[:COMMon]?

[ROUTe:]PATH[:COMMon]?<*comm*>,<*channel*> returns either a 1 or a 0 indicating whether the specified path is closed (continuity exists) or open (the signal path is broken). <*comm*> is a 2-digit number and <*channel*> is a 3-digit number.

Parameters

Name	Type	Range of Values
<*comm*>	numeric	00-05, 10-13, 20-25, 30-33
<*channel*>	numeric	000-002, 010-012, 020-022, 030-032, 040-042, 050-052, 100-102, 110-112, 120-122, 130-132, 200-202, 210-212, 220-222, 230-232, 240-242, 250-252, 300-302, 310-312, 320-322, 330-332

Comments

Continuity Results. The output buffer contains an unquoted string signifying the result: 0 = the specified path does NOT have continuity or 1 = the specified path DOES have continuity

Command is Hardware Readback. PATH? is a hardware readback command. It returns the current state of the hardware controlling the specified path. PATH? does not account for a failed relay.

NOTE *Use PATH? to determine if a path is closed. Closing one path may open another path if both paths use the same relays. See Chapter 1 to determine if this might happen.*

Invalid Values. Invalid <*comm*> and <*channel*> values or combinations may cause one of the following errors:

2001, “Invalid Channel Number” for invalid <*channel*>
2023, “Invalid Common Bank Number” for invalid <*comm*>.
2024, “Invalid Source Bank Number” for invalid <*channel*>
2025, “Invalid common-source combination” for invalid combination of <*comm*> and <*channel*> parameters.

***RST Condition**. Channel *bb*0 connects to COM *bb* for all 3-to-1 multiplexer banks. This is equivalent to PATH *bb,bb*0 (where *bb* is the <*comm*> number).

Example Querying Paths Opened/Closed

```
PATH 2,1                    !Connects COMMON in Bank 02
                            !to channel 1 in bank 00
PATH? 2,1                   !Returns 1
PATH? 0,002                 !Returns 0
```

SYSTem

The SYSTem subsystem returns error numbers and error messages in the error queue of a module and the SCPI compliance year (version).

Subsystem Syntax

```
SYSTem
    :ERRor?
    :VERsion?
```

SYSTem:ERRor?

SYSTem:ERRor? returns the error numbers and corresponding error messages in the error queue. See Appendix C for a listing of the applicable error numbers and messages.

Comments

Error Numbers/Messages in the Error Queue: Each error records an error number and corresponding error message in the error queue. Each error message can be up to 255 characters long but typically is much shorter.

Clearing the Error Queue: An error number/message is removed from the queue each time the SYSTem:ERRor? query command is sent. The errors are cleared first-in, first-out. When the queue is empty, each following SYSTem:ERRor? query command returns 0, “No error”. To clear all error numbers/messages in the queue, execute either the *CLS or *RST command.

Maximum Error Numbers/Messages in the Error Queue: The queue holds a maximum of 30 error numbers/messages for each module. If the queue overflows, the last error number/message in the queue is replaced by -350, “Too many errors”. The least recent error numbers/messages remain in the queue and the most recent are discarded.

Example

Reading the Error Queue

```
SYST:ERR?                    !Query the error queue
```

SYSTem:VERSion?

SYStem:VERSion? returns SCPI compliance version of E1470A driver.

Comments

Return Value. The return value is in the form: "YYYY.N"

Example

Returning SCPI Compliance Version

```
SYST:VERS?                   !Returns compliance version
```

IEEE 488.2 Common Commands Quick Reference

The following table lists the IEEE 488.2 Common (*) Commands accepted by the E1470A module driver. For more information on Common Commands, see the the ANSI/IEEE Standard 488.2-1987.

