Agilent Technologies E8480A High Power General Purpose Switch Module User's Manual

Agilent Technologies

Manual Part Number: E8480-90001
Printed in U.S.A. E0301

Contents

Agilent E8480A User's Manual

AGILENT TECHNOLOGIES WARRANTY STATEMENT

AGILENT PRODUCT: E8480A High Power General Purpose 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

E8480A High Power General Purpose Switch Module User's Manual
Edition 1

Documentation History

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

Edition 1 . . . . March, 2001

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.	∿	Alternating current (AC)
		⎓	Direct current (DC).
		⚡	Warning. Risk of electrical shock.
⏚	Indicates the field wiring terminal that must be connected to earth ground before operating the equipment — protects against electrical shock in case of fault.	**WARNING**	Calls attention to a procedure, practice, or condition that could cause bodily injury or death.
⊥ or ⊥	Frame or chassis ground terminal—typically connects to the equipment's metal frame.	**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:	*Basic, Emerging and Systems Technologies Product Generation Unit* 815 14th Street S.W. Loveland, CO 80537 USA

Declares, that the product

Product Name:	*High Power General Purpose Switch Module*
Model Number:	E8480A
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:

EMC	**Standard**	**Limit**
	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	Jim White
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
Getting Started

About This Chapter

This chapter describes the Agilent E8480A 40-Channel High Power General Purpose (GP) Switch module, contains information on how to program it using SCPI (Standard Commands for Programmable Instruments) commands, and provides an example program to check initial operation. Chapter contents include:

Agilent E8480A Module Description

The Agilent E8480A 40-Channel High Power General Purpose Switch Module is a single-slot VXIbus C-Size register-based product which can operate in a C-Size VXIbus mainframe. It is ideal for switching and routing high-current sources such as AC and DC power supplies in the automated test systems.

For the General Purpose Switch module, switching consists of opening or closing a channel relay to provide alternate connections to user devices. Scanning consists of closing a set of channel relays, one at a time.

Basic Operation

As shown in Figure 1-1, the E8480A module consists of 40 channels (channels 00 through 39). Each channel uses a non-latching Form A relay. Signals are switched by opening or closing the appropriate channel relays. At power-on, power-off, or reset, all channels of the module are open. User inputs and outputs to each channel are made via the connectors (J1, J2, and J3) on the module’s front panel. See “Connecting Field Wiring to the Module” on page 20 for more information.

In addition, to get the full life of the relays, varistors can be mounted onto the module’s PC board for relay protection. The "Emergency Reset" port on the module’s front panel provides an easy way to allow user to take immediate action for relay protection in case of an emergency. See “Protecting Relays and Circuits” on page 23 of this manual for more information.

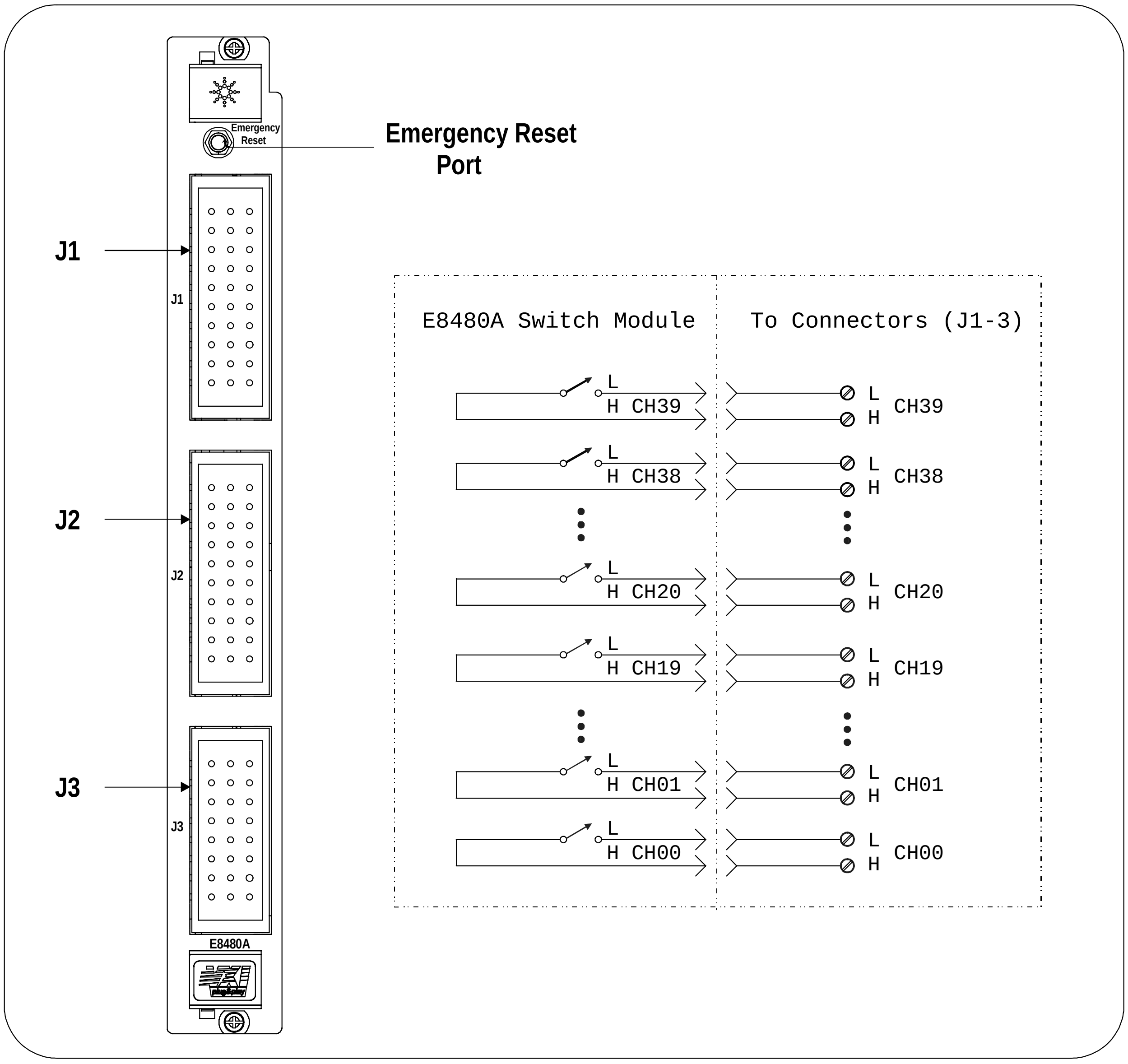


Figure 1-1. Front Panel and Simplified Schematic of the E8480A

Typical Configuration

Each relay channel on the E8480A module can accept a maximum current of 12A. The maximum voltage per channel is 150 Vdc or 280 Vac. The maximum rated power capacity (external load) is 3360 VA or 360 Wdc per channel.

For a Standard Commands for Programmable Instruments (SCPI) environment, one or more E8480A modules can be configured as a switchbox instrument. All modules within the switchbox can be addressed using a single interface address.

Instrument Definition

The plug-in modules installed in an Agilent mainframe or used with an Agilent command module are treated as independent instruments each having a unique secondary GPIB address. Each instrument is also assigned a dedicated error queue, input and output buffers, status registers and, if applicable, dedicated mainframe/command module memory space for readings or data. An instrument may be composed of a single plug-in module (such as a counter) or multiple plug-in modules (for a switchbox or scanning multimeter instrument).

Programming the Module

To program the module using SCPI commands, you must select the controller language, interface address, and SCPI commands to be used. See the *C-Size VXIbus System Configuration Guide* for detailed interface addressing and controller language information. For uses in other systems or mainframes, see the appropriate manuals. For more details of SCPI commands applicable to the module, refer to Chapter 4 of this manual.

NOTE *The module can also be programmed by directly writing to its registers. See Appendix B for the details on register programming.*

Specifying SCPI Commands

To address specific channels within an E8480A module, you must specify the appropriate SCPI command and channel addresses. Table 1-1 lists the most commonly used commands. Refer to Chapter 4 of this manual for a complete list of SCPI commands applicable to the module.

Table 1-1. Commonly Used SCPI Commands

SCPI Commands	Commands Description
CLOSe <*channel_list*>	Close (connect) the specified channels.
OPEN <*channel_list*>	Open (disconnect) the specified channels.
SCAN <*channel_list*>	Closes a serials of channels, one at a time.

Channel Addresses

Only valid channel addresses can be included in the *channel_list*. For the E8480A, the channel address has the form of (@*ccnn*) where,

cc = card number (01-99)
nn = channel number (00-39)

NOTE *Only valid channels can be accessed in a channel list or channel range. Also, the channel range must be from a lower channel number to a higher channel number. Otherwise, an error will be generated.*

To specify a *channel_list*, use the form of:

- (@*ccnn*) for a single channel
- (@*ccnn,ccnn*) for multiple channels
- (@*ccnn:ccnn*) for sequential channels
- (@*ccnn:ccnn,ccnn:ccnn*) for groups of sequential channels
- or any combination of the above.

Channel Number

The channel number (*nn* of the *channel_list*) identifies which relay on the selected module will be addressed. The channel numbers of the E8480A module are 00 through 39.

Card Number

The card number (*cc* of the *channel_list*) identifies which module within a switchbox will be addressed. The card number assigned depends on the switchbox configuration used. Leading zeroes can be ignored for the card number.

- **Single-module Switchbox**. In a single-module switchbox configuration, the card number is always 01.

- **Multiple-module Switchbox**. In a multiple-module switchbox configuration, modules are set to successive logical addresses. The module with the lowest logical address is always card number 01. The module with the next successive logical address is card number 02, and so on. Figure 1-2 illustrates the card numbers and logical addresses of a typical multiple-module switchbox installed in an Agilent C-Size mainframe with an Agilent command module.

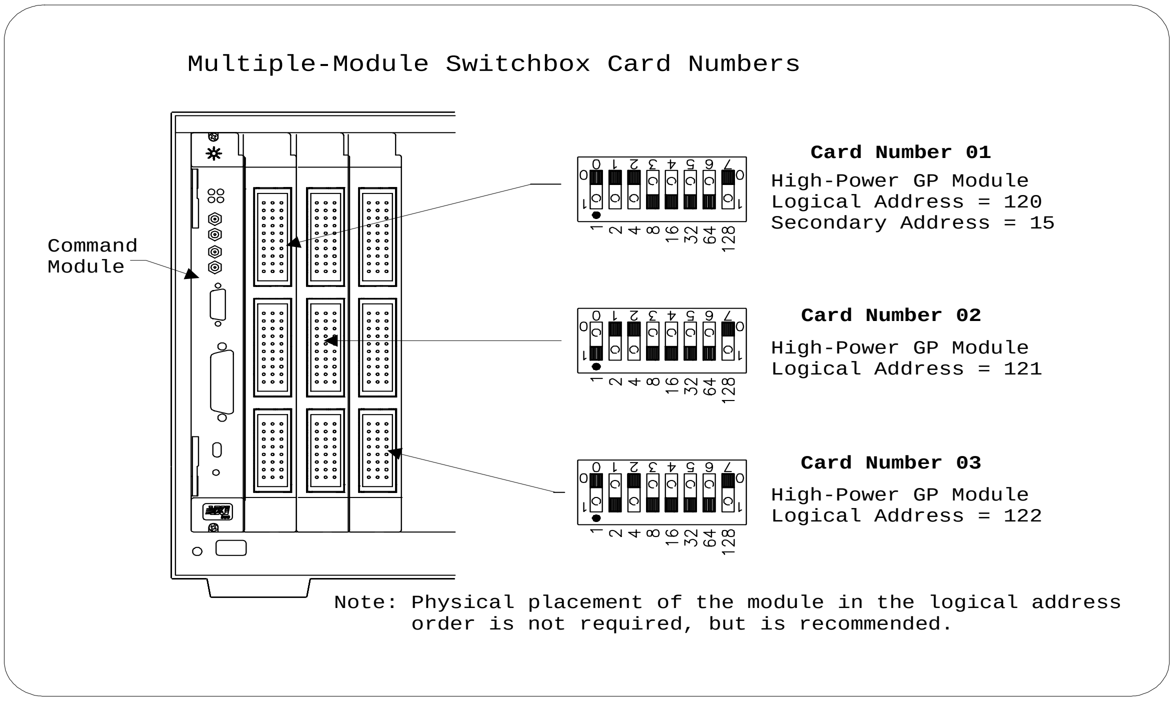


Figure 1-2. Multiple-Module Switchbox Instrument

Initial Operation

Use the following example programs to perform the initial operation on the E8480A module. To run the programs, an Agilent E1406A command module is required. Also, you must download the E8480A SCPI driver into the E1406A command module and have the Agilent SICL Library, the VISA extensions, and an Agilent 82350 GPIB card installed and properly configured in your PC.

In the examples, the computer interfaces to the mainframe via GPIB. The GPIB interface select code is 7, the GPIB primary address is 09, and the E8480A module is at logical address 120 (secondary address = 120/8 = 15). Refer to the *Agilent E1406A Command Module User's Guide* for more addressing information. For more details on the related SCPI commands used in the examples, see Chapter 4 of this manual.

Example: Closing a Channel (HTBasic)

This example program was written in HTBasic programming language. The program closes channel 102 of the module, then queries the channel closure state. The result is returned to the computer and displayed on the screen (1 = channel closed, 0 = channel open).

```
10   DIM Ch_Stat$[20]                        ! Dimension a variable.
20   OUTPUT 70915; "*RST"                    ! Resets the module.
30   OUTPUT 70915; "CLOS (@102)"             ! Close channel 102.
40   OUTPUT 70915; "CLOS? (@102)"            ! Query channel 102 closed
                                               state.
50   ENTER 70915; Ch_Stat$                   ! Enter results into Ch_stat$.
60   PRINT Ch_Stat$                          ! "1" should be displayed.
70   END
```

Example: Closing a Channel (C/C++)

This example program was developed and tested in Microsoft® Visual C++ 6.0 but should compile under any standard ANSI C compiler. The program closes channel 102 of the module, then queries the channel closure state. The result is returned to the computer and displayed on the screen (1 = channel closed, 0 = channel open).

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

    /* Module logical address is 120, secondary address is 15 */
#define INSTR_ADDR "GPIB0::9::15::INSTR"

int main()
{
   ViStatus errStatus;                 /* Status from each VISA call */
   ViSession viRM;                     /* Resource manager session */
   ViSession E8480A;                   /* Module session */
   char state[10];                     /* Channel state */
```

```
    /* Open the default resource manager */
   errStatus = viOpenDefaultRM (&viRM);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpenDefaultRM() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the module instrument session */
   errStatus = viOpen(viRM,INSTR_ADDR, VI_NULL,VI_NULL,&E8480A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Reset the module */
   errStatus = viPrintf(E8480A, "*RST;*CLS\n");
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Close channel 102 */
   errStatus = viPrintf(E8480A, "CLOS (@102)\n");
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Query state of channel 102 */
   errStatus = viQueryf(E8480A, "ROUT:CLOS? (@102)\n", "%t", state);
   if (VI_SUCCESS > errStatus) {
      printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
      return errStatus;}
   printf("Channel State is: %s\n", state);

    /* Close the module instrument session */
   errStatus = viClose (E8480A);
   if (VI_SUCCESS > errStatus) {
      printf("ERROR: viClose() returned 0x%x\n", errStatus);
      return 0;}