Command	Command Description
*CLS	Clears all status registers and clears the error queue.
*ESE<*register value*>	Enable Standard Event.
*ESE?	Enable Standard Event Query.
*ESR?	Standard Event Register Query.
*IDN?	Instrument ID Query; returns identification string of the module: HEWLETT-PACKARD,E1470A,B.01.00
*OPC	Causes E1470A to set bit 0 (Operation Complete Message) in the Standard Event Status Register when all pending operations are complete. This allows for synchronization between instrument and computer or between multiple instruments. For the E1470A, the only pending operation is the time delay (approximately 16 msec) provided to allow the relays to settle. If this command waits longer than about 60 msec, the error -240, “Hardware error” is generated.
*OPC?	Operation Complete Query. The E1470A places a “1” in the output buffer when all pending operations are complete. For the E1470A, the only pending operation is the time delay (~ 16 msec) provided to allow the relays to settle. If this command waits longer than about 60 msec, the error -240, “Hardware error” is generated.
*RCL<*numeric state*>	Recalls the instrument state saved by *SAV.
*RST	Resets the module to its power-on state; Channel 0 connects to COMmon for all banks. This is equivalent to PATH x0,x00 (where x is the bank number).
*SAV<*numeric state*>	Stores up to 10 module states.
*SRE<*register value*>	Service request enable, enables status register bits.
*SRE?	Service request enable query.
*STB?	Read status byte query.
*TST?	Executes an internal self-test. *TST? compares the actual relay positions (by reading the hardware) to the specified states (by reading the software state). If the self-test passes, a “0” is returned. If a discrepancy occurs, the number returned is the decimal weighted sum of the following errors: 1 Register 20_h fails self-test. See Appendix B. 2 Register 22_h fails self-test. See Appendix B. 4 Register 24_h fails self-test. See Appendix B. 8 Register 26_h fails self-test. See Appendix B. 16 Register 28_h fails self-test. See Appendix B. *TST? is only valid if the module was programmed using the SCPI [ROUTe:]PATH command. Register writes and the DIAG subsystem will invalidate the software state and generate a *TST? error.

Command	Command Description
*WAI	Wait to Complete. For the E1470A, the only pending operation is the time delay (approximately 16 msec) provided to allow the relays to settle. If this command waits longer than about 60 msec, the error -240, “Hardware error” is generated.

SCPI Commands Quick Reference

	Description
DIAGnostic:CLOSe DIAGnostic:CLOSe? DIAGnostic:OPEN DIAGnostic:OPEN? DIAGnostic:RELAY	Closes individual relays. Returns a number that indicates the closed state of each relay in the list. Opens individual relays. Returns a number that indicates the open state of each relay in the list. Returns the relay numbers for all relays that are closed.
[ROUTe:]PATH[:COMMon] [ROUTe:]PATH[:COMMon]?	Connect a path. Query if path connected.
SYSTem:ERRor? SYSTem:VERSion?	Returns error number/message in the Error Queue. Returns SCPI compliance year.

Appendix A
RF Switch Specifications

Configuration: 80 signal connections 60 inputs (channel numbers xx0 through xx2) 20 commons (channel numbers COMxx) One 60:1, two 30:1,... up to 20 3:1 multiplexers can be configured	**3dB Bandwidth:** 3:1 Multiplexer: 500 MHz 30:1 Multiplexer: 200 MHz (30:1 specifications apply for channels 000 - 132 to COM05 or channels 200 - 332 to COM25 multiplexers) 60:1 Multiplexer: 100 MHz
Terminated Isolation: 10 MHz: 80 dB 100 MHz: 60 dB 200 MHz: 50 dB 500 MHz: 40 dB	**VSWR (Voltage Standing Wave Ratio) for a 3:1 Multiplexer:** 100 MHz: 1.4 200 MHz: 1.45 500 MHz: 1.7
VSWR (Voltage Standing Wave Ratio) for a 30:1 Multiplexer: 200 MHz: 1.5 (30:1 muxs are channels 000 - 132 to COM05 or channels 200 - 332 to COM25 multiplexers)	**VSWR (Voltage Standing Wave Ratio) for a 60:1 Multiplexer:** 100 MHz: 1.5
Characteristic Impedance: 50Ω	**Power Consumption:** +5 Volt power supply, 3.5A (all relays closed)
Relay Ratings: Maximum RF Power: 10W Switch Life: no load: $5x10^6$ closures 0.01A @ 24 Vdc: $3x10^5$ closures 10 Watts RF: $1x10^5$ closures	**A16 Register-based device:** A16 Register-based Slave-only device as defined in the VMEbus standard and the VXIbus specification, rev. 1.4.
Power-on/Reset State: Channel xx0 connected to COMxx. Channel 0 in each 3:1 is connected to its respective common.	

Notes:

Appendix B
Register-Based Programming

About This Appendix

This appendix contains the information you can use for register-based programming of the E1470A Cascade RF Switch module. The contents include:

Register Addressing

The E1470A Cascade RF Switch is a register-based module that does not support the VXIbus word serial protocol. When a SCPI command is sent to the module, the instrument driver resident in the command module parses the command and programs the module at the register level.

Addressing Overview

Register-based programming is a series of *reads* and *writes* directly to the multiplexer registers. This can increase throughput speed since it eliminates command parsing and allows the use of an embedded controller. It also allows use of an alternate VXI controller, eliminating the command module.

To access a specific register for either read or write operations, the address of the register must be used. Register addresses for the plug-in modules are in an address space known as VXI A16. The exact location of A16 within a VXIbus master's memory map depends on the design of the VXIbus master you are using. For the E1406 Command Module, the A16 space location starts at $1F0000_h$.

The A16 space is further divided so that the modules are addressed only at locations above $1FC000_h$ within A16. Every module is allocated 64 register addresses (40_h). The address of a module is determined by its logical address (set by the address switches on the module) times 64 (40_h).
In the case of the Cascade RF Switch module, the factory setting is 120 or 78_h, so the addresses start at $1E00_h$.

Register addresses for register-based devices are located in the upper 25% of VXI A16 address space. Every VXI device (up to 256) is allocated a 64 byte block of addresses. Figure B-1 shows the register address location within A16. Figure B-2 shows the location of A16 address space in the E1406 Command Module.

The Base Address

When you are reading or writing to a module register, a hexadecimal or decimal register address is specified. This address consists of a base address plus a register offset. The base address used in register-based programming depends on whether the A16 address space is outside or inside the E1406 Command Module.