    /* Close the resource manager session */
   errStatus = viClose (viRM);
   if (VI_SUCCESS > errStatus) {
      printf("ERROR: viClose() returned 0x%x\n", errStatus);
      return 0;}

   return VI_SUCCESS;
}
```

Chapter 2
Configuring the Module

About This Chapter

This chapter shows how to configure the E8480A module for use in a VXIbus mainframe, install it in a mainframe, as well as how to connect external wiring to the module. Chapter contents include:

Warnings and Cautions

WARNING **SHOCK HAZARD. Only qualified, service-trained personnel who are aware of the hazards involved should install, configure, or remove the High-Power Switch module. Use only wire rated for the highest input voltage and disconnect all power sources from the mainframe and installed modules before installing or removing a module.**

Caution **MAXIMUM VOLTAGE/CURRENT. The maximum allowable voltage per channel for the Switch module is 150 Vdc or 280 Vac rms. The maximum current per channel is 12 Adc or ac (non-inductive). The maximum rated power capacity (external load) is 360 Wdc or 3360 VA per channel. Exceeding any limit may damage the High-Power Switch module.**

Caution **STATIC ELECTRICITY. Static electricity is a major cause of component failure. To prevent damage to the electrical components in the High-Power Switch module, observe anti-static techniques whenever removing a module from the mainframe or whenever working on a module. DO NOT install the Switch module without its metal shield attached.**

Setting the Logical Address

The logical address switch (LADDR) factory setting is 120. Valid address values are from 1 to 255. Figure 2-1 shows the address switch position and setting information.

NOTE *The address switch selected value must be a multiple of 8 if the module is the first module in a switchbox used with a VXIbus command module, and being instructed by SCPI commands.*

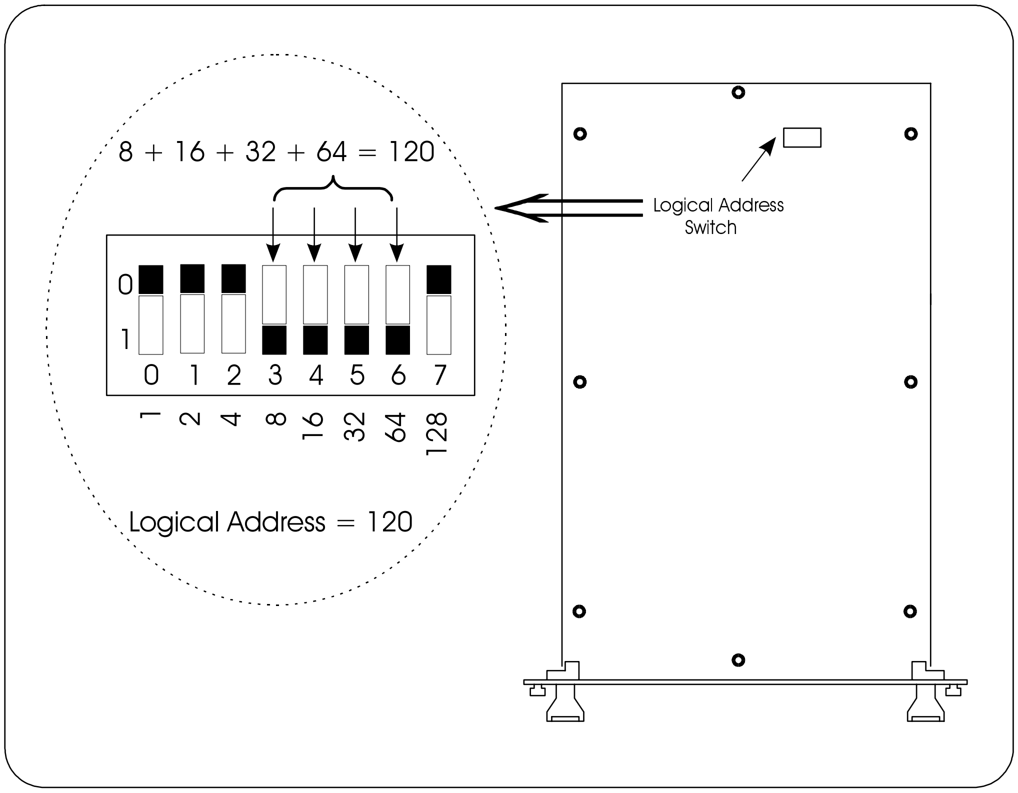


Figure 2-1. Setting the Logical Address Switch

Setting the Interrupt Priority

The E8480A module generates an interrupt after a channel has been closed. These interrupts are sent to, and acknowledgments are received from, the command module (Agilent E1406A) via the VXIbus backplane interrupt lines.

For most applications, the default interrupt priority line should not have to be changed. This is because the VXIbus interrupt lines have the same priority and interrupt priority is established by installing modules in slots numerically closest to the command module. Thus, slot 1 has a higher priority than slot 2, slot 2 has a higher priority than slot 3, etc.

By default, the interrupt priority level is Level 1. It can be set to any one of the VXI backplane lines 1-7 (corresponding to Levels 1-7) either by sending SCPI or directly writing to the Interrupt Selection Register. Level 1 is the lowest priority and Level 7 is the highest priority. The interrupt can also be disabled at power-up, after a SYSRESET, or by sending SCPI or directly writing to the Status/Control Register. See page 57 of this manual for more details of the related SCPI commands. For more information about register writing, see "Register-Based Programming" on page 87 of this manual.

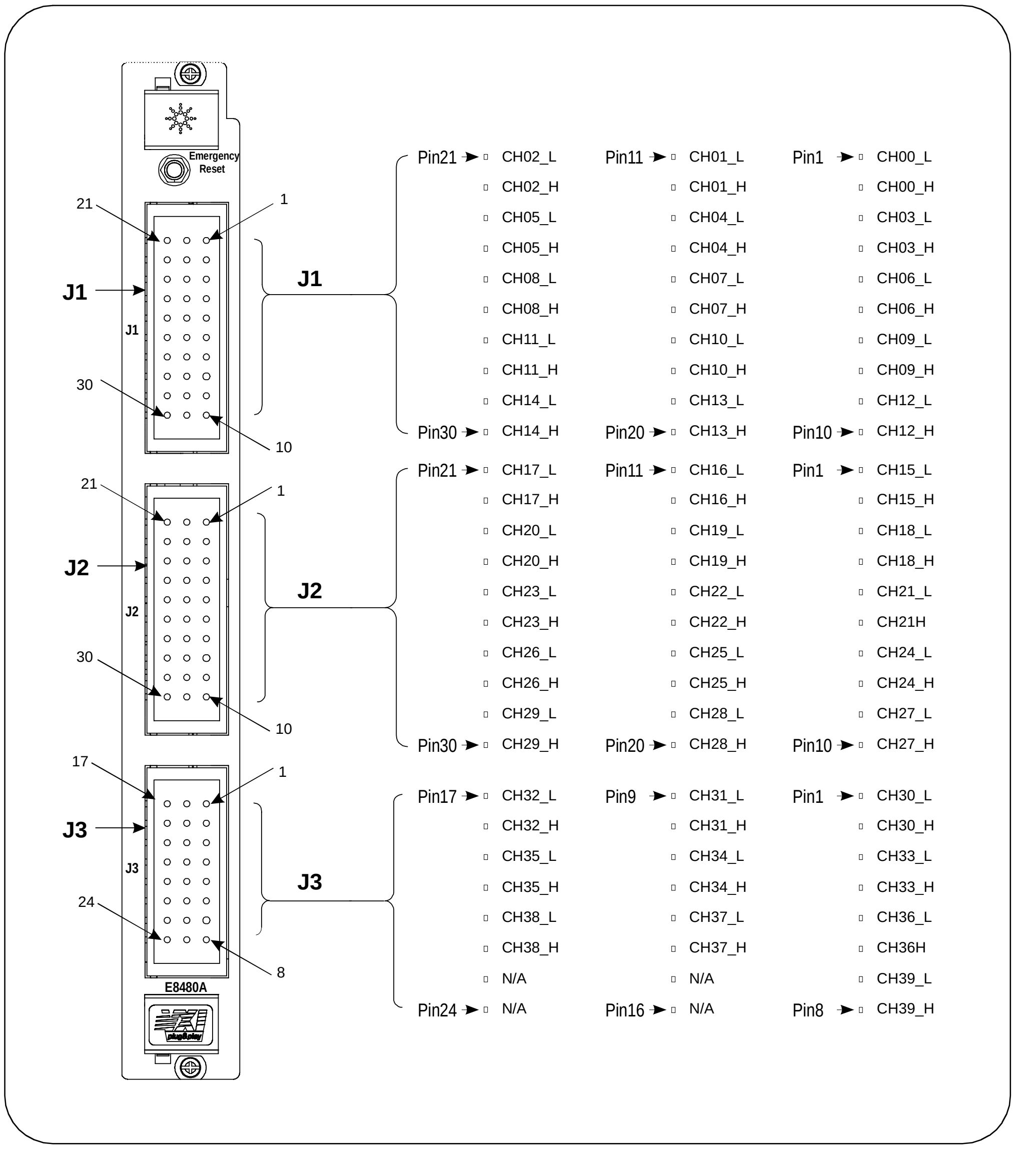
NOTE *Changing the interrupt priority level is not recommended. DO NOT change it unless specially instructed to do so. Refer to the E1406A Command Module User's Manual for more details.*

Connecting Field Wiring to the Module

User inputs to each channel are made via the user-supplied connectors which mates to the connectors (J1, J2, and J3) on the module's front panel. Additional accessories, such as cables, contacts and hand tools, are also required for wiring. The following sections provide the detailed information on the module's connectors pinout, the accessories required for user connection, as well as the procedure on how to connect field wiring to the module.

Front Panel & Connectors Pinout

Figure 2-2 shows the front panel of the E8480A module, as well as the connectors pinout and the corresponding channel numbers.



J1					
Pin21	CH02_L	Pin11	CH01_L	Pin1	CH00_L
	CH02_H		CH01_H		CH00_H
	CH05_L		CH04_L		CH03_L
	CH05_H		CH04_H		CH03_H
	CH08_L		CH07_L		CH06_L
	CH08_H		CH07_H		CH06_H
	CH11_L		CH10_L		CH09_L
	CH11_H		CH10_H		CH09_H
	CH14_L		CH13_L		CH12_L
Pin30	CH14_H	Pin20	CH13_H	Pin10	CH12_H
J2					
Pin21	CH17_L	Pin11	CH16_L	Pin1	CH15_L
	CH17_H		CH16_H		CH15_H
	CH20_L		CH19_L		CH18_L
	CH20_H		CH19_H		CH18_H
	CH23_L		CH22_L		CH21_L
	CH23_H		CH22_H		CH21H
	CH26_L		CH25_L		CH24_L
	CH26_H		CH25_H		CH24_H
	CH29_L		CH28_L		CH27_L
Pin30	CH29_H	Pin20	CH28_H	Pin10	CH27_H
J3					
Pin17	CH32_L	Pin9	CH31_L	Pin1	CH30_L
	CH32_H		CH31_H		CH30_H
	CH35_L		CH34_L		CH33_L
	CH35_H		CH34_H		CH33_H
	CH38_L		CH37_L		CH36_L
	CH38_H		CH37_H		CH36H
	N/A		N/A		CH39_L
Pin24	N/A	Pin16	N/A	Pin8	CH39_H

Figure 2-2. E8480A Module Front Panel and Connectors Pinout

Accessories for Wiring

The accessories that are necessary to connect the field wiring are not supplied with the module but can be ordered either from Agilent or from Positronic, Inc[1]. This allows you to purchase the number of connectors, contacts and tools you require for your application. Refer to Table 2-1 to order the accessories from Agilent. To purchase these products from Positronic, refer to Table 2-2 for order information.

NOTE *Agilent does not provide the tools (Hand Crimp Tool, Contact Insertion Tool and Contact Extraction Tool). You should order them from Positronic, Inc. as required.*

Table 2-1. Accessories Ordered from Agilent

Agilent Part No.	Description
Option 105	**Two 30-pin female connectors, each with 30 crimp-and-insert contacts:** Used to accept wires, then directly mating to the module’s J1 and J2 (30-pin) male connector.
Option 106	**One 24-pin female connector with 24 crimp-and-insert contacts:** Used to accept wires, then directly mating to the module’s J3 (24-pin) male connector.

Table 2-2. Recommended Accessories Ordered from Positronic

Positronic Part No.	Description
PLC30F7000	**30-pin female connector:** Used to accept wires, then directly mating to the module’s J1 or J2 (30-pin) male connector.
PLC24F7000	**24-pin female connector:** Used to accept wires, then directly mating to the module’s J3 (24-pin) male connector.
FC112N2	**Contacts:** Used to accept a wire size up to 12 AWG (4.0 mm^2) and carry a maximum current of 25 A. Wires are crimpt onto it, then inserted directly into the female connectors.
9501	**Hand Crimp Tool** - Used to crimp contacts onto wires.
9099	**Contact Insertion Tool** - Used to insert the contacted wires up to 12 AWG (4.0 mm^2) or smaller into the connector.
9081	**Contact Extraction Tool** - Used to remove the contacts from the connector.

1. Contact Positronic, Inc. 423 N. Campbell Ave. P.O. Box 8247, Springfield, MO 65801, U.S.A.
Telephone: 417-866-2322, Fax: 417-866-4115, Toll Free: 800-641-4054.
Email Address: info@positronic.com. Web Site: http://www.positronic.com

Attaching Connectors to the Module

Figure 2-3 shows the procedure to connect the field wiring. Use the guidelines below when making the connections.

- Maximum wire size is 12 AWG. Wire ends should be stripped 5.84 mm (0.23 inch) and tinned to prevent single strands from shorting to adjacent terminals.
- The maximum voltage that may be applied to any connector on the E8480A is 150 Vdc or 280 Vac. The maximum current that may be applied to any connector is 12 Adc or Aac. The maximum rated power capacity (external load) is 360 Wdc or 3360 VA per channel. Exceeding any limit may damage the module.

NOTE *We highly recommend to decentralize the channels when carrying high current. That is, six channels each carrying 12 A should use channels 0, 7, 14, 21, 28 and 35 instead of using channels 0 through 5.*

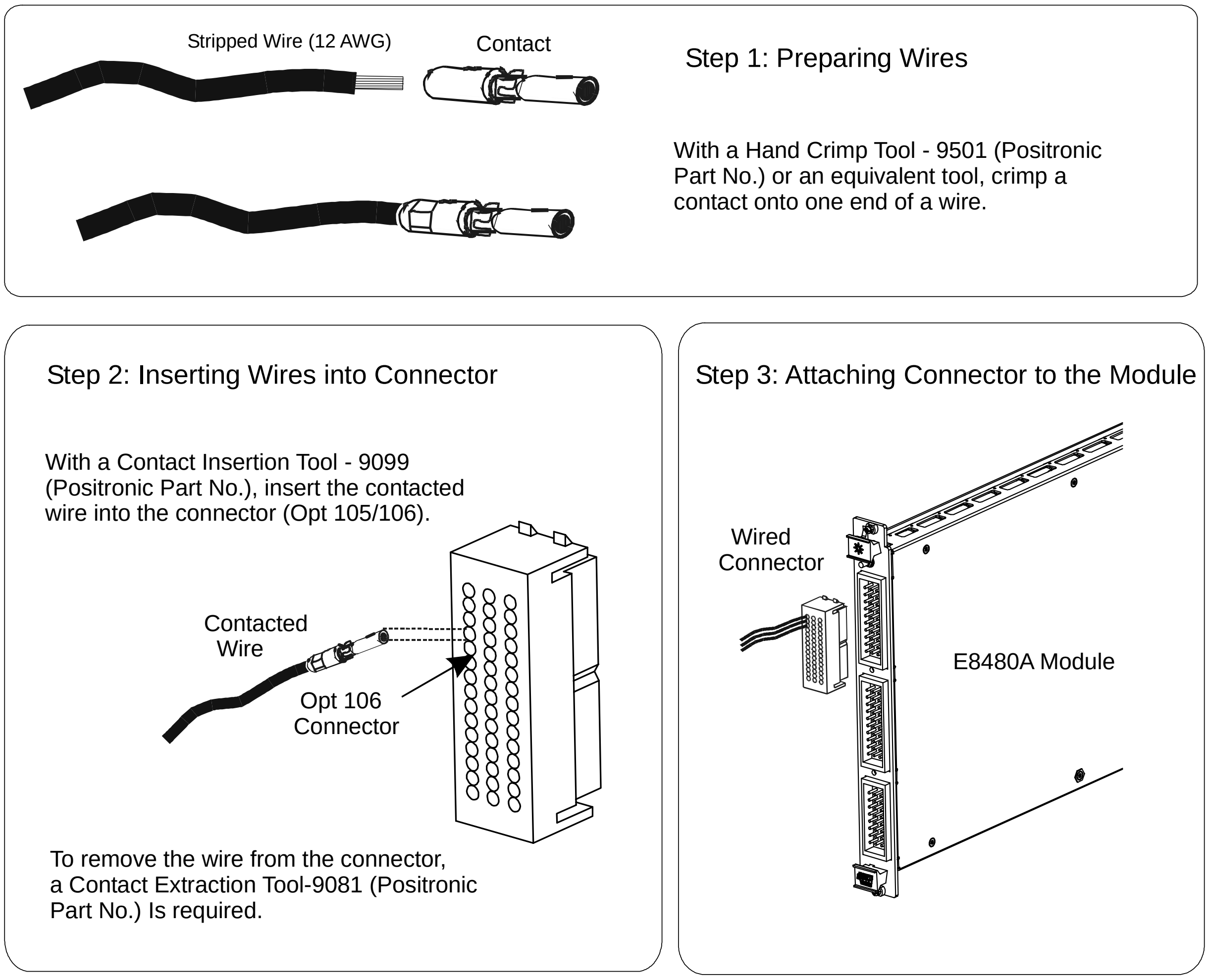

Stripped Wire (12 AWG)
Contact
Step 1: Preparing Wires
With a Hand Crimp Tool - 9501 (Positronic Part No.) or an equivalent tool, crimp a contact onto one end of a wire.
Step 2: Inserting Wires into Connector
With a Contact Insertion Tool - 9099 (Positronic Part No.), insert the contacted wire into the connector (Opt 105/106).
Contacted Wire
Opt 106 Connector
To remove the wire from the connector, a Contact Extraction Tool-9081 (Positronic Part No.) Is required.
Step 3: Attaching Connector to the Module
Wired Connector
E8480A Module

Figure 2-3. Wiring Connections

Protecting Relays and Circuits

Electromechanical relays are subject to normal wear-out. Relay life depends on several factors, such as relay loads, switching frequency, etc. See Appendix D on page 99 of this manual for details. To get the full life of the relays on the module, some protection circuits are designed on the module.

Adding Varistors

When relay contacts open or close, electrical breakdown can occur between the contacts. This can cause high frequency radiation, voltage and current surges, and physical damage to the relay contacts, especially when switching inductive loads.

When shipped from the factory, the E8480A module is not installed with the varistors. However, spaces have been made on the module's PC board for adding varistors for relay protection as required.

To protect the relay (labeled with *Kxxx* on the board), simply solder a varistor across the specified pads which are in parallel with the relay and labeled with *RVxxx* (*xxx* is same as the protected relay label). Now as the voltage goes up, the varistor draws current to protect the relay. Figure 2-4 shows the locations where the varistors can be added.

NOTE *Make certain that the selected varistor has a voltage rating sufficient for your application. We highly recommend to order P/N 0837-0227 for varistors with 250 VAC and P/N 0837-0507 for varistors with 300 VAC.*

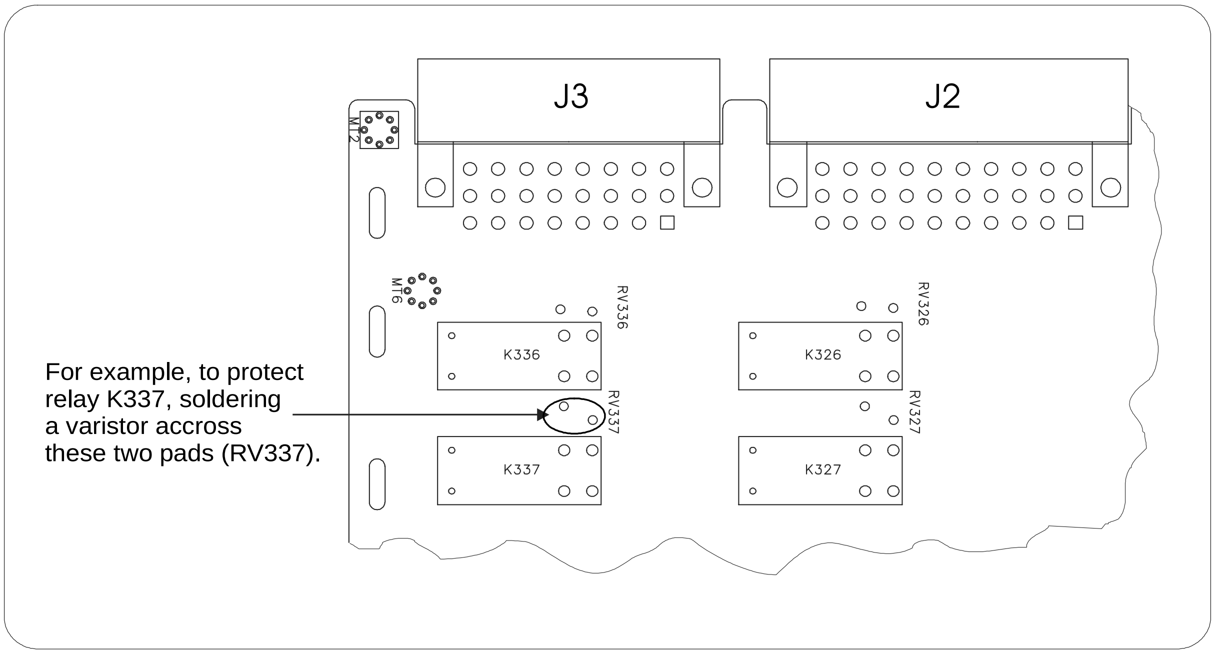

J3
J2
MT2
MT6
RV336
K336
RV326
K326
RV337
K337
RV327
K327
For example, to protect relay K337, soldering a varistor accross these two pads (RV337).

Figure 2-4. Adding Varistors for Relay Protection

Emergency Reset

In some hazardous cases (for example, the board inside temperature becomes too high), you may need to instantly open all channel relays and prevent any operation on the relays of the module. This can be done by applying a TTL low voltage or a +5V negative-going pulse to the "Emergency Reset" port (when enabled) on the front panel of the module, as shown in Figure 2-5.

At power-up or after a reset (*RST), the "Emergency Reset" port is disabled to accept an external emergency reset signal. You should enable the "Emergency Reset" port by DIAGnostic:EMERgency:TRIGger:STATe command as required. When enabled, the "Emergency Reset" port can accept a TTL low voltage or a +5V negative-going pulse to force the module to open all channel relays. Furthermore, all relays on the module can not be operated any more unless the current emergency state is cleared by DIAGnostic:EMERgency:CLEar command or *RST command. For more information on the related SCPI commands, see Chapter 4 starting on page 55 of this manual.

The "Emergency Reset" port can also be enabled or disabled by directly writing to the Emergency Control Register, see Appendix B starting on page 96 of this manual for details.

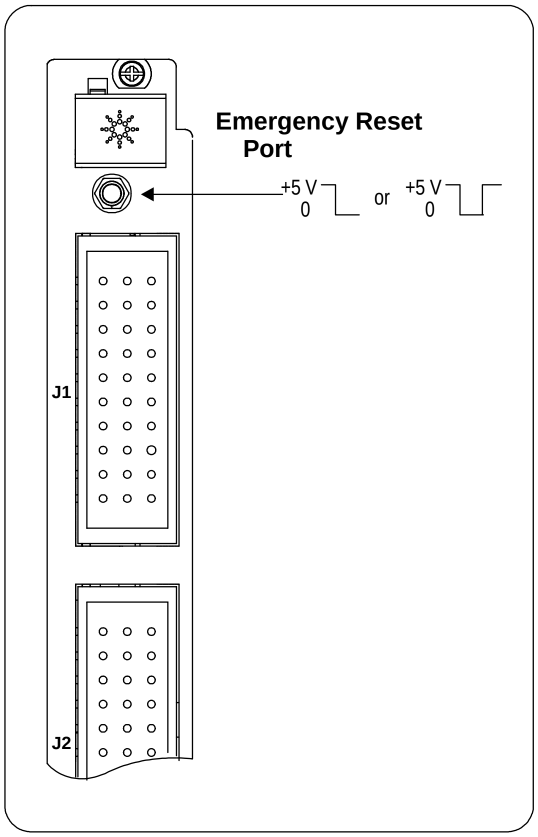


Figure 2-5. Emergency Reset Port of the Module

Maximum Allowable Module Switch Current

The E8480A has an individual channel current specification of 12 A. However, if you apply the 12 A to all the channels with a relay contact resistance of 0.1Ω, the power dissipation would be 576 W. Since the E8404A mainframe can only provide cooling for 100W per slot (keeps the temperature rise to 15^{o}C), this cannot be allowed to happen.

A reasonable currents and combination of channels for the entire module is shown in Figure 2-6. For example, six channels each carrying 12 A will produce about 86.5 W of internal dissipation, leading to a 15^{o}C temperature rise. Figure 2-6 shows how to derate the channels, in terms of current throughout the channels, to keep internal power dissipation under 86.5W or 15^{o}C temperature rise.

NOTE *We highly recommend to decentralize the channels when carrying high current. That is, six channels each carrying 12 A should use channels 0, 7, 14, 21, 28 and 35 instead of using channels 0 through 5.*

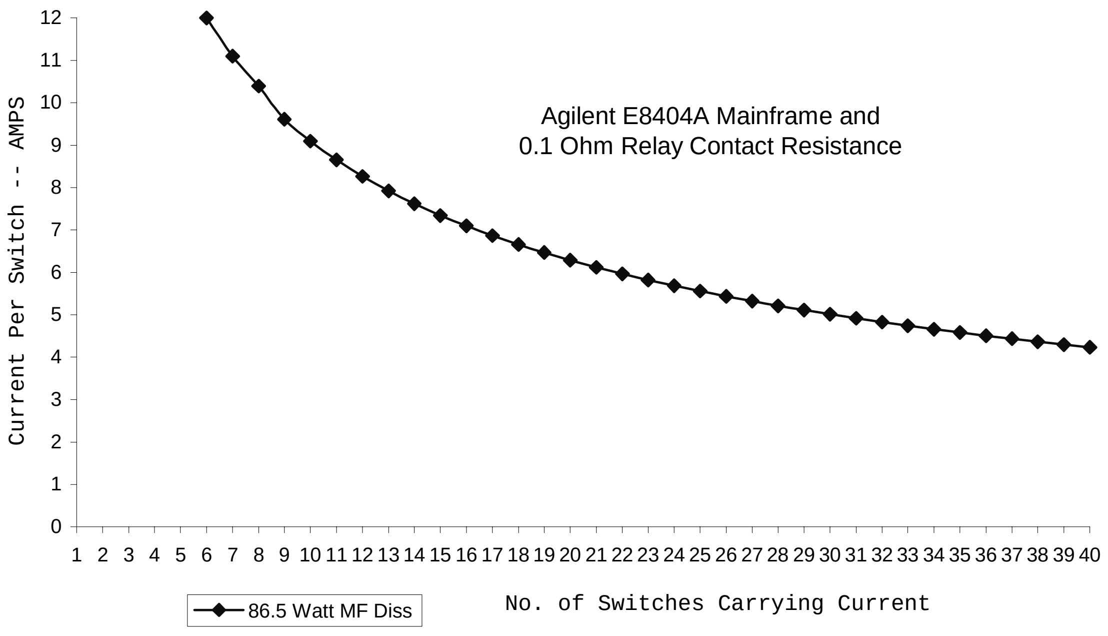


Figure 2-6. Allowable Switch Current

Notes:

Chapter 3
Using the Module

About This Chapter

This chapter uses typical examples to show how to use the E8480A module. See Chapter 4, "Command Reference" for the details of related commands used in this chapter. Chapter contents are:

All example programs in this chapter were developed on an external PC using HTBasic or Visual C/C++ as the programming language. They are tested with the following system configuration:

- An E1406A command module and an E8480A High Power General Purpose Switch module are installed in the mainframe.
- The computer is connected to the E1406A command module via GPIB interface. The GPIB select code is 7, the GPIB primary address is 09, and the E8480A module is at logical address 120 (secondary address = 120/8 = 15).
- The E8480A SCPI driver had been downloaded into the E1406A command module.
- The SICL Library, the VISA extensions, and an Agilent 82350 GPIB module had been installed and properly configured in the computer.

Refer to the *Agilent E1406A Command Module User's Guide* for more addressing information. For more details on the related SCPI commands used in this chapter, see Chapter 4 of this manual.

NOTE *Do not do register writes if you are controlling the module by a high level driver such as SCPI or VXIplug&play. This is because the driver will not know the module state and an interrupt may occur causing the driver and/or command module to fail.*

Module Commands Summary

Table 3-1 explains some of the SCPI commands used in this chapter. Refer to Chapter 4 for more information on these commands.

Table 3-1. Commonly Used Commands

Commands	Description
[ROUTe:]CLOSe <*channel_list*>	Close the channels in the channel list.
[ROUTe:]CLOSe? <*channel_list*>	Query the state of the channels in the channel list.
[ROUTe:]OPEN <*channel_list*>	Open the channels in the channel list.
[ROUTe:]OPEN? <*channel_list*>	Query the state of the channels in the channel list.
[ROUTe:]SCAN <*channel_list*>	Define the channel list to be scanned. Channels specified are closed one at a time.
INITiate[:IMMediate]	Start the scan sequence and close the first channel in the channel list.
TRIGger:SOURce <*source*>	Select the trigger source to advance the scan.

Power-On and Reset Conditions

At power-on or following a reset (*RST command), all channels of the module are open. The *RST command also invalidates the current scan list (that is, you must specify a new scan list for scanning). Command parameters are set to the default conditions as shown in Table 3-2.

Table 3-2. *RST Default Conditions

Parameter	Default	Description
ARM:COUNt	1	Number of scanning cycles is 1.
DIAGnostic:EMERgency:TRIGger:STATe	OFF	"Emergency Reset" port is disabled.
TRIGger:SOURce	IMM	Advances through a scanning list automatically.
INITiate:CONTinuous	OFF	Continuous scanning is disabled.
OUTPut:ECLTrg*n*[:STATe]	OFF	Trigger output from ECL trigger line is disabled.
OUTPut[:EXTernal][:STATe]	OFF	Trigger output from "Trig Out" port is disabled.
OUTPut:TTLTrg*n*[:STATe]	OFF	Trigger output from TTL trigger line is disabled.

Module Identification

The following example programs use the *RST, *CLS, *IDN?, SYST:CTYP?, and SYST:CDES? commands to reset and identify the module.

Example: Identifying Module (HTBasic)