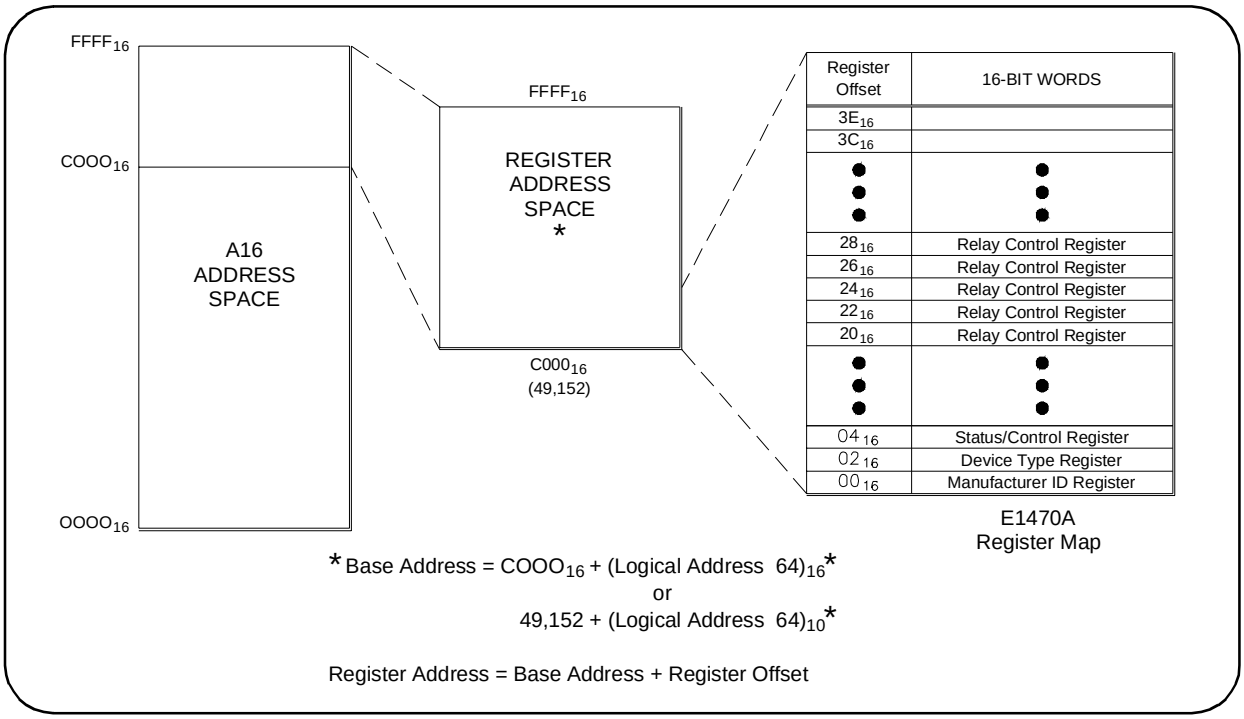


Figure B-1. Register Address Locations Within VXI A16

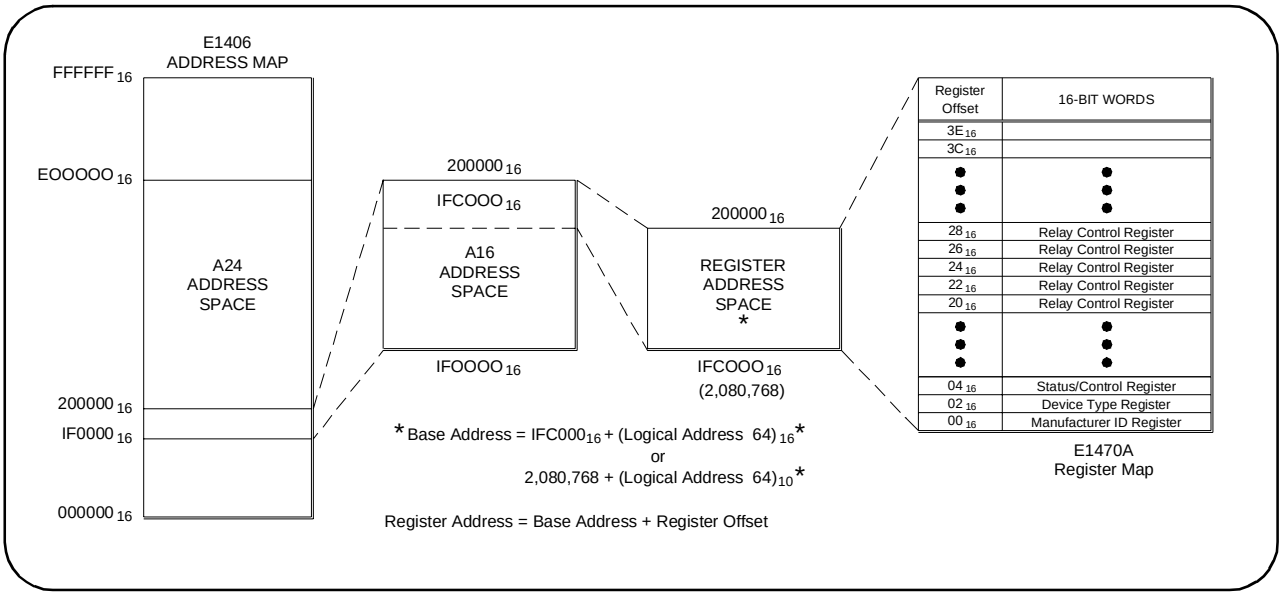


Figure B-2. A16 Address Space in the E1406 Command Module

A16 Address Space Outside the Command Module

When the E1406 Command Module is not part of your VXIbus system, the E1470 base address is computed as:

$A16_{base} = C000_h + (LADDR_h * 40_h)$

or (decimal)

$A16_{base} = 49{,}152 + (LADDR * 64)$

where $C000_h$ (49,152) is the starting location of the register addresses, LADDR is the module's logical address, and 64 is the number of address bytes per VXI device.

For example, the E1470 factory-set logical address is 120 (78_h). Therefore, it will have a base address of:

$A16_{base} = C000_h + (78_h * 40_h) = C000_h + 1E00_h =$ **$DE00_h$**

or (decimal)

$A16_{base} = 49{,}152 + (120 * 64) = 49{,}152 + 7680 =$ **56,832**

A16 Address Space Inside the Command Module or Mainframe

When the A16 address space is inside the E1406 Command Module, the E1470 base address is computed as:

$1FC000_h + (LADDR_h * 40_h)$

or (decimal)

2,080,768 + (LADDR * 64)

where $1FC000_h$ (2,080,768) is the starting location of the VXI A16 addresses, LADDR is the module's logical address, and 64 is the number of address bytes per register-based device. The E1470 factory-set logical address is 120. If this address is not changed, the module will have a base address of:

$1FC000_h + (78_h * 40_h) = 1FC000_h + 1E00_h =$ **$1FDE00_h$**

or (decimal)

2,080,768 + (120 * 64) = 2,080,768 + 7680 = **2,088,448**

Register Offset

The register offset is the register's location in the block of 64 address bytes that belong to the module. For example, the module's Status/Control Register has an offset of 04_h. When you write a command to this register, the offset is added to the base address to form the register address:

$DE00_h + 04_h = DE04_h$ $1FDE00_h + 04_h =$ **$1FDE04_h$**
or (decimal)
56,832 + 4 = 56,836 2,088,488 + 4 = **2,088,492**

Table B-1 shows general programming method to access E1470 registers.

Table B-1. General Register-based Programming Method

System	Typical Commands	Base Address
External Computer (over GPIB to E1406 Command Module)	VXI:READ? *logical_address, offset* VXI:WRITE *logical_address,offset,data* DIAG:PEEK? (*Base_addr + offset,width*) DIAG:POKE (*Base_addr +offset,width, data*) When using DIAG:PEEK? and DIAG:POKE, the width must be either 8 or 16.	Module Logical Address setting (LADDR*) offset = register number Base_addr = $1FC000_h$ + (LADDR * $40)_h$ or = 2,080,768 + (LADDR * 64) offset = register number

* LADDR = E1470 Logical Address = 120 / 8 = 15

Reset and Registers

When the E1470A undergoes power-on or a *RST in SCPI, the bits of the registers are put into the following states. Manufacturer ID Register, Device Type Register, and Status/Control Register are unaffected and Relay Control Registers have a “0” written to each bit. This forces all relays to their power-on/reset state. To reset the module, write a “1” and then a “0” to bit 0 of the Status/Control Register.

Register Definitions

You can program the E1470A Cascade RF Switch module using its hardware registers. The procedures for reading or writing to a register depend on your operating system and programming language. Whatever the access method, you will need to identify each register with its address. These addresses are given in Table B-2.

Table B-2. Register Map

Register Name	Address
Manufacturer ID (read only register)	Base + 00_h
Device ID (read only register)	Base + 02_h
Card /Status/Control (read/write register)	Base + 04_h
Relay Control Register (read/write register)	Base + 20_h
Relay Control Register (read/write register)	Base + 22_h
Relay Control Register (read/write register)	Base + 24_h
Relay Control Register (read/write register)	Base + 26_h
Relay Control Register (read/write register)	Base + 28_h

The interrupt protocol supported is “release on interrupt acknowledge.” An interrupt is cleared by a VXIbus interrupt acknowledge cycle.

CAUTION Registers have been documented as 8-bit bytes. If you access them using 16-bit transfers from a Motorola CPU, the high and low byte will be swapped. The E1406 uses Motorola CPUs. Motorola CPUs place the highest weighted byte in the lower memory location and the lower weighted byte in the higher memory address while Intel processors do just the opposite. VXI registers are memory-mapped. Thus, you will see this Motorola/Intel byte swap difference when doing register programming.

Manufacturer Identification Register

The Manufacturer Identification Register is a read-only register at address 00_h (Most Significant Byte (MSB)) and 01_h (Least Significant Byte (LSB)). Reading this register returns the Hewlett-Packard identification, $FFFF_h$.

Device Identification Register

The Device Identification Register is a read-only register accessed at address 02_h. Reading this register returns the module identification of 581 (245_h).

Status/Control Register

The Card Status/Control Register is a read/write register accessed at address 04_h. You read the Status Register and write to the Control Register.

Table B-3. Status Register Bit Patterns (read)

Address b+04$_h$								Address b+05$_h$							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	CDI0	CDI1	BSY	IEN	1	1	1	1	1	SR

SR (soft Reset): 0 = not in reset, 1 = held in reset state.
IEN: Main interrupt enable. Bit is set to 0 when interrupts are enabled; 1when interrupts are disabled.
BSY: Bit is set to 0 when module is busy - relays are settling. Bit is set to 1 if the module is not busy.
CDI0 and CDI1: When set to 0, indicates the relay assemblies are connected to the driver assembly. CDI0 is the right hand relay assembly, CDI1 is the left hand assembly. If either bit is set to a 1, the respective relay assembly is not installed.

Table B-4. Control Register Bit Pattern (write)

Address b+04$_h$								Address b+05$_h$							
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	1	1	1	1	1	1	1	IEN	1	1	1	1	1	SR

SR (soft reset): Writing a “1” and then a “0” to this bit resets all relays on the module to their power-on/reset state.
IEN: Main interrupt enable. Writing a 1 to this bit causes an interrupt to be generated 16 msec after a value is written to any relay control register to indicate that a relay closure should be complete. At power-on/reset, this bit is set to 0.