```
10   DIM A$[50], B$[50], C$[50]            ! Dimension three string
                                             variables to fifty characters.
20   OUTPUT 70915; "*RST; *CLS"            ! Reset the module and clear
                                             Status Register.
30   OUTPUT 70915; "*IDN?"                 ! Query for module
                                             identification.
40   ENTER 70915; A$                       ! Enter the result into A$.

50   OUTPUT 70915; "SYST:CDES? 1"          ! Query for module description.
60   ENTER 70915; B$                       ! Enter the result into B$.

70   OUTPUT 70915; "SYST:CTYP? 1"          ! Query for module type.
80   ENTER 70915; C$                       ! Enter the result into C$

90   PRINT A$, B$, C$                      ! Print the contents of the
                                             variable A$, B$ and C$.
100  END
```

Example: Identifying Module (C/C++)

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

    /* Module logical address is 120, secondary address is 15 */
#define INSTR_ADDR "GPIB0::9::15::INSTR"

int main()
{
   ViStatus errStatus;                     /* Status from each VISA call */
   ViSession viRM;                         /* Resource manager session */
   ViSession E8480A;                       /* Module session */
   char id_string[256];                    /* ID string */
   char m_desp[256];                       /* Module description */
   char m_type[256];                       /* Module type */

    /* Open the default resource manager */
   errStatus = viOpenDefaultRM (&viRM);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpenDefaultRM() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the module instrument session */
   errStatus = viOpen(viRM,INSTR_ADDR, VI_NULL,VI_NULL,&E8480A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}
    /* Reset the module and clear the status registers */
   errStatus = viPrintf(E8480A, "*RST;*CLS\n");
   if(VI_SUCCESS > errStatus){
       printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
       return errStatus;}

    /* Query the module ID string */
   errStatus = viQueryf(E8480A, "*IDN?\n", "%t", id_string);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
       return errStatus;}
   printf("ID is %s\n", id_string);

    /* Query the module description */
   errStatus = viQueryf(E8480A, "SYST:CDES? 1\n", "%t", m_desp);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
       return errStatus;}
   printf("Module Description is %s\n", m_desp);

    /* Query the module type */
   errStatus = viQueryf(E8480A, "SYST:CTYP? 1\n", "%t", m_type);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
       return errStatus;}
   printf("Module Type is %s\n", m_type);

    /* Close the module instrument session */
   errStatus = viClose (E8480A);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viClose() returned 0x%x\n", errStatus);
       return 0;}

    /* Close the resource manager session */
   errStatus = viClose (viRM);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viClose() returned 0x%x\n", errStatus);
       return 0;}

   return VI_SUCCESS;
}
```

Switching Channels

One primary use of the E8480A High Power General Purpose Switch module is to switch and route high-current sources such as AC and DC power supplies in an automated test system. Use CLOSe <*channel_list*> to close the channel relays, or use OPEN <*channel_list*> to open the channel relays. The *channel_list* has the form of:

- (@*ccnn*) for a single channel
- (@*ccnn,ccnn*) for multiple channels
- (@*ccnn:ccnn*) for sequential channels
- (@*ccnn:ccnn,ccnn:ccnn*) for groups of sequential channels
- or any combination of the above.

where *cc* = card number (01-99) and *nn* = channel number (00-39).

Figure 3-1 shows a typical general purpose relay configuration for voltage switching. When the channel 00 relay is closed, the power supply voltage is applied to Device Under Test 1 (DUT-1). When the channel 02 relay is closed, the voltage is applied to Device Under Test 2 (DUT-2).

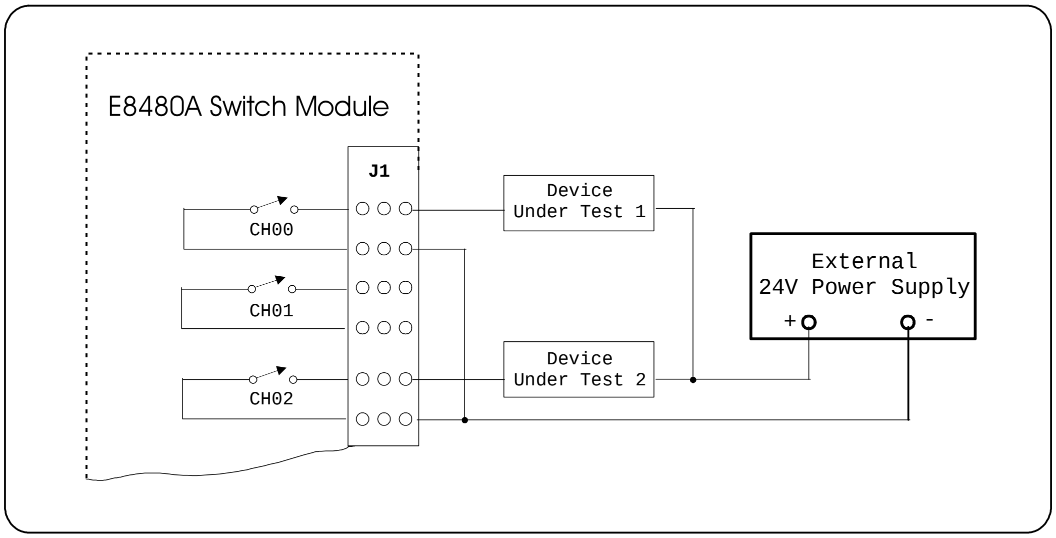


Figure 3-1. Voltage Switching

The following example programs were written in HTBasic and C/C++ programming languages respectively. In the example, it will close channels 00 and 02 to apply the external power supply to both devices (DUT-1 and DUT-2), then query to see whether they are closed. The result is returned to the computer and displayed (1 = channel closed, 0 = channel open).

Example: Closing Multiple Channels (HTBasic)

```
10   DIM A$[20]                                   ! Dimension a string variable to
                                                    twenty characters.
20   OUTPUT 70915; "*RST; *CLS"                   ! Reset the module and clear
                                                    Status Register.
30   OUTPUT 70915; "ROUT:CLOS (@100,102)"
                                                  ! Close channels 100 and 102.
40   OUTPUT 70915; "ROUT:CLOS? (@100,102)"
                                                  ! Query closure state of channels
                                                    100 and 102.
50   ENTER 70915; A$                              ! Enter the result into A$.
60   PRINT A$                                     ! "1,1" returned indicates they
                                                    are closed.
70   END
```

Example: Closing Multiple Channels (C/C++)

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

    /* Module logical address is 120, secondary address is 15 */
#define INSTR_ADDR "GPIB0::9::15::INSTR"

int main()
{
   ViStatus errStatus;                  /* Status from each VISA call */
   ViSession viRM;                      /* Resource manager session */
   ViSession E8480A;                    /* Module session */
   char stat[20];                       /* channel states*/

    /* Open the default resource manager */
   errStatus = viOpenDefaultRM (&viRM);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpenDefaultRM() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the module instrument session */
   errStatus = viOpen(viRM,INSTR_ADDR, VI_NULL,VI_NULL,&E8480A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Reset the module and clear status registers*/
   errStatus = viPrintf(E8480A, "*RST;*CLS\n");
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Close channels 00 and 02 */
   errStatus = viPrintf(E8480A, "CLOS (@100,102)\n");
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
      return errStatus;}
```

```
  /* Query channels 00 and 02 closure state */
  errStatus = viQueryf(E8480A, "ROUT:CLOS? (@100,102)\n", "%t", stat);
  if (VI_SUCCESS > errStatus) {
    printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
    return errStatus;}
  printf("The states of channels 00 and 02 are: %s\n", stat);

  /* Close the module instrument session */
  errStatus = viClose (E8480A);
  if (VI_SUCCESS > errStatus) {
    printf("ERROR: viClose() returned 0x%x\n", errStatus);
    return 0;}

  /* Close the resource manager session */
  errStatus = viClose (viRM);
  if (VI_SUCCESS > errStatus) {
    printf("ERROR: viClose() returned 0x%x\n", errStatus);
    return 0;}

  return VI_SUCCESS;
}
```

Scanning Channels

For the E8480A module, scanning channels consists of closing a specified set of channels, one at a time. You can scan any combination of channels for a single-module or a multiple-module switchbox. Single, multiple, or continuous scanning modes are available.

For multiple-module switchbox, the channels to be scanned can extend across switch modules. For example, for a two-module switchbox instrument, SCAN(@100:239) will scan all channels of both modules.

Use TRIGger:SOURce command to specify the source to advance the scan. Use OUTPut subsystem commands to select the E1406A command module Trig Out port, or ECL Trigger bus lines (0-1), or TTL Trigger bus lines (0-7). Use ARM:COUNt <*number*> to set multiple/continuous scans (from 1 to 32,767 scans). Use INITiate:CONTinuous ON to set continuous scanning. See Chapter 4 of this manual for information about these SCPI commands.

Example: Scanning Channels Using Trig In/Out Ports

This example uses E1406A command module's "Trig In" and "Trig Out" ports to synchronize E8480A module channel closures with an external measurement multimeter (Agilent 34401A). See Figure 3-2 for typical user connections. For measurement synchronization:

- -- E1406A's **Trig Out** port (connected to the 34401A multimeter's **External Trigger** port) is used by the E8480A module to trigger the multimeter to perform a measurement.
- -- E1406A's **Trig In** port (connected to the 34401A multimeter's **Voltmeter Complete** port) is used by the multimeter to advance the E8480A channel to scan.

For this example, the Low (L) contacts of channels 00-09 are connected to the different DUTs (devices under test). The High (H) contacts of channels 00-09 are connected together to the multimeter’s measurement input. These channels are then scanned and different DUTs are switched in for a measurement.

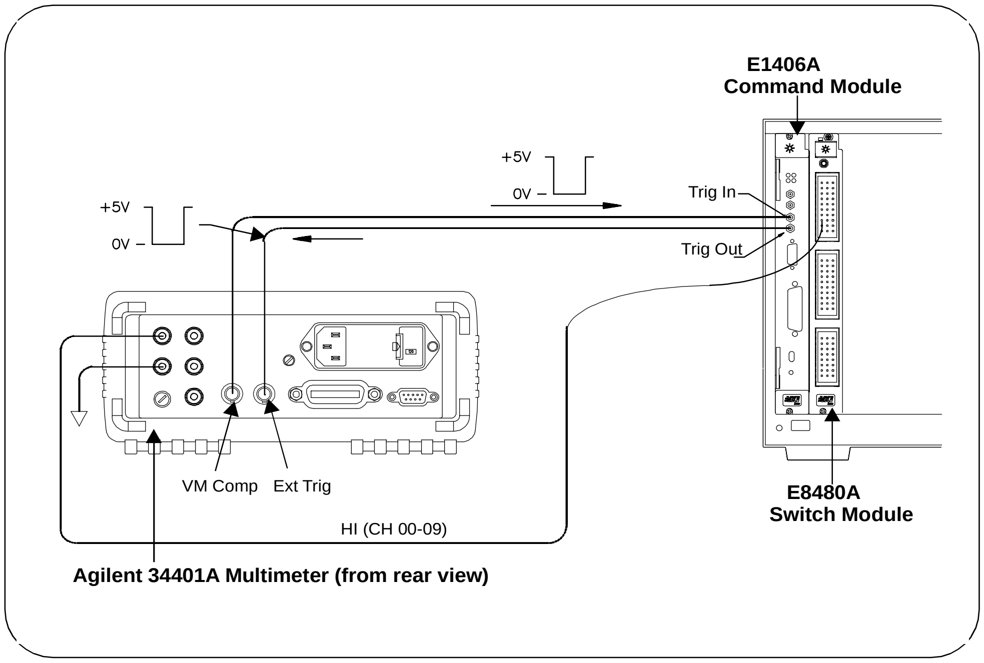


Figure 3-2. Scanning Channels using Trig In/out Ports

Programming with HTBasic

The following HTBasic program sets up the external multimeter (Agilent 34401A) to scan making DC voltage measurements. The E8480A switch module has a logical address 120 (secondary address 15), and the external multimeter has an address of 722.

```
10   DIM Rdgs(1:10)                     ! Dimension a variable to store
                                          readings.
20   OUTPUT 722; "*RST;*CLS"            ! Reset the dmm and clear its
                                          status registers.
30   OUTPUT 70915; "*RST;*CLS"          ! Reset the switch module and
                                          clear its status registers.
40   OUTPUT 722; "CONF:VOLT:DC 12"      ! Set the dmm for DCV
                                          measurement, 12 V maximum.
50   OUTPUT 722; "TRIG:SOUR EXT"        ! Set the dmm trigger source to
                                          EXTernal triggering.
60   OUTPUT 722; "TRIG:COUN 10"         ! Set the dmm trigger count
                                          to 10.
70   OUTPUT 722; "INIT"                 ! Set the dmm to the
                                          wait-for-trigger state.
80   WAIT 1                             ! Wait for 1 second.

90   OUTPUT 70915; "OUTP ON"            ! Set the switch module output
                                          pulses on E1406A "Trig Out"
                                          port when channel closed.
```

```
100 OUTPUT 70915; "TRIG:SOUR EXT"       ! Set the switch module trigger
                                          source to external triggering.
110 OUTPUT 70915; "SCAN (@100:109)"     ! Define channel list (00-09) for
                                          scanning.
120 OUTPUT 70915; "INIT"                ! Start scan and close channel
                                          100.
130 OUTPUT 722; "FETCH?"                ! Read measurement results
                                          from the dmm.
140 ENTER 722; Rdgs(*)                  ! Enter measurement results.
150 PRINT Rdgs(*)                       ! Display measurement results.
160 END
```

Programming with C/C++

The following program was written and tested in Microsoft® Visual C++ using the VISA extensions but should compile under any standard ANSI C compiler. This example configures the external multimeter (Agilent 34401A) to scan making DC voltage measurements.

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

    /* Module logical address is 120, secondary address is 15 */
#define INSTR_ADDR "GPIB0::9::15::INSTR"
    /* Interface address for 34401A Multimeter */
#define MULTI_ADDR "GPIB0::22::INSTR"

int main()
{
   ViStatus errStatus;                  /* Status from each VISA call */
   ViSession viRM;                      /* Resource manager session */
   ViSession E8480A;                    /* Module session */
   ViSession dmm;                       /* Multimeter session */
   int loop;                            /* loop counter */
   int opc_int;                         /* OPC? variable */
   double readings [10];                /* Reading Storage */

    /* Open the default resource manager */
   errStatus = viOpenDefaultRM (&viRM);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpenDefaultRM() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the switch module instrument session */
   errStatus = viOpen(viRM,INSTR_ADDR,VI_NULL,VI_NULL,&E8480A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the multimeter instrument session */
   errStatus = viOpen(viRM,MULTI_ADDR,VI_NULL,VI_NULL,&dmm);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}
```

```
  /* Set timeout value for multimeter and switch module */
viSetAttribute (dmm,VI_ATTR_TMO_VALUE,1000000);
viSetAttribute (E8480A,VI_ATTR_TMO_VALUE,1000000);

  /* Reset the multimeter and clear its status registers */
errStatus = viPrintf(dmm, "*RST;*CLS\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Configure dmm for DCV measurements, 12V maximum */
errStatus = viPrintf(dmm, "CONF:VOLT:DC 12\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set multimeter trigger source to EXTernal */
errStatus = viPrintf(dmm, "TRIG:SOUR EXT\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set trigger count to 10 */
errStatus = viPrintf(dmm, "TRIG:COUN 10\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Initialize multimeter, wait for triggering */
errStatus = viPrintf(dmm, "INIT\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* The dmm requires about 20 ms to change to wait-for-trigger state*/
_sleep(1000);

  /* Reset the switch module and clear its status registers */
errStatus = viPrintf(E8480A, "*RST;*CLS\n");
if (VI_SUCCESS > errStatus) {
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Enable the switch module output pulses on E1406A "Trig Out" port */
  /* when a channel is closed */
errStatus = viPrintf(E8480A, "OUTP ON\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}
  /* Set switch module trigger source to EXTernal */
errStatus = viPrintf(E8480A, "TRIG:SOUR EXT\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set up a scan list: channels 100 through 109*/
errStatus = viPrintf(E8480A, "SCAN (@100:109)\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Pause until ready */
errStatus = viQueryf(E8480A, "*OPC?\n", "%t", &opc_int);
if(VI_SUCCESS > errStatus){
    printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Start scan and close channel 100*/
errStatus = viPrintf(E8480A, "INIT\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Wait for scan to complete */
errStatus = viPrintf(E8480A, "STAT:OPER:ENAB 256\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

for (; ;){
    errStatus = viQueryf(E8480A, "*STB?\n", "%d", &opc_int);
    if (opc_int&0x80)
      break;}
printf("Scan has completed!\n");

  /* Get readings from multimeter */
errStatus = viQueryf(dmm, "FETC?\n", "%,10lf", readings);
if(VI_SUCCESS > errStatus){
    printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Display the measurement results */
for (loop=0;loop<10;loop++) {
    printf ("Reading %d is: %lf\n", loop, readings[loop]);  }

  /* Close the E8480A instrument session */
errStatus = viClose (E8480A);
if (VI_SUCCESS > errStatus) {
    printf("ERROR: viClose() returned 0x%x\n", errStatus);
    return 0;}
    /* Close the multimeter instrument session */
    errStatus = viClose (dmm);
    if (VI_SUCCESS > errStatus) {
        printf("ERROR: viClose() returned 0x%x\n", errStatus);
        return 0;}

    /* Close the resource manager session */
    errStatus = viClose (viRM);
    if (VI_SUCCESS > errStatus) {
        printf("ERROR: viClose() returned 0x%x\n", errStatus);
        return 0;}

    return VI_SUCCESS;
}
```

Example: Scanning Channels Using TTL Trigger

This example uses E1406A command module's TTL trigger bus lines to synchronize E8480A channel closures with a system multimeter (Agilent E1412A). See Figure 3-3 for typical user connections. For measurement synchronization:

-- E1406A's TTL trigger bus line 0 is used by the E8480A module to trigger the multimeter to perform a measurement.

-- E1406A's TTL trigger bus line 1 is used by the multimeter to advance the E8480A channel to scan.

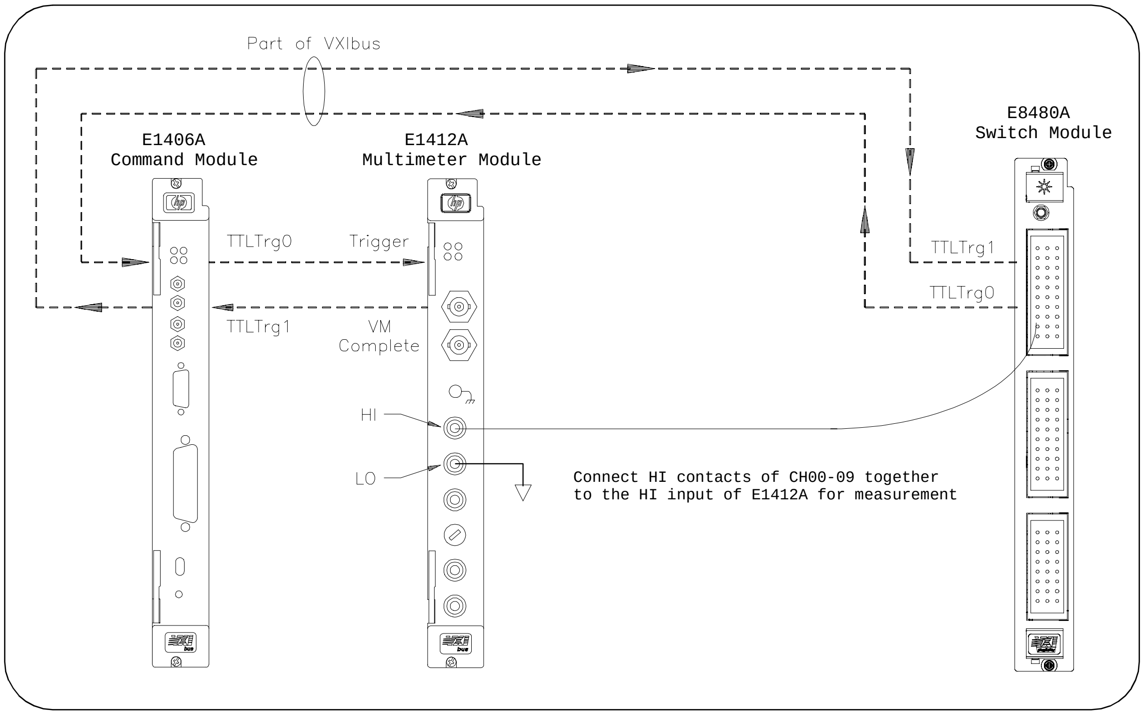


Figure 3-3. Scanning Using TTL Trigger Bus Lines

Figure 3-3 shows how to connect the switch module to the E1412A multimeter module.The connections shown with dotted lines are not actual hardware connections. These connections indicate how the E1406A firmware operates to accomplish the triggering. For this example, the Low (L) contacts of channels 00-09 are connected to the different DUTs. The High (H) contacts of channels 00-09 are connected together, and the measurements are taken from them. These channels are then scanned and different DUTs are switched in for a measurement.

Programming with HTBasic

This example program was written in HTBasic programming language. It configures the multimeter (E1412A) for DC voltage measurements, sets the switch module to scan channels 00 through 09. The E1412A multimeter has a GPIB address of 70903 and the switch module has a logical address of 120 (GPIB address of 70915).

```
10   DIM Rdgs(1:10)                       ! Dimension a variable to
                                            store readings.
20   OUTPUT 70903; "*RST;*CLS"            ! Reset the dmm and clear its
                                            status registers.
30   OUTPUT 70915; "*RST;*CLS"            ! Reset the switch module and
                                            clear its status registers.
40   OUTPUT 70903; "CONF:VOLT 12,MIN"     ! Set the dmm for DCV
                                            measurement, 12 V maximum,
                                            min resolution.
50   OUTPUT 70903; "OUTP:TTLT1:STAT ON"
                                          ! Set the dmm pulses TTL trigger
                                            line 1 on measurement
                                            complete.
60   OUTPUT 70903; "TRIG:SOUR TTLT0"      ! Set the dmm to be triggered by
                                            TTL trigger line 0.
70   OUTPUT 70903; "TRIG:DEL 0.01"        ! Set the dmm trigger delay time
                                            to 10 ms
80   OUTPUT 70903; "TRIG:COUN 10"         ! Set the dmm trigger count
                                            to 10.
90   OUTPUT 70903; "*OPC?"                ! Check to see if dmm ready
100  ENTER 70903; Check
110  OUTPUT 70903; "INIT"                 ! Set the dmm to the
                                            wait-for-trigger state.

120  OUTPUT 70915; "OUTP:TTLT0:STAT ON"
                                          ! Set the switch module pulses
                                            TTL trigger line 0 on channel
                                            closed.
130  OUTPUT 70915; "TRIG:SOUR TTLT1"      ! Set the switch module to be
                                            triggered by TTL trigger
                                            line 1.
140  OUTPUT 70915; "SCAN (@100:109)"      ! Define channels 00-09 for
                                            scanning.
150  OUTPUT 70915; "INIT"                 ! Initialize scan and close
                                            channel 100.

160  OUTPUT 70903; "FETCH?"               ! Read measurement results
                                            from the dmm.
170  ENTER 70903; Rdgs(*)                 ! Enter measurement results.
180  PRINT Rdgs(*)                        ! Display measurement results.
190  END
```

Programming with C/C++

The following program was written and tested in Microsoft® Visual C++ using the VISA extensions but should compile under any standard ANSI C compiler. This example configures the multimeter for DC voltage measurements, sets the switch module to scan channels 00 through 09.