Relay Control Registers

These registers control the individual E1470A relays. When a “1” is written to a bit, the relay controlled by that bit becomes SET (COMMON to NO). When a “0” is written to a bit, the relay controlled by that bit becomes RESET (common to NC, the power-on state). All bits are “0” at power-on and reset. Reading a bit returns the state of that bit.

The left-hand relay assembly (when viewed from the front panel of the of the module) has relays K000 through K133. The right-hand relay assembly has relays K201 through K331.

Table B-5. Left-hand Relay Assembly Registers (b + 20h)

$b + 20_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Write	K034	K033	K032	K031	K024	K023	K022	K021	K014	K013	K012	K011	K041	K003	K002	K001
Read	K034	K033	K032	K031	K024	K023	K022	K021	K014	K013	K012	K011	K041	K003	K002	K001

Table B-6. Left-hand Relay Assembly Registers (b + 22h)

$b + 22_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Write	K134	K133	K132	K131	K124	K123	K122	K121	K114	K113	K112	K111	K051	K103	K102	K101
Read	K134	K133	K132	K131	K124	K123	K122	K121	K114	K113	K112	K111	K051	K103	K102	K101

Table B-7. Right-hand Relay Assembly Registers (b+24h)

$b + 24_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Write	K334	K333	K332	K331	K324	K323	K322	K321	K314	K313	K312	K311	K251	K203	K202	K201
Read	K334	K333	K332	K331	K324	K323	K322	K321	K314	K313	K312	K311	K251	K203	K202	K201

Table B-8. Right-hand Relay Assembly Registers (b+26h

$b + 26_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Write	K234	K233	K232	K231	K224	K223	K222	K221	K214	K213	K212	K211	K241	K203	K202	K201
Read	K234	K233	K232	K231	K224	K223	K222	K221	K214	K213	K212	K211	K241	K203	K202	K201

Table B-9. Relays on BOTH Assemblies Register (b+28h)

$b + 28_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Write	K256	K255	K254	K253	K252	K244	K243	K242	k056	K055	K054	K053	K052	K044	K043	K042
Read	K256	K255	K254	K253	K252	K244	K243	K242	k056	K055	K054	K053	K052	K044	K043	K042

Writing to Relay Control Registers

To set one or more relays write a “1” to the bit controlling that relay:

1. Determine the register and bit locations for the relays you want to set.
2. Add the decimal values for each bit you want to set in a register.
3. Use the VXI:REG:WRITe command to write that decimal value to that register.

Examples: Writing to Relay Control Registers

In these examples, since you are writing 1s to specific bits, the process actually writes 0s to all other bits in that register thus resetting those relays. To maintain previously established signal paths, you should read the register state and “mask” those bits when writing to the register.

NOTE *If Bit 15 is a “1”, BASIC language programming uses a 2s compliment number so the decimal value is negative. For example, $FFFF_h$ = -1, 8000_h = -32768.*

If all relays are in their power-on/reset state, to set relay K002 (connect channel CH002 to COM 00), set bit 1 (decimal value 2) in register (base + 20_h). Use the commands:

```
VXI:SEL 120                    !Selects logical address
VXI:REG:WRIT 20,2              !Writes value 2 to register 20h
```

To set relays K001, K003, K014, K013, and K024 (connect CH 001 to COM 02) set bits 0, 2, 7, 6, and 11 (decimal values 1, 4, 128, 64, 2048 respectively), send the decimal value 2245 (1 + 4 + 128 + 64 + 2048 = 2245) to register 20_h. Use the commands:

```
VXI:SEL 120                    !Selects logical address
VXI:REG:WRIT 20,2245
```

Similarly, to reset a relay to its power-on/reset state, write a “0” to the respective bit.

Reading from Relay Control Registers

Use the VXI:REG:READ? command to read the value of a register. The value returned is the decimal-weighted sum of all the bits in that register that are set to “1” (relays in the “set” state). At power-on/reset, the value returned should be 0. Use the command:

```
VXI:SEL 120                    !Selects logical address
VXI:REG:READ? 20               !Reads from register base + 20h
```

Examples: Writing to Relay Control Registers

The following table shows examples of the decimal values needed to write to a register(s) to connect signal paths. Hundreds more combinations are possible. These tables only show representative samples. Negative values are 2s compliment.