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

    /* Switch module logical address is 120, secondary address is 15 */
#define INSTR_ADDR "GPIB0::9::15::INSTR"
    /* Interface address for E1412 Multimeter */
#define MULTI_ADDR "GPIB0::9::3::INSTR"

int main()
{
   ViStatus errStatus;                    /* Status from each VISA call*/
   ViSession viRM;                        /* Resource manager session */
   ViSession E8480A;                      /* Module session */
   ViSession E1412A;                      /* Multimeter session */
   int loop;                              /* loop counter */
   char opc_int[21];                      /* OPC? variable */
   double readings [10];                  /* Reading Storage*/

    /* Open the default resource manager */
   errStatus = viOpenDefaultRM (&viRM);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpenDefaultRM() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the switch module instrument session */
   errStatus = viOpen(viRM,INSTR_ADDR, VI_NULL,VI_NULL, &E8480A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the multimeter instrument session */
   errStatus = viOpen(viRM,MULTI_ADDR, VI_NULL,VI_NULL, &E1412A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Set timeout value for multimeter and switch module */
   viSetAttribute (E1412A, VI_ATTR_TMO_VALUE, 1000000);
   viSetAttribute (E8480A, VI_ATTR_TMO_VALUE, 1000000);

    /* Reset the multimeter, clear status registers */
   errStatus = viPrintf(E1412A, "*RST;*CLS\n");
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
      return errStatus;}
```

```
  /* Configure multimeter for DCV measurements, 12V max, min resolution */
errStatus = viPrintf(E1412A, "CONF:VOLT 12,MIN\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set multimeter to be triggered by TTL trigger line 0 */
errStatus = viPrintf(E1412A, "TRIG:SOUR:TTLT0\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Enable the E1412A pulses TTL trigger line 1 on measurement complete */
errStatus = viPrintf(E1412A, "OUTP:TTLT1 ON\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set trigger delay time to 1 ms, trigger count to 10 */
errStatus = viPrintf(E1412A, "TRIG:DEL 0.001;COUN 10\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Pause until multimeter is ready */
errStatus = viQueryf(E1412A, "*OPC?\n", "%t", opc_int);
if(VI_SUCCESS > errStatus){
    printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Initialize multimeter, wait for trigger */
errStatus = viPrintf(E1412A, "INIT\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Reset the switch module, clear the status registers */
errStatus = viPrintf(E8480A, "*RST;*CLS\n");
if (VI_SUCCESS > errStatus) {
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set the switch module pulses TTL Trigger line 0 on channel closed */
errStatus = viPrintf(E8480A, "OUTP:TTLT0 ON\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}
  /* Set the switch module pulses TTL Trigger line 0 on channel closed */
errStatus = viPrintf(E8480A, "TRIG:SOUR TTLT1\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Set up a scan list: channels 00 through 09 */
errStatus = viPrintf(E8480A, "SCAN (@100:109)\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Pause until ready */
errStatus = viQueryf(E8480A, "*OPC?\n", "%t", opc_int);
if(VI_SUCCESS > errStatus){
    printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Start scan and close channel 100 */
errStatus = viPrintf(E8480A, "INIT\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Wait for scan complete*/
errStatus = viPrintf(E8480A, "STAT:OPER:ENAB 256\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
    return errStatus;}

for (; ;){
    errStatus = viQueryf(E8480A, "*STB?\n", "%d", &opc_int);
    if (opc_int&0x80)
      break;}
printf("Scan has completed!\n");

  /* Get readings from multimeter */
errStatus = viQueryf(E1412A, "FETC?\n", "%,10lf", readings);
if(VI_SUCCESS > errStatus){
    printf("ERROR: viQueryf() returned 0x%x\n", errStatus);
    return errStatus;}

  /* Display measurement results */
for (loop=0;loop<10;loop++) {
    printf ("Reading %d is: %lf\n", loop, readings[loop]);  }

  /* Close the E8480A instrument session */
errStatus = viClose (E8480A);
if (VI_SUCCESS > errStatus) {
    printf("ERROR: viClose() returned 0x%x\n", errStatus);
    return 0;}
    /* Close the multimeter instrument session */
   errStatus = viClose (E1412A);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viClose() returned 0x%x\n", errStatus);
       return 0;}

    /* Close the resource manager session */
   errStatus = viClose (viRM);
   if (VI_SUCCESS > errStatus) {
       printf("ERROR: viClose() returned 0x%x\n", errStatus);
       return 0;}

   return VI_SUCCESS;
}
```

Using the Scan Complete Bit

You can use the Scan Complete bit (bit 8) in the Operation Status Register (in the command module) of a switchbox to determine when a scanning cycle completes (no other bits in the register apply to the switchbox). Bit 8 has a decimal value of 256 and you can read it directly with the STATus:OPERation[:EVENt]? command. See Page 76 in Chapter 4 for more information.

When enabled by the STAT:OPER:ENAB 256 command, the Scan Complete bit will be reported as bit 7 of the Status Byte Register. Use the GPIB Serial Poll or the IEEE 488.2 Common Command *STB? to read the Status Byte Register.

When bit 7 of the Status Register is enabled by the *SRE 128 Common Command to assert a GPIB Service Request (SRQ), you can interrupt the computer when the Scan Complete bit is set, after a scanning cycle completes. This allows the computer to do other operations while the scanning cycle is in progress.

The following example program was written in HTBasic programming language. It monitors bit 7 of the Status Byte Register to determine when the scanning cycle is complete. The computer interfaces with the E1406A command module over GPIB. The GPIB select code is 7, the GPIB primary address is 09, and the GPIB secondary address is 15.

Example: Using the Scan Complete Bit (HTBasic)

```
10   OUTPUT 70915; "*RST;*CLS"              ! Reset the switch module.
20   OUTPUT 70915; "STATUS:OPER:ENABLE 256"
                                            ! Enable Scan Complete Bit.
30   OUTPUT 70915; "TRIG:SOUR IMM"          ! Set the switch module for
                                              internal triggering.
40   OUTPUT 70915; "SCAN (@100:105)"        ! Set up channel list to scan.
50   OUTPUT 70915; "*OPC?"                  ! Wait for operation complete.
60   ENTER 70915; A$
70   PRINT "*OPC? =";A$
```

```
80  OUTPUT 70915; "*STB?"                 ! Query status byte register.
90  ENTER 70915; A$
100 PRINT "Switch Status = "; A$
110 OUTPUT 70915; "INIT"                  ! Start scan and close the
                                            channel 100.
120 I =0
130 WHILE(I =0)                           ! Stay in loop until value
                                            returned from the command
                                            SPOLL (70915).
140     I = SPOLL (70915)
150     PRINT "Waiting for scan to complete..."
160 END WHILE
170 I = SPOLL (70915)                     ! "128" returned indicates scan
                                            has completed.
180 PRINT "Scan complete: spoll = ";I
190 END
```

Example: Using the Scan Complete Bit (C/C++)

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

    /* Module logical address is 120, secondary address is 15 */
#define INSTR_ADDR "GPIB0::9::15::INSTR"

int main()
{
   ViStatus errStatus;                 /* Status from each VISA call */
   ViSession viRM;                     /* Resource manager. session */
   ViSession E8480A;                   /* Module session */
   int scanbit;                        /* Variable for Scan Complete
                                          Bit*/

    /* Open the default resource manager */
   errStatus = viOpenDefaultRM (&viRM);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpenDefaultRM() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Open the module instrument session */
   errStatus = viOpen(viRM,INSTR_ADDR, VI_NULL,VI_NULL,&E8480A);
   if(VI_SUCCESS > errStatus){
      printf("ERROR: viOpen() returned 0x%x\n", errStatus);
      return errStatus;}

    /* Set timeout value for the module */
   viSetAttribute (E8480A,VI_ATTR_TMO_VALUE,1000000);

    /* Reset the module and clear its status registers */
   errStatus = viPrintf(E8480A, "*RST;*CLS\n");
   if (VI_SUCCESS > errStatus) {
      printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
      return errStatus;}
    /* Enable the Scan Complete Bit */
    errStatus = viPrintf(E8480A, "STAT:OPER:ENAB 256\n");
    if(VI_SUCCESS > errStatus){
        printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
        return errStatus;}

    /* Set trigger source to IMMediate for internal triggering */
    errStatus = viPrintf(E8480A, "TRIG:SOUR IMM\n");
    if(VI_SUCCESS > errStatus){
        printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
        return errStatus;}

    /* Specify a channel list for scanning */
    errStatus = viPrintf(E8480A, "SCAN (@100:105)\n");
    if(VI_SUCCESS > errStatus){
        printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
        return errStatus;}

    /* Start Scan and close channel 100 */
    errStatus = viPrintf(E8480A, "INIT\n");
    if(VI_SUCCESS > errStatus){
        printf("ERROR: viPrintf() returned 0x%x\n", errStatus);
        return errStatus;}

    /* Stay in loop until scan complete */
    for (; ;){
        errStatus = viQueryf(E8480A, "*STB?\n", "%d", &scanbit);
        printf("Waiting for scan to complete...\n");
        if (scanbit&0x80)
          break;}
    printf("Scan has completed!\n");

    /* Close the E8480A instrument session */
    errStatus = viClose (E8480A);
    if (VI_SUCCESS > errStatus) {
        printf("ERROR: viClose() returned 0x%x\n", errStatus);
        return 0;}

    /* Close the resource manager session */
    errStatus = viClose (viRM);
    if (VI_SUCCESS > errStatus) {
        printf("ERROR: viClose() returned 0x%x\n", errStatus);
        return 0;}

    return VI_SUCCESS;
}
```

Recalling and Saving States

The *SAV <*numeric_state*> command saves the current instrument state. The state number (0-9) is specified by the *numeric_state* parameter. The settings saved by this command are as follows:

- Channel relays states (open or closed)
- ARM:COUNt
- TRIGger:SOURce
- OUTPut:STATe
- INITiate:CONTinuous

The *RCL <*numeric_state*> command recalls a previously saved state specified by the *numeric_state* parameter. If no *SAV was previously executed for the *numeric_state*, *RST default settings are used.

Example: Saving and Recalling Instrument State (HTBasic)

The following example program was written in HTBasic programming language. It demonstrates how to save and recall the switch module states. It first closes channels 100 through 119, then saves current channel states to the state 3. After reset the module to open all channels of the module, then recall the stored state 3 and verify whether the channels are set to the saved state (channels 100 through 119 are closed).

```
10   DIM A$[100]                        ! Dimension a string variables to
                                          100 characters.
20   OUTPUT 70915; "*RST; *CLS"         ! Reset the module and clear
                                          Status Register.

30   OUTPUT 70915; "CLOS (@100:119)"    ! Close channels 00 through 19.
40   OUTPUT 70915; "*SAV 3"             ! Save as numeric state 3.

50   OUTPUT 70915; "*RST; *CLS"         ! Reset the module and clear
                                          Status Register.
60   OUTPUT 70915; "CLOS? (@100:139)"   ! Query all channels state after a
                                          reset.
70   ENTER 70915; A$                    ! Enter the result into A$.
80   PRINT "After a reset, all channels states: "; A$
                                        ! Print the contents of the
                                          variable A$.
90   OUTPUT 70915; "*RCL 3"             ! Recall numeric state 3.
100  OUTPUT 70915; "CLOS? (@100:139)"   ! Queries the closed channels
                                          after recalling the state 3.

110  ENTER 70915; A$                    ! Enter the result into A$.

120  PRINT "After recall, all channels states: "; A$
                                        ! Print the contents of the
                                          variable A$. 1s for the first 20
                                          channels and 0s for the
                                          remaining 16 channels should
                                          be displayed.
130  END
```

Querying the Module

All query commands end with a "?". The data is sent to the output buffer where you can retrieve it into your computer. The following summarizes the query commands you can use to obtain the specific information of the module. See Chapter 4 for more details of these commands.

Channel closed:	CLOS?
Channel open.	OPEN?
Module Description:	SYST:CDES?
Module Type:	SYST:CTYP?
System error:	SYST:ERR?
Emergency Status:	DIAG:EMER:STAT?
Emergency Port Status:	DIAG:EMER:TRIG:STAT?

Detecting Error Conditions

The SYSTem:ERRor? command queries the instrument's error queue for error conditions. If no error occurs, the switch module responds with 0,"No error". If errors do occur, the module will respond with the first one in its error queue. Subsequent queries continue to read the error queue until it is empty. The response takes the following form:

<err_number>, <err_message>

where *<err_number>* is an integer ranging from -32768 to 32767, and the *<err_message>* is a short description of the error and the maximum string length is 255 characters. See Appendix C of this manual for a listing of the module error numbers and corresponding messages.

Example: Querying Errors (HTBasic)

The following example program was written in HTBasic programming language. It attempts to access an illegal channel number and then polls for the error message.

```
10   DIM Err_num$[256]                   ! Dimension a string variable.
20   OUTPUT 70915; "CLOS (@140)"         ! Try to close an illegal
                                           channel 140.
30   OUTPUT 70915; ":SYST:ERR?"          ! Check for a system error.
40   ENTER 70915;Err_num$                ! Enter the error into Err_num$.

50   PRINT "Error: ";Err_num$            ! Print error +2001, "Invalid
                                           channel number".
60   END
```

Synchronizing the Instruments

This section shows how to synchronize a switch module with other instruments when making measurements. In the following example, the module switches a signal to a multimeter, then verifies that the switching is complete before the multimeter begins a measurement.

Example: Synchronizing the Instruments (HTBasic)

This example program was written in HTBasic language. Assuming the multimeter (E1412A) has the GPIB address of 70903 and the switch module has a logical address of 120 (GPIB address of 70915).

```
10   OUTPUT 70915; "*RST"                    ! Reset the module.
20   OUTPUT 70915; "CLOS (@101)"             ! Close a channel.
30   OUTPUT 70915; "*OPC?"                   ! Wait for operation complete.
40   ENTER 70915;OPC_value
50   OUTPUT 70915; "CLOS? (@101)"            ! Verify that the channel is
                                               closed.
60   ENTER 70915;A
70   IF A=1 THEN
80      OUTPUT 70903; "MEAS:VOLT:DC?"        ! When channel is closed, make
                                               the measure.
90      ENTER 70903; Meas_value
100     PRINT Meas_value                     ! Print the measured value.
110  ELSE
120     PRINT "CHANNEL NOT CLOSE"
130  END IF
140  END
```

Chapter 4
Command Reference

About This Chapter

This chapter describes Standard Commands for Programmable Instruments (SCPI) and summarizes IEEE 488.2 Common (*) commands applicable to the module. See the *E1406A Command Module User's Manual* for additional information on SCPI and common commands. This chapter contains the following sections:

Command Types

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

Common Command Format

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

```
*RST        *ESR <unmask>        *STB?
```

SCPI Command Format

The SCPI commands perform functions like closing/opening switches, making measurements, 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:]
   CLOSe <channel_list>
   SCAN <channel_list>
```

[ROUTe:] is the root command, CLOSe and SCAN are the second level commands with <*channel_list*> as a parameter.

Command Separator

A colon (:) always separates one command from the next lower level command as shown below:

```
ROUTe:SCAN <channel_list>
```

Colons separate the root command from the second level command (ROUTe:SCAN). If a third level existed, the second level is also separated from the third level by a colon.

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 TRIGger, then TRIG and TRIGGER are both acceptable forms. Other forms of TRIGger, such as TRIGG or TRIGGE will generate an error. You may use upper or lower case letters. Therefore, TRIGGER, trigger, and TrIgGeR are all acceptable.

Implied Commands

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

```
[ROUTe:]
    CLOSe? <channel_list>
```

The root command [ROUTe:] is an implied command. To make a query about a channel's present status, you can send either of the following command statements:

ROUT:CLOS? <*channel_list*> ***or*** CLOS? <*channel_list*>

Variable Commands

Some commands have what appears to be a variable syntax. For example:

OUTPut:TTLTrg*n*

In this command, the "*n*" is replaced by a number (range from 0 to 7). No space is left between the command and the number because the number is part of the command syntax instead of a parameter.

Parameters

Parameter Types. The following table contains explanations and examples of parameter types you might see later in this chapter.

Parameter Type	Explanations and Examples
Numeric	Accepts all commonly used decimal representations of number including optional signs, decimal points, and scientific notation. 123, 123E2, -123, -1.23E2, .123, 1.23E-2, 1.23000E-01. Special cases include MINimum, MAXimum, and DEFault.
Boolean	Represents a single binary condition that is either true or false ON, OFF, 1, 0
Discrete	Selects from a finite number of values. These parameters use mnemonics to represent each valid setting. An example is the TRIGger:SOURce <*source*> command where source can be BUS, EXT, HOLD, or IMM.

Optional Parameters. Parameters shown within square brackets ([ ]) are optional parameters. (Note that the brackets are not part of the command and are not sent to the instrument.) If you do not specify a value for an optional parameter, the instrument uses the default value. For example, consider the ARM:COUNt?[<MIN | MAX>] command. If you send the command without specifying a parameter, the present ARM:COUNt setting is returned. If you send the MIN parameter, the command returns the minimum count available. If you send the MAX parameter, the command returns the maximum count available. Be sure to place a space between the command and the parameter.

Linking Commands

Linking IEEE 488.2 Common Commands with SCPI Commands. Use a semicolon between the commands. For example:

```
*RST;CLOS (@100)   or   TRIG:SOUR  BUS;*TRG
```

Linking Multiple SCPI Commands. Use both a semicolon and a colon between the commands. For example:

```
ARM:COUN1;:TRIG:SOUR  EXT
```

SCPI also allows several commands within the same subsystem to be linked with a semicolon. For example:

```
ROUT:CLOS (@100);:ROUT:CLOS? (@100)
```

- *or* -

```
ROUT:CLOS (@100);CLOS? (@100)
```

SCPI Command Reference

This section describes the Standard Commands for Programmable Instruments (SCPI) reference commands for the E8480A module. Commands are listed alphabetically by subsystem and also within each subsystem.

ABORt

The **ABORt** command stops a scan in progress when the scan is enabled via the interface, and the trigger source is either TRIGger:SOURce BUS or TRIGger:SOURce HOLD.

Subsystem Syntax

```
ABORt
```

Comments

ABORt Actions: The ABORt command terminates the scan and invalidates the current channel list. When the ABORt command is executed, the last channel closed during scanning remains in the closed position.

Affect on Scan Complete Status Bit: Aborting a scan will not set the "scan complete" status bit.

Stopping Scan Enabled Via Interface: When a scan is enabled via an interface, and the trigger source is neither HOLD nor BUS, an interface clear command (CLEAR 7 or viClear () function in VISA) can be used to stop the scan. When the scan is enabled via the interface and TRIGger:SOURce BUS or HOLD is set, you can use ABORt command to stop the scan.

Restarting a Scan: Use the INITiate command to restart the scan.

Related Commands: ARM, INITiate:CONTinuous, [ROUTe:]SCAN, TRIGger

Example

Stopping a Scan with ABORt

This example stops a continuous scan in progress.

```
TRIG:SOUR  BUS                          ! BUS is trigger source.
INIT:CONT  ON                           ! Set continuous scanning.
SCAN  (@100:105)                        ! Set channel list to be scanned.
INIT                                    ! Start scan, close channel 100.
 .                                        .
 .                                        .
 .                                        .
ABOR                                    ! Abort scan in progress.
```

The **ARM** subsystem selects the number of scanning cycles (1 to 32,767) for each INITiate command.

Subsystem Syntax

```
ARM
    :COUNt  <number>  MIN | MAX
    :COUNt?  [<MIN | MAX>]
```

ARM:COUNt

ARM:COUNt *<number>* MIN | MAX allows scanning cycles to occur a multiple of times (1 to 32,767) with one INITiate command when INITiate:CONTinuous OFF | 0 is set. MIN sets 1 cycle and MAX sets 32,767 cycles.

Parameters

Name	Type	Range of Values	Default Value
<number>	numeric	1 - 32,767 | MIN | MAX	1

Comments

Number of Scans: Use only values between 1 and 32767, MIN, or MAX for the number of scanning cycles.

Related Commands: ABORt, INITiate[:IMMediate], INITiate:CONTinuous

***RST Condition:** ARM:COUNt 1

Example

Setting Ten Scanning Cycles

```
ARM:COUN  10                              ! Set 10 scanning cycles.
SCAN (@100:103)                           ! Scan channels 100 to 103.
INIT                                      ! Start scan, close channel 100.
```

ARM:COUNt?

ARM:COUNt? [<MIN | MAX>] returns the current number of scanning cycles set by ARM:COUNt. The current number of scan cycles is returned when MIN or MAX parameter is not specified. With MIN or MAX as a parameter, "1" is returned for the MIN parameter; or "32767" is returned for the MAX parameter regardless of the ARM:COUNt value set.

Parameters

Name	Type	Range of Values	Default Value
<MIN | MAX>	numeric	MIN = 1, MAX = 32,767	current cycles

Comments

Related Commands: INITiate[:IMMediate]

Example

Querying Number of Scanning Cycles

```
ARM:COUN 10        ! Set 10 scanning cycles per INIT
                     command.
ARM:COUN?          ! Query number of scanning cycles.
```

The **DIAGnostic** subsystem is used to control the module's interrupt capability, emergency protection capability, as well as the time interval between the two scanned channels. All these settings can also be queried with this subsystem.

Subsystem Syntax

```
DIAGnostic
    :EMERgency
        :CLEar  <card_number>
        :STATus? <card_number>
        :TRIGger
            :STATe <card_number>, <mode>
            :STATe? <card_number>
    :INTerrupt
        [:LINE]  <card_number>, <line_number>
        [:LINE]?  <card_number>
        :TIMer  <card_number>, <time_interval>
        :TIMer?  <card_number>
    :SCAN
        :DELay  <card_number>, <time_interval>
        :DELay?  <card_number>
    :TEST
        [:RELays]?
        :SEEProm?  <card_number>
```

DIAGnostic:EMERgency:CLEar

DIAGnostic:EMERgency:CLEar <*card_number*> command clears the emergency status of the selected module if an external emergency trigger has ever occurred on its "Emergency Reset" port when enabled. Use DIAGnostic:EMERgency:TRIGger:STATe command to enable/disable the "Emergency Reset" port. Use DIAGnostic:EMERgency:STATus? command to check whether an external emergency trigger happened or not.

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A

Comments

This Command Not Always Used: This command is not required if the external emergency trigger is disabled by the DIAGnostic:EMERgency:TRIGger:STATe command or no emergency trigger occurs on the "Emergency Reset" port.

Using This Command: Once an external emergency trigger occurs, all relays on the module are open and can not be operated any more. In such case, use DIAGnostic:EMERgency:CLEar command to clear the current emergency state and recover the operation on relays.

Related Commands: DIAGnostic:EMERgency:STATus?, DIAGnostic:EMERgency:TRIGger:STATe

Example **Clearing the Emergency State Occurred on Module #1**

```
DIAG:EMER:CLEar 1                          ! Clear emergency state of module #1.
```

DIAGnostic:EMERgency:STATus?

DIAGnostic:EMERgency:STATus? <*card_number*> command queries the selected module to determine whether an external emergency trigger had occurred on the "Emergency Reset" port while enabled. Return value of "1" indicates an external emergency trigger occurs on the port. Otherwise, return value of "0".

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A

Comments **Related Commands:** DIAGnostic:EMERgency:CLEar, DIAGnostic:EMERgency:TRIGger:STATe?

Example **Querying the Emergency State Happened or Not**

```
DIAG:EMER:CLEar 1                          ! Clear the emergency status ever
                                             occurred on module #1
DIAG:EMER:STAT? 1                          ! "0" returned indicates that the
                                             emergency status has been cleared.
```

DIAGnostic:EMERgency:TRIGger:STATe

DIAGnostic:EMERgency:TRIGger:STATe <*card_number*>, <*mode*> enables or disables the "Emergency Reset" port on the selected module to accept external emergency trigger. The <*mode*> is set to "1" if the port is enabled or "0" if disabled. By default, the "Emergency Reset" port is disabled at power-up.

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A
<mode>	boolean	ON | OFF | 1 | 0	OFF | 0

Comments **Enabling "Emergency Reset" Port:** When enabled, the "Emergency Reset" port can accept a TTL low voltage or a +5V negative-going pulse to force the module to open all channels. In such case, you can not operate the module any more unless clearing the current emergency state by sending the DIAGnostic:EMERgency:CLEar command or power-off the module.

Related Commands: DIAGnostic:EMERgency:CLEar, DIAGnostic:EMERgency:STATus?, DIAGnostic:EMERgency:TRIGger:STATe?

***RST Condition:** The "Emergency Reset" port is disabled.

Example **Enabling the "Emergency Reset" Port on Module #1**

DIAG:EMER:TRIG:STAT 1,1	*! Enable module #1 to accept emergency trigger.*

DIAGnostic:EMERgency:TRIGger:STATe?

DIAGnostic:EMERgency:TRIGger:STATe? <*card_number*> queries the present setting for the "Emergency Reset" port of the selected module. The command returns "1" if the port is enabled or "0" if disabled.

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A

Comments **Related /Commands:** DIAGnostic:EMERgency:TRIGger:STATe, DIAGnostic:EMERgency:STATus?

Example **Querying the "Emergency Reset" Port State (Enable/Disable)**

DIAG:EMER:TRIG:STAT 1,1	*! Enable module #1 to accept emergency trigger.*
DIAG:EMER:TRIG:STAT? 1	*! Return of "1" indicates the emergency function of module #1 is enabled.*

DIAGnostic:INTerrupt[:LINe]

DIAGnostic:INTerrupt[:LINe] <*card_number*>, <*line_number*> sets the interrupt line of the specified module. The <*card_number*> specifies which E8480A in a multiple-module switchbox, is being referred to. The <*line_number*> can be 1 through 7 corresponding to VXI backplane interrupt lines 1 through 7. The default value is 1 (lowest interrupt level).

NOTE *Changing the interrupt priority level is not recommended. DO NOT change it unless specially instructed to do so.*

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A
<line_number>	numeric	0 - 7	1

Comments **Disable Interrupt:** Setting <*line_number*> = 0 will disable the module’s interrupt capability.

Select an Interrupt Line: The *line_number* can be 1 through 7 corresponding to VXI backplane interrupt lines 1-7. Only one value can be set at one time. The default value is 1 (lowest interrupt level).

Related Commands: DIAGnostic:INTerrupt[:LINe]?

Example **Setting the Module's Interrupt Line to 1**

DIAG:INT:LIN 1, 1	*! Set the interrupt line of Module #1 to line 1.*

DIAGnostic:INTerrupt[:LINe]?

DIAGnostic:INTerrupt[:LINe]? <*card_number*> queries the module's VXI backplane interrupt line and the returned value is one of 1, 2, 3, 4, 5, 6, 7 which corresponds to the module's interrupt lines 1-7. The returned value being 0 indicates that the module's interrupt is disabled. The *card_number* specifies which E8480A in a multiple-module switchbox is being referred to.

Parameters

Name	Type	Range of Values	Default Value
<*card_number*>	numeric	1 - 99	N/A

Comments Return value of "0" indicates that the module's interrupt is disabled. Return values of 1-7 correspond to VXI backplane interrupt lines 1 through 7.

When power-on or reset the module, the default interrupt line is 1.

Example **Querying the Module's Interrupt Line**

DIAG:INT:LIN 1, 1	*! Set the interrupt line of Module #1 to line 1.*
DIAG:INT:LIN? 1	*! Query the module's interrupt line.*

DIAGnostic:INTerrupt:TIMer

DIAGnostic:INTerrupt:TIMer <*card_number*>, <*timer*> is used to set the amount of time (in second) the module will wait after a relay close or open command is given before sending an interrupt and clearing the "busy" bit. The *card_number* parameter specifies which E8480A in a multiple-module switchbox is to be set.

Parameters

Name	Type	Range of Values	Default Value
<*card_number*>	numeric	1 - 99	N/A
<*timer*>	numeric	0.000001 - 0.064 seconds	0.015 second

Comments We highly recommend you set the time to 15 ms for the E8480A relay.

*RST does not change the selected time.

NOTE *Setting the interrupt timer too small can cause system problems. We DO NOT recommend to change it unless specially instructed to do so.*

Example **Setting Interrupt Timer of Module #1 to 15 ms**

```
DIAG:INT:TIM 1, 0.015                    ! Set Module 1 Interrupt timer to 15 ms.
```

DIAGnostic:INTerrupt:TIMer?

DIAGnostic:INTerrupt:TIMer? <*card_number*> queries the specified module and returns the interrupt delay time set by the DIAG:INT:TIM command.

Example **Querying the Interrupt Timer Set for Module #1**

```
DIAG:INT:TIM? 1                          ! Query the interrupt timer set for the
                                           Module #1.
```

DIAGnostic:SCAN:DELay

DIAGnostic:SCAN:DELay <*card_number*>, <*delay_timer*> sets the amount of extra time (in second) the module will wait between opening one channel and closing the next in a scan operation. The *card_number* parameter specifies which E8480A in a multiple-module switchbox is to be set.

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A
<delay_timer>	numeric	0 - 6.5535 seconds	0 second

Example **Setting the Delay Time for Scanning Operation**

```
DIAG:SCAN:DEL 1, 0.01                    ! Module #1 will wait 10 ms between
                                           opening one channel and closing the
                                           next specified in the scan list.
```

DIAGnostic:SCAN:DELay?

DIAGnostic:SCAN:DELay? <*card_number*> queries the specified module and returns the delay time (in second) set by the DIAG:SCAN:DEL command.

Example **Querying the Scan Delay Time Set for Module #1**

```
DIAG:SCAN:DEL? 1                         ! Query the scan delay time setting on the
                                           Module #1.
```

DIAGnostic:TEST[:RELays]?

DIAGnostic:TEST[:RELays]? causes the instrument to perform a self test which includes writing to and reading from all relay registers and verifying the correct values. A failure may indicate a potential hardware problem.

Comments

Returned Value: Returns 0 if all tests passed; otherwise the card fails.

Error Codes: If the card fails, the returned value is in the form ***100*card number + error code***. Error codes are:

1 = Internal driver error;
2 = VXI bus time out;
3 = Card ID register incorrect;
5 = Card data register incorrect;
10 = Card did not interrupt;
11 = Card busy time incorrect;
40 = Relay register read and written data don't match.

WARNING **Disconnect any connections to the module when performing this function.**

Example

Performing Diagnostic Test to Check Error(s)

DIAG:TEST? *! Returned value can be either 0 or other value. "0" indicates that the system has passed the self test otherwise the system has an error.*

DIAGnostic:TEST:SEEProm?

DIAGnostic:TEST:SEEProm? *<card_number>* checks the integrity (checksum) of the serial EEPROM on the module. Return value of "0" if no error. Otherwise, return value of "-1".

Parameters

Name	Type	Range of Values	Default value
<card_number>	numeric	1 - 99	N/A

Comments

Related Commands: SYST:CTYPE? *<card_number>*

Example

Checking EEPROM Checksum on Module #1

DIAG:TEST:SEEProm? 1 *! Return "0" if no error.*

The **DISPlay** subsystem monitors the channel state of the selected module in a switchbox. This subsystem operates with an E1406A command module when a display terminal is connected. With an RS-232 terminal connected to the E1406A command module's RS-232 port, these commands control the display on the terminal, and would in most cases be typed directly from the terminal keyboard. It is possible however, to send these commands over the GPIB interface, and control the terminal's display. In this case, care must be taken that the instrument receiving the DISPlay command is the same one that is currently selected on the terminal; otherwise, the GPIB command will have no visible affect.

Subsystem Syntax

```
DISPlay
    :MONitor
        :CARD  <number>  | AUTO
        :CARD?
        [:STATe]  <mode>
        [:STATe]?
```

DISPlay:MONitor:CARD

DISPlay:MONitor:CARD <*number*> | AUTO selects the module in a switchbox to be monitored when the monitor mode is enabled. Use the DISPlay:MONitor:STATe command to enable or disable the monitor mode.

Parameters

Name	Type	Range of Values	Default Value
<*number*> | AUTO	numeric	1 - 99 | AUTO	AUTO

Comments

Selecting a Specific Module to be Monitored: Use the DISPlay:MONitor:CARD command to send the card number for the switchbox to be monitored.

Selecting the Present Module to be Monitored: Use the DISPlay:MONitor:CARD AUTO command to select the last module addressed by a switching command (for example, [ROUTe:]CLOSe).

***RST Conditions:** DISPlay:MONitor:CARD AUTO

Example

Selecting Module #2 in a Switchbox for Monitoring

```
DISPlay:MONitor:CARD 2              ! Select module #2 in a switchbox to be
                                      monitored.
```

DISPlay:MONitor:CARD?

DISPlay:MONitor:CARD? queries the setting of the DISPlay:MONitor:CARD command and returns the module in a switchbox being monitored.

DISPlay:MONitor[:STATe]

DISPlay:MONitor[:STATe] <*mode*> turns the monitor mode ON or OFF. When monitor mode is on, the RS-232 terminal display presents an array of values indicating the closed channels on the module. The display is dynamically updated each time a channel is opened or closed.

Parameters

Name	Type	Range of Values	Default Value
<*mode*>	boolean	ON | OFF | 1 | 0	OFF | 0

Comments

Monitoring Switchbox Channels: DISPlay:MONitor[:STATe] ON or DISPlay:MONitor[:STATe] 1 turns the monitor mode ON to show the channel state of the selected module. DISPlay:MONitor[:STATe] OFF or DISPlay:MONitor[:STATe] 0 turns the monitor mode OFF.

NOTE *Typing in another command on the RS-232 terminal will cause the* DISPlay:MONitor[:STATe] *to automatically be set to OFF (0). Use of the OFF parameter is useful only if the command is issued over the GPIB interface.*

Selecting the Module to be Monitored: Use the DISPlay:MONitor:CARD <*number*> | AUTO command to select the module.

Monitor Mode for an E8480A: When monitoring mode is turned ON, the states of all 40 channels are displayed at the bottom of the terminal in three groups (channels 0-15, channels 16-31, and channels 32-39). Following each channel range numbers are the channel states within two brackets in the order of the channels order. "1" represents the corresponding channel is closed and "0" represents the corresponding channel is open. For example, the display:

```
Chan: 0-15 (0110000000000001) 16-31 (1000000000000001)
32-39 (10000000)
```

The example indicates that channels 01, 02, 15, 16, 31 and 32 are closed.

***RST Condition:** DISPlay:MONitor[:STATe] OFF | 0.

Example

Enabling the Monitor Mode for Module #2

```
DISP:MON:CARD 2          ! Select module #2 in a switchbox to be
                           monitored.
DISP:MON ON              ! Turn on monitor mode.
```

DISPlay:MONitor[:STATe]?

DISPlay:MONitor[:STATe]? queries the monitor mode state whether it is set to ON or OFF.

The **INITiate** command subsystem selects continuous scanning cycles and starts the scanning cycle.

Subsystem Syntax

```
INITiate
    :CONTinuous  <mode>
    :CONTinuous?
    [:IMMediate]
```

INITiate:CONTinuous

INITiate:CONTinuous *<mode>* enables or disables continuous scanning cycles for the switchbox.

Parameters

Name	Type	Range of Values	Default Value
<mode>	boolean	ON | OFF | 1 | 0	OFF | 0

Comments

Continuous Scanning Operation: Continuous scanning is enabled with the INITiate:CONTinuous ON or INITiate:CONTinuous 1 command. Sending the INITiate:IMMediate command closes the first channel in the channel list. Each trigger from the trigger source specified by the TRIGger:SOURce command advances the scan through the channel list. A trigger at the end of the channel list closes the first channel in the channel list and the scan cycle repeats.

Noncontinuous Scanning Operation: Noncontinuous scanning is enabled with the INITiate:CONTinuous OFF or INITiate:CONTinuous 0 command. Sending the INITiate:IMMediate command closes the first channel in the channel list. Each trigger from the trigger source specified by the TRIGger:SOURce command advances the scan through the channel list. A trigger at the end of the channel list opens the last channel in the list and the scanning cycle stops.

Stopping Continuous Scan: Refer to the ABORt command on page 52.

Related Commands: ABORt, ARM:COUNt, INITiate[:IMMediate], TRIGger:SOURce.

***RST Condition:** INITiate:CONTinuous OFF | 0

Example

Enabling Continuous Scanning

This example enables continuous scanning of channels 100 through 105 of a single-module switchbox. Since TRIGger:SOURce IMMediate (default) is set, use an interface clear command (such as CLEAR 7 or viClear() in VISA) to stop the scan.

```
INIT:CONT  ON                              ! Enable continuous scanning.
SCAN  (@100:105)                           ! Set channel list to be scanned.
INIT                                       ! Start scan, close channel 100.
```

INITiate:CONTinuous?

INITiate:CONTinuous? queries the scanning state. With continuous scanning enabled, the command returns "1" (ON). With continuous scanning disabled, the command returns "0" (OFF).

Example **Querying Continuous Scanning**

```
INIT:CONT  ON                          ! Enable continuous scanning.
INIT:CONT?                             ! Query continuous scanning state.
                                         It returns "1" (ON).
```

INITiate[:IMMediate]

INITiate[:IMMediate] starts the scanning process and closes the first channel in the channel list. Successive triggers from the source specified by the TRIGger:SOURce command advances the scan through the channel list.

Comments **Starting the Scanning Cycle:** The INITiate:IMMediate command starts scanning by closing the first channel in the channel list. Each trigger received advances the scan to the next channel in the channel list. An invalid channel list generates an error (see the [ROUTe:]SCAN command on page 71).

Stopping Scanning Cycles: Refer to the ABORt command.

Related Commands: ABORt, ARM:COUNt, INITiate:CONTinuous, TRIGger, TRIGger:SOURce

Example **Enabling a Single Scan**

This example enables a single scan of channels 100 through 105 of a single-module switchbox. The trigger source to advance the scan is immediate (internal) triggering set with TRIGger:SOURce:IMMediate (default).

```
SCAN  (@100:105)                       ! Scan channels 00-05.
INIT                                   ! Start scan, close channel 00 (use
                                         immediate triggering).
```

OUTPut

The **OUTPut** command subsystem selects the source of the output trigger generated when a channel is closed during a scan. The selected output can be enabled, disabled, or queried. The three available outputs are ECLTrg, TTLTrg trigger buses, and the "Trig Out" port on the command module’s front panel (e.g. E1406A).

Subsystem Syntax

```
OUTPut
    :ECLTrgn       (:ECLTrg0 or :ECLTrg1)
        [:STATe]  <mode>
        [:STATe]?
    [:EXTernal]
        [:STATe]  <mode>
        [:STATe]?
    :TTLTrgn       (:TTLTrg0 through :TTLTrg7)
        [:STATe]  <mode>
        [:STATe]?
```

OUTPut:ECLTrg*n*[:STATe]

OUTPut:ECLTrg*n*[:STATe] <*mode*> selects and enables which ECL Trigger bus line (0 and 1) will output a trigger when a channel is closed during a scan. This is also used to disable a selected ECL Trigger bus line. "*n*" specifies the ECL Trigger bus line (0 or 1) and <*mode*> enables (ON or 1) or disables (OFF or 0) the specified ECL Trigger bus line.

Parameters

Name	Type	Range of Values	Default Value
n	numeric	0 or 1	N/A
<*mode*>	boolean	0 | 1 | OFF | ON	OFF | 0

Comments

Enabling ECL Trigger Bus: When enabled, a trigger pulse is output from the selected ECL Trigger bus line (0 or 1) each time a channel is closed during a scan. The output is a negative going pulse.

ECL Trigger Bus Line Shared by Switchboxes: Only one switchbox configuration can use the selected trigger at a time. When enabled, the selected ECL Trigger bus line (0 or 1) is pulsed by the switchbox each time a scanned channel is closed. To disable the output for a specific switchbox, send the OUTPut:ECLTrgn OFF or 0 command for that switchbox.

One Output Selected at a Time: Only one output (ECLTrg*n*, TTLTrg*n* or EXTernal) can be enabled at one time. Enabling a different output source will automatically disable the active output. For example, if ECLTrg0 is the active output and ECLTrg1 is enabled, ECLTrg0 will become disabled and ECLTrg1 will become the active output.

Related Commands: [ROUTe:]SCAN, TRIGger:SOURce, OUTPut:ECLTrg*n*[:STATe]?

***RST Condition:** OUTPut:ECLTrg*n*[:STATe] OFF (disabled)

Example **Enabling ECL Trigger Bus Line 0**

```
OUTP:ECLT0:STAT 1                   ! Enable ECL Trigger bus line 0
                                      to output pulse after each scanned
                                      channel is closed.
```

OUTPut:ECLTrg*n*[:STATe]?

OUTPut:ECLTrg*n*[:STATe]? queries the state of the specified ECL Trigger bus line. The command returns "1" if the specified ECL Trg bus line is enabled or "0" if it is disabled.

Example **Querying ECL Trigger Bus Enable State**

This example enables ECL Trigger bus line 1 and queries the enable state. The OUTPut:ECLTrg*n*? command returns "1" since the line is enabled.

```
OUTP:ECLT1:STAT 1                   ! Enable ECL Trigger bus line 1.
OUTP:ECLT1?                         ! Query bus enable state.
```

OUTPut[:EXTernal][:STATe]

OUTPut[:EXTernal][:STATe] *<mode>* enables or disables the "Trig Out" port on the E1406A command module to output a trigger when a channel is closed during a scan.

- OUTPut[:EXTernal][:STATe] ON **|** 1 enables the port.
- OUTPut[:EXTernal][:STATe] OFF **|** 0 disables the port.

Parameters

Name	Type	Range of Values	Default Value
<*mode*>	boolean	ON | OFF | 1 | 0	OFF | 0

Comments **Enabling "Trig Out" Port:** When enabled, a pulse is output from the "Trig Out" port each time a channel is closed during scanning. If disabled, a pulse is not output from the port after channel closures.

Output Pulse: The pulse is a +5 V negative-going pulse.

"Trig Out" Port Shared by Switchboxes: Only one switchbox configuration can use the selected trigger at a time. When enabled, the "Trig Out" port may is pulsed by the switchbox each time a scanned channel is closed. To disable the output for a specific switchbox, send the OUTP OFF or 0 command for that switchbox.

One Output Selected at a Time: Only one output (ECLTrg*n*, TTLTrg*n* or EXTernal) can be enabled at one time. Enabling a different output source will automatically disable the active output. For example, if TTLTrg1 is the active output and TTLTrg4 is enabled, TTLTrg1 will become disabled and TTLTrg4 will become the active output.

Related Commands: [ROUTe:]SCAN, TRIGger:SOURce

***RST Condition:** OUTPut[:EXTernal][:STATe] OFF (port disabled)

Example **Enabling "Trig Out" Port**

```
OUTP ON                              ! Enable "Trig Out" port to output pulse
                                       after each scanned channel is closed.
```

OUTPut[:EXTernal][:STATe]?

OUTPut[:EXTernal][:STATe]? queries the present state of the "Trig Out" port on the E1406A command module. The command returns "1" if the port is enabled or "0" if disabled.

Example **Querying "Trig Out" Port State**

```
OUTP  ON                             ! Enable "Trig Out" port for pulse output.
OUTP?                                ! Query port enable state.
```

OUTPut:TTLTrg*n*[:STATe]

OUTPut:TTLTrg*n*[:STATe] <*mode*> selects and enables which TTL Trigger bus line (0 to 7) will output a trigger when a channel is closed during a scan. This command is also used to disable a selected TTL Trigger bus line. "*n*" specifies the TTL Trigger bus line (0 to 7) and <*mode*> enables (ON or 1) or disables (OFF or 0) the specified TTL Trigger bus line.

Parameters

Name	Type	Range of Values	Default Value
n	numeric	0 to 7	N/A
<*mode*>	boolean	ON | OFF | 1 | 0	OFF | 0

Comments **Enabling TTL Trigger Bus:** When enabled, a pulse is output from the selected TTL Trigger bus line (0 to 7) after each channel is closed during a scan. If disabled, a pulse is not output from the selected TTL Trigger bus line after channel closures. The output is a negative-going pulse.

TTL Trigger Bus Line Shared by Switchboxes: Only one switchbox configuration can use the selected trigger at a time. When enabled, the selected TTL Trigger bus line (0 to 7) is pulsed by the switchbox each time a scanned channel is closed. To disable the output for a specific switchbox, send the OUTPut:TTLTrg*n* OFF or 0 command for that switchbox.

One Output Selected at a Time: Only one output (ECLTrg*n*, TTLTrg*n* or EXTernal) can be enabled at one time. Enabling a different output source will automatically disable the active output. For example, if TTLTrg1 is the active output and TTLTrg4 is enabled, TTLTrg1 will become disabled and TTLTrg4 will become the active output.

Related Commands: [ROUTe:]SCAN, TRIGger:SOURce, OUTPut:TTLTrg*n*[:STATe]?

***RST Condition:** OUTPut:TTLTrg*n*[:STATe] OFF (disabled)

Example **Enabling TTL Trigger Bus Line 7**

```
OUTP:TTLT7:STAT 1                    ! Enable TTL Trigger bus line 7 to output
                                       pulse after each scanned channel is
                                       closed.
```

OUTPut:TTLTrg*n*[:STATe]?

OUTPut:TTLTrg*n*[:STATe]? queries the present state of the specified TTL Trigger bus line. The command returns "1" if the specified TTLTrg bus line is enabled or "0" if disabled.

Example **Querying TTL Trigger Bus Enable State**

This example enables TTL Trigger bus line 7 and queries the enable state. The OUTPut:TTLTrg*n*? command returns "1" since the port is enabled.

```
OUTP:TTLT7:STAT 1                    ! Enable TTL Trigger bus line 7.
OUTP:TTLT7?                          ! Query bus enable state.
```

The [ROUTe:] command subsystem controls switching and scanning operations for the E8480A modules in a switchbox.

Subsystem Syntax

```
[ROUTe:]
     CLOSe  <channel_list>
     CLOSe?  <channel_list>
     OPEN  <channel_list>
     OPEN?  <channel_list>
     SCAN  <channel_list>
```

[ROUTe:]CLOSe

[ROUTe:]CLOSe *<channel_list>* closes the channels specified in the *channel_list*. *Channel_list* is in the form (@ccnn), where cc = card number (01-99) and nn = channel number (00-39).

Parameters

Name	Type	Range of Values	Items
<channel_list>	numeric numeric	1 - 99 00 - 39	card (cc) channel (nn)

Comments

Closing Channels: To close:

-- a single channel, use CLOS (@ccnn);
-- multiple channels, use CLOS (@ccnn,ccnn,...);
-- sequential channels, use CLOS (@ccnn:ccnn);
-- groups of sequential channels, use CLOS (@ccnn:ccnn;ccnn:ccnn);
-- or any combination of the above.

Closure order for multiple channels with a single command is not guaranteed. Use sequential CLOSe commands when needed.

NOTE *Channel numbers in the <channel_list> can be in any random order.*

Related Commands: [ROUTe:]OPEN, [ROUTe:]CLOSe?

***RST Condition:** All channels are open.

Example

Closing Multiple Channels

This example closes channels 100 and 213 of a two-module switchbox.

```
CLOS (@100,213)                          ! Close channels 100 and 213.
```

[ROUTe:]CLOSe?

[ROUTe:]CLOSe? *<channel_list>* returns the current state of the channel(s) queried. *Channel_list* is in the form (@ccnn). The command returns "1" if the channel is closed or returns "0" if the channel is open. If a list of channels is queried, a comma delineated list of 0 or 1 values is returned in the same order of the channel list.

Comments

Query is Software Readback: The ROUTe:CLOSe? command returns the current software state of the channel(s) specified. It does not account for relay hardware failures.

***Channel_list* Definition:** See "[ROUTe:]CLOSe" on page 69 for the channel_list definition.

NOTE *A maximum of 128 channels can be queried at one time. Therefore, if you want to query more than 128 channels, you must enter the query data in two separate commands.*

Example

Querying Channel Closure State

This example closes channels 100 and 213 of a two-module switchbox and queries channel closure. Since the channels are programmed to be closed, "1,1" is returned.

```
CLOS (@100,213)                ! Close channels 100 and 213.
CLOS? (@100,213)               ! Query channels 100 and 213 closure
                                 state, returned value "1,1" indicates
                                 that both channels are closed.
```

[ROUTe:]OPEN

[ROUTe:]OPEN *<channel_list>* opens the channels specified in the *channel_list. Channel_list* is in the form (@ccnn), where cc = card number (01-99) and nn = channel number (00-39).

Parameters

Name	Type	Range of Values	Items
<channel_list>	numeric numeric	1 - 99 00 - 39	card (cc) channel (nn)

Comments

Opening Channels: To open:

-- a single channel, use OPEN (@ccnn);
-- multiple channels, use OPEN (@ccnn,ccnn,...);
-- sequential channels, use OPEN (@ccnn:ccnn);
-- groups of sequential channels, use OPEN (@ccnn:ccnn;ccnn:ccnn);
-- or any combination of the above.

Opening order for multiple channels with a single command is not guaranteed.

Related Commands: [ROUTe:]CLOSe, [ROUTe:]OPEN?

***RST Condition:** All channels are open.

Example **Opening Channels**

This example opens channels 100 and 213 of a two-module switchbox.

```
OPEN (@100, 213)                              !Open channels 100 and 213.
```

[ROUTe:]OPEN?

[ROUTe:]OPEN? *<channel_list>* returns the current state of the channel(s) queried. The *channel_list* is in the form (@ccnn). The command returns "1" if channel(s) are open or returns "0" if channel(s) are closed. If a list of channels is queried, a comma delineated list of 0 or 1 values is returned in the same order of the channel list.

Comments **Query is Software Readback:** The ROUTe:OPEN? command returns the current software state of the channel(s) specified. It does not account for relay hardware failures.

***Channel_list* Definition:** See the [ROUTe:]OPEN command on page 70 for the channel_list definition.

NOTE *A maximum of 128 channels can be queried at one time. Therefore, if you want to query more than 128 channels, you must enter the query data in two separate commands.*

Example **Querying Channel Open State**

This example opens channels 100 and 213 of a two-module switchbox and queries channel 213 state. Since channel 213 is programmed to be open, "1" is returned.