Table B-10. Writing to Relay Control Registers

To connect CH000 to:	COM00	COM01	COM02	COM03	COM04	COM05	COM25
Write to register 20_h	0	132	2244	-29500	-13116	-13116	-13116
Write to register 28_h	x	x	x	x	4	38	-32714
To connect CH001 to:	**COM00**	**COM01**	**COM02**	**COM03**	**COM04**	**COM05**	**COM25**
Write to register 20_h	1	133	2245	-29499	-13115	-13115	-13115
Write to register 28_h	x	x	x	x	4	38	-32714
To connect CH002 to:	**COM00**	**COM01**	**COM02**	**COM03**	**COM04**	**COM05**	**COM25**
Write to register 20_h	2	134	2246	-29498	-13114	-13114	-13114
Write to register 28_h	x	x	x	x	4	38	-32714

To connect CH010 to:	COM01	COM02	COM03	COM04	COM05	COM25
Write to register 20_h	0	2112	-29632	-13248	-13248	-13248
Write to register 28_h	x	x	x	4	38	-32714
To connect CH011 to:	**COM01**	**COM02**	**COM03**	**COM04**	**COM05**	**COM25**
Write to register 20_h	16	2128	-29616	-13232	-13232	-13232
Write to register 28_h	x	x	x	4	38	-32714
To connect CH012 to:	**COM01**	**COM02**	**COM03**	**COM04**	**COM05**	**COM25**
Write to register 20_h	32	2144	-29600	-13216	-13216	-13216
Write to register 28_h	x	x	x	4	38	-32714

To connect CH030 to:	COM03	COM03	COM05	COM25
Write to register 20_h	0	16384	16384	16384
Write to register 28_h	x	4	38	-32714
To connect CH031 to:	**COM03**	**COM03**	**COM05**	**COM25**
Write to register 20_h	4096	20480	20480	20480
Write to register 28_h	x	4	38	-32714
To connect CH032 to:	**COM03**	**COM03**	**COM05**	**COM25**
Write to register 20_h	8192	24576	24576	24576
Write to register 28_h	x	4	38	-32714

Register Programming Example

This example program reads the ID and Device Type registers and then reads the Status register. Next, the program closes a signal path from channel CH031 to COM 05, writes the value 20480 (5000 hexadecimal) to register 20_h and then writes the value 38 (26 hexadecimal) to register 28_h. Then, the program resets the module to open all channels. A typical printout for the program is:

```
ID register = 0xFFFF
Device Type register = 0x 218
Status register = 0xFFBE
Left-hand Assembly Register 20h = 0x5000
Left-hand Assembly Register 22h = 0x      0
Right-hand Assembly Register 24h = 0x     0
Right-hand Assembly Register 26h = 0x     0
Register 28h for Both Assemblies = 0x    26
```

```
#include <visa.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

ViSession defaultRM,rf_mux;
void err_handler();
void wait();

void main(void)
{
    unsigned short reg_20h, reg_22h;  /* Registers for Left-hand assembly*/
    unsigned short reg_24h, reg_26h;  /* Registers for Right-hand assy*/
    unsigned short reg_28h;           /* Register for both assemblies */
    unsigned short id_reg, dt_reg;    /* ID and device type registers */
    unsigned short stat_reg;    /* status register */

    /* create and open a device session */

    ViStatus err;
    viOpenDefaultRM (&defaultRM);
    viOpen (defaultRM,"GPIB-VXI0::9::120",VI_NULL,VI_NULL,&rf_mux);

    /* reset the module */

    err = viOut16(rf_mux,VI_A16_SPACE,0x04,1);
        if(err < VI_SUCCESS)err_handler(rf_mux,err);

    /* wait 1 second (must wait at least 100 usec before writing a "0") */

    wait(1);

    err = viOut16(rf_mux,VI_A16_SPACE,0x04,0);
```

```
    if(err < VI_SUCCESS)err_handler(rf_mux,err);

/* read the ID and Device Type registers */

err = viIn16(rf_mux,VI_A16_SPACE,0x00,&id_reg);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
err = viIn16(rf_mux,VI_A16_SPACE,0x02,&dt_reg);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
printf("ID register = 0x%4X\nDevice Type register =
0x%4X\n",id_reg,dt_reg);

/* read the Status Register */

err = viIn16(rf_mux,VI_A16_SPACE,0x04,&stat_reg);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
printf("Status register = "0x%4X\n",stat_reg);

/* close relays on registers 20h & 28h for signal path */
/* from CH031 to COM05. 20840 decimal = 5000h and */
/* 38 decimal = 26h */

err = viOut16(rf_mux,VI_A16_SPACE,0x20,0x5000);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
err = viOut16(rf_mux,VI_A16_SPACE,0x28,0x26);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);

/* read relay control registers and print their values */

err = viIn16(rf_mux,VI_A16_SPACE,0x20,&reg_20h);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
err = viIn16(rf_mux,VI_A16_SPACE,0x22,&reg_22h);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
err = viIn16(rf_mux,VI_A16_SPACE,0x24,&reg_24h);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
err = viIn16(rf_mux,VI_A16_SPACE,0x26,&reg_26h);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
err = viIn16(rf_mux,VI_A16_SPACE,0x28,&reg_28h);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);

printf("\n\nLeft-hand Assembly Register 20h = 0x%4X\n",reg_20h);
printf("Left-hand Assembly Register 22h = 0x%4X\n", reg_22h);
printf("Right-hand Assembly Register 24h = 0x%4X\n", reg_24h);
printf("Right-hand Assembly Register 26h = 0x%4X\n", reg_26h);
printf("Register 28h for Both Assemblies = 0x%4X\n",reg_28h);

/* wait 5 seconds before resetting module */
wait (5);

/* reset the E1470A to open all closed channels */
/* writing a 0 to the relay control registers also opens channels */
err = viOut16(rf_mux,VI_A16_SPACE,0x04,1);
    if(err < VI_SUCCESS)err_handler(rf_mux,err);
```

```
    /* wait 1 second (must wait at least 100 usec before writing a "0") */

    wait(1);
    err = viOut16(rf_mux,VI_A16_SPACE,0x04,0);
        if(err < VI_SUCCESS)err_handler(rf_mux,err);
    printf("\n\nE1470A is reset");

    /* Close Session */

    viClose (rf_mux);
    viClose (defaultRM);

}

void err_handler()        /* Error Handling Routine */
    {
    ViStatus err;
    char err_msg[1024] = {0};
    viStatusDesc(rf_mux,err,err_msg);
    if(strcmp ("VI_SUCCESS: No error",err_msg) != 0)
    printf("ERROR  = %s\n",err_msg);
    return;
    }

void wait (int wait_seconds)   /* Wait for specified period in seconds */
    {
    time_t current_time;
    time_t entry_time;
    fflush(stdout);
    if(-1 == time(&entry_time))
        {
            printf ("Call failed, exiting ...\n");
            exit(1);
        }
    do
    {
        if (-1 == time)&current_time))
            {
                printf("Call failed, exiting ...\n");
                exit(1);
            }
        }
    while ((current_time - entry_time) ((time-t)wait_seconds));
    fflush(stdout);
    }
```

Notes:

Appendix C
RF Switch Error Messages

The following error messages are unique to the E1470A. See the appropriate command module or VXI Controller module manual for a more complete list of possible error messages.

Error Number	Message Generated	Description	Commands that may cause error
-108	"Parameter not allowed"	A parameter was specified that is not valid for the command. More than 80 channels specified in command.	DIAG:CLOS; DIAG:CLOS?; DIAG:OPEN; DIAG:OPEN?
-222	"Data out of Range"	Invalid numerical state parameter.	*RCL *SAV
-240	"Hardware Error"	More than 60 msec was required for the relays to settle.	*OPC *OPC? *WAI
2001	"Invalid channel number"	Invalid channel number in <*source*> parameter	[ROUT:]PATH[:COMM] [ROUT:]PATH[:COMM]?
2022	"Invalid relay number"	Invalid relay number in command	DIAG:CLOS; DIAG:CLOS?; DIAG:OPEN; DIAG:OPEN?
2023	"Invalid common bank number"	Invalid <*comm*> parameter in command.	[ROUT:]PATH[:COMM] [ROUT:]PATH[:COMM]?
2024	"Invalid source bank number"	Invalid source number in <*source*> parameter	[ROUT:]PATH[:COMM] [ROUT:]PATH[:COMM]?
2025	"Invalid Common-Source Combination"	Even though the <*comm*> and <*source*> parameters are valid, the combination is not valid. The specified source cannot connect to the specified COM terminal.	[ROUT:]PATH[:COMM] [ROUT:]PATH[:COMM]?

Index
E1470A Cascade RF Switch User's Manual

Agilent Technologies

Manual Part Number: E1470-90002
Printed in U.S.A. E1100