```
OPEN  (@100,213)                              ! Open channels 100 and 213.
OPEN? (@213)                                  ! Query channel 213 state.
```

[ROUTe:]SCAN

[ROUTe:]SCAN *<channel_list>* defines the channels to be scanned. *Channel_list* is in the form (@ccnn), where cc = card number (01-99) and nn = channel number (00-39).

Parameters

Name	Type	Range of Values	Items
<channel_list>	numeric numeric	1 - 99 00 - 39	card (cc) channel (nn)

Comments

Defining Scan List: When ROUTe:SCAN is executed, the channel list is checked for valid card and channel numbers. An error is generated for an invalid channel list.

Scanning Channels: To scan:

-- a single channel, use SCAN (@ccnn);
-- multiple channels, use SCAN (@ccnn,ccnn,...);
-- sequential channels, use SCAN (@ccnn:ccnn);
-- groups of sequential channels, use SCAN (@ccnn:ccnn;ccnn:ccnn);
-- or any combination of the above.

Scanning Operation: When a valid channel list is defined, INITiate[:IMMediate] begins the scan and closes the first channel in the *channel_list*. Successive triggers from the source specified by TRIGger:SOURce advance the scan through the channel list. At the end of the scan, the last trigger opens the last channel.

Stopping Scan: See the ABORt command on page 52.

Related Commands: TRIGger, TRIGger:SOURce

***RST Condition:** All channels are open.

Example

Scanning Channels Using External Triggers

This example uses external triggering (TRIG:SOUR EXT) to scan channels 100 through 109 of a single-module switchbox. The trigger source to advance the scan is the input to the "Trig In" on the E1406A command module. When INIT is executed, the scan is started and channel 00 is closed. Then, each trigger received at the "Trig In" port advances the scan to the next channel.

```
TRIG:SOUR  EXT                       ! Set trigger source to external.
SCAN  (@100:109)                     ! Set channel list to be scanned.
INIT                                 ! Start scanning cycle and close channel
                                       100.
(trigger externally)                 ! Advance scan to next channel.
```

STATus

The **STATus** subsystem reports the bit values of the Operation Status Register. It also allows you to unmask the bits you want reported from the Standard Event Register and to read the summary bits from the Status Byte Register.

Subsystem Syntax

```
STATus
    :OPERation
        :CONDition?
        :ENABle  <unmask>
        :ENABle?
        [:EVENt]?
    :PRESet
```

The STATus system contains four registers (that is, they reside in a SCPI driver, not in the hardware), two of which are under IEEE 488.2 control: the Standard Event Status Register (*ESE?) and the Status Byte Register (*STB?). The operational status bit (OPR), service request bit (RQS), standard event summary bit (ESB), message available bit (MAV) and questionable data bit (QUE) in the Status Byte Register (bits 7, 6, 5, 4 and 3 respectively) can be queried with the *STB? command. Use the *ESE? command to query the <*unmask*> value for the Standard Event Register (the bits you want logically OR'd into the summary bit). The registers are queried using decimal weighted bit values. The decimal equivalents for bits 0 through 15 are included in Figure 4-1 on page 74.

A numeric value of 256 executed in a STAT:OPER:ENABle <*unmask*> command allows only bit 8 to generate a summary bit. The decimal value for bit 8 is 256.

The decimal values are also used in the inverse manner to determine which bits are set from the total value returned by an EVENt or CONDition query. The E8480A module driver exploits only bit 8 of Operation Status Register. This bit is called the scan complete bit which is set whenever a scan operation completes. Since completion of a scan operation is an event in time, you will find that bit 8 will never appear set when STAT:OPER:COND? is queried. However, you can find bit 8 set with the STAT:OPER:EVEN? query command.

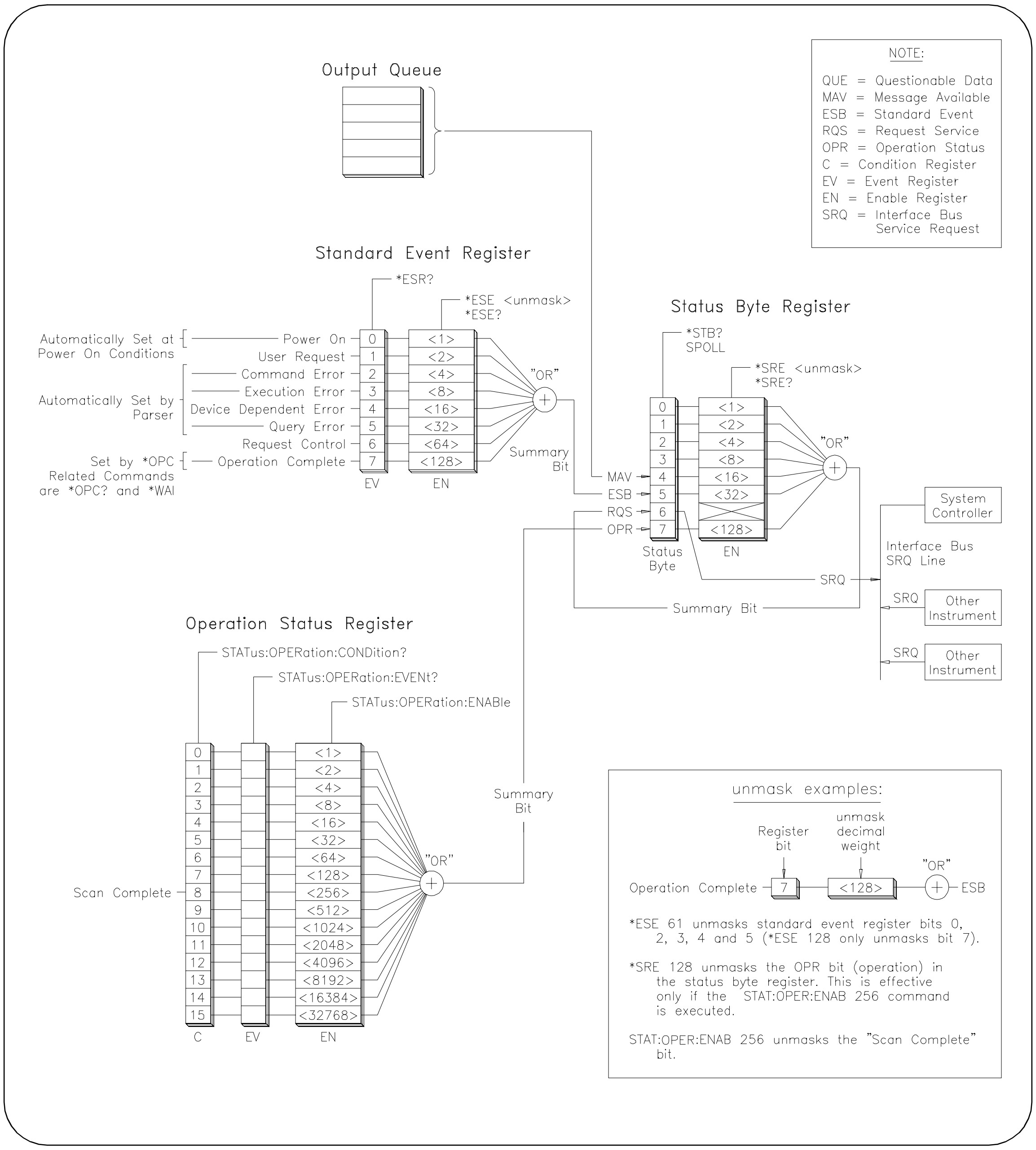


Figure 4-1. E8480A Status System Register Diagram

STATus:OPERation:CONDition?

STATus:OPERation:CONDition? returns the state of the Condition Register in the Operation Status Group. The state represents conditions which are part of the instrument's operation. The module's driver does not set bit 8 in this register (see STATus:OPERation[:EVENt]?).

STATus:OPERation:ENABle

STATus:OPERation:ENABle *<unmask>* sets an enable mask to allow events recorded in the Event Register (Operation Status Group) to send a summary bit to the Status Byte Register (bit 7). For the E8480A module, when bit 8 in the Operation Status Register is set to "1" and that bit is enabled by the STATus:OPERation:ENABle 256 command, bit 7 in the Status Byte Register is set to "1".

Parameters

Name	Type	Range of Values	Default Value
<unmask>	numeric	0 - 65,535	N/A

Comments

Setting Bit 7 of the Status Byte Register: STATus:OPERation:ENABle 256 sets bit 7 of the Status Register to 1 after bit 8 of the Operation Status Register is set to 1.

Related Commands: [ROUTe:]SCAN

Example

Enabling Operation Status Register Bit 8

```
STAT:OPER:ENAB  256                    ! Enable bit 8 of the Operation Status
                                         Register to be reported to bit 7
                                         (OPR) in the Status Byte Register.
```

STATus:OPERation:ENABle?

STATus:OPERation:ENABle? returns which bits in the Event Register (Operation Status Group) are unmasked.

Comments

Output Format: Returns a decimal weighted value from 0 to 65,535 indicating which bits are set to true.

Maximum Value Returned: The value returned is the value set by the STAT:OPER:ENAB *<unmask>* command. However, the maximum decimal weighted value used in this module is 256 (bit 8 set to true).

Example

Querying the Operation Status Enable Register

```
STAT:OPER:ENAB?                        ! Query the Operation Status Enable
                                         Register.
```

STATus:OPERation[:EVENt]?

STATus:OPERation[:EVENt]? returns which bits in the Event Register (Operation Status Group) are set. The Event Register indicates when there has been a time-related instrument event.

Comments

Setting Bit 8 of the Operation Status Register: Bit 8 (scan complete) is set to "1" after a scanning cycle completes. Bit 8 returns to "0" after sending the STATus:OPERation[:EVENt]? command.

Returned Data after sending the STATus:OPERation[:EVENt]? Command: The command returns "+256" if bit 8 of the Operation Status Register is set to "1". The command returns "+0" if bit 8 of the Operation Status Register is set to "0".

Event Register Cleared: Reading the Event Register with the STATus:OPERation:EVENt? command clears it.

Aborting a Scan: Aborting a scan will leave bit 8 set to 0.

Related Commands: [ROUTe:]SCAN

Example

Reading the Operation Status Register After a Scanning Cycle

```
STAT:OPER?                      ! Return the bit values of the Operation
                                  Status Register. +256 shows bit 8 is
                                  set to 1; +0 shows bit 8 is set to 0.
```

STATus:PRESet

STATus:PRESet affects only the Enable Register by setting all Enable Register bits to 0. It does not affect either the Status Byte Register or the Standard Event Status Register. PRESet does not clear any of the Event Registers.

The **SYSTem** subsystem returns the error numbers and error messages in the error queue of a switchbox. It can also return the types and descriptions of modules in a switchbox.

Subsystem Syntax

```
SYSTem
    :CDEScription?  <card_number>
    :CPON  <card_number> | ALL
    :CTYPe?  <card_number>
    :ERRor?
    :VERSion?
```

SYSTem:CDEScription?

SYSTem:CDEScription? *<card_number>* returns the description of a selected module in a switchbox.

Parameters

Name	Type	Range of Values	Default Value
<card_*number*>	numeric	1 - 99	N/A

Comments

Module Description: The SYSTem:CDEScription? *<card_number>* command returns:

"40-Channel High Power General Purpose Switch"

Example

Reading the Description of Module #1

```
SYST:CDES?  1                              ! Return the description of module #1.
```

SYSTem:CPON

SYSTem:CPON *<card_number>* | ALL resets the selected module, or multiple modules to their power-on state.

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99 or ALL	N/A

Comments

Module Power-on State: The power-on state of the module is all channels (relays) open. Note that SYSTem:CPON ALL and *RST opens all channels of all modules in a switchbox, while SYSTem:CPON *<number>* opens the channels in only the module (card) specified in the command.

Example

Setting Module #1 to its Power-on State

```
SYST:CPON 1                          ! Set module #1 to its power-on state (All
                                       channels open).
```

SYSTem:CTYPe?

SYSTem:CTYPe? *<card_number>* returns the module type of a selected module in a switchbox.

Parameters

Name	Type	Range of Values	Default Value
<card_number>	numeric	1 - 99	N/A

Comments

Agilent E8480A Module Model Number: Sending this command returns:

```
HEWLETT-PACKARD,E8480A,<10-digit number>,A.11.01
```

where the <10-digit number> is the module’s serial number and A.11.01 is an example of the module revision code number.

NOTE *The <10-digit number> returns 0 (zero) if the checksum of the EEPROM on the module has error.The checksum of EEPROM on the module is always checked each time the* SYST:CTYP? *<number>* *command is executed. Refer to DIAGnostic:TEST:SEEProm? command on page 60 for details.*

Related Commands: DIAG:TEST:SEEProm? <*card*_number>

Example

Reading the Model Number of Module #1

```
SYST:CTYP? 1                         ! Return the model number of module #1.
```

SYSTem:ERRor?

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

Comments **Error Numbers/Messages in the Error Queue:** Each error generated by a module stores an error number and corresponding error message in the error queue of a switchbox. The 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? command is sent. The errors are cleared first-in, first-out. When the queue is empty, each following SYSTem:ERRor? command returns: +0, "No error". To clear all error numbers/messages in the queue, execute the *CLS command.

Maximum Error Numbers/Messages in the Error Queue: The queue holds a maximum of 30 error numbers/messages for each switchbox. If the queue overflows, the last error number/message in the queue is replaced by: -350, "Too many errors". The least recent (oldest) 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 the version of the SCPI standard to which this instrument complies.

Comments **SCPI Version:** This command always returns a decimal value "1990.0", where "1990" is the year, and "0" is the revision number within that year.

Example **Reading SCPI Version**

```
SYST:VERS?                               ! Read the version of the SCPI standard.
```

The **TRIGger** command subsystem controls the triggering operation of the modules in a switchbox.

Subsystem Syntax

```
TRIGger
    [:IMMediate]
    :SOURce  <source>
    :SOURce?
```

TRIGger[:IMMediate]

TRIGger[:IMMediate] causes a trigger event to occur when the defined trigger source is TRIGger:SOURce BUS or TRIGger:SOURce HOLD. This can be used to trigger a suspended scan operation.

Comments

Executing the TRIGger[:IMMediate] Command: A channel list must be defined with [ROUTe:]SCAN <*channel_list*> and an INITiate[:IMMediate] command must be executed before TRIGger[:IMMediate] will execute.

BUS or HOLD Source Remains: If selected, the TRIGger:SOURce BUS or TRIGger:SOURce HOLD commands remain in effect after triggering a switchbox with the TRIGger[:IMMediate] command.

Related Commands: INITiate, [ROUTe:]SCAN, TRIGger:SOURce

Example

Advancing Scan Using TRIGger Command

This example uses the TRIGger command to advance the scan of a single-module switchbox from channel 100 through 103. Since TRIGger:SOURce HOLD is set, the scan is advanced one channel each time TRIGger is executed.

```
TRIG:SOUR  HOLD          ! Set trigger source to HOLD.
SCAN  (@100:103)         ! Define channel list to be scanned.
INIT                     ! Start scanning cycle, close channel 100.
loop statement           ! Start count loop.
TRIG                     ! Advance scan to next channel.
increment loop           ! Increment loop count.
```

TRIGger:SOURce

TRIGger:SOURce *<source>* specifies the trigger source to advance the channel list during scanning.

Parameters

Name	Type	Parameter Description
BUS	discrete	*TRG or GET command
ECLTrg*n*	numeric	ECL Trigger bus line 0 - 1
EXTernal	discrete	"Trig In" port
HOLD	discrete	Hold Triggering
IMMediate	discrete	Immediate Triggering
TTLTrg*n*	numeric	TTL Trigger bus line 0 - 7

Comments

Enabling the Trigger Source: The TRIGger:SOURce command only selects the trigger source. The INITiate[:IMMediate] command enables the trigger source. The trigger source must be selected with TRIGger:SOURce command before executing the INIT command.

Using the TRIGger Command: You can use TRIGger[:IMMediate] to advance the scan when TRIGger:SOURce BUS or TRIGger:SOURce HOLD is selected.

Using Bus Triggers: To trigger the switchbox with TRIGger:SOURce BUS selected, use the IEEE 488.2 common command *TRG or the GPIB Group Execute Trigger (GET) command.

Using TTL or ECL Trigger Bus Inputs: These triggers are from the VXI backplane trigger lines ECL[0,1] and TTL[0-7]. These may be used to trigger the "SWITCH" driver from other VXI instruments.

Using External Trigger Inputs: With TRIGger:SOURce EXTernal selected, only one switchbox at a time can use the external trigger input at the E1406A "Trig In" port. The trigger input is assigned to the first switchbox requesting the external trigger source (with a TRIGger:SOURce EXTernal command).

One Trigger Input Selected at a Time: Only one input (ECLTrg0 or 1; TTLTrg0, 1, 2, 3, 4, 5, 6 or 7; or EXTernal) can be selected at one time. Enabling a different trigger source will automatically disable the active input. For example, if TTLTrg1 is the active input, and TTLTrg4 is enabled, TTLTrg1 will become disabled and TTLTrg4 will become the active input.

"Trig Out" Port Shared by Switchboxes: See the “OUTPut” on page 65 for more information.

Assigning EXTernal, TTLTrg*n*, and ECLTrg*n* Trigger Inputs: After using TRIGger:SOURce EXT|TTLT*n*|ECLT*n*, the selected trigger source remains assigned to the "SWITCH" driver until it is relinquished through use of the TRIG:SOUR BUS|HOLD command. While the trigger is in use by the "SWITCH" driver, no other drivers operating on the E1406 command module will have access to that particular trigger source.

Related Commands: ABORt, [ROUTe:]SCAN, OUTPut

***RST Condition:** TRIGger:SOURce IMMediate

Example **Scanning Using External Triggers**

This example uses external triggering (TRIG:SOUR EXT) to scan channels 00 through 03 of a single-module switchbox. The trigger source to advance the scan is the input to the "Trig In" on the E1406A command module. When INIT is executed, the scan is started and channel 00 is closed. Then, each trigger received at the "Trig In" port advances the scan to the next channel.

```
TRIG:SOUR  EXT                 ! Set trigger source to external.
SCAN  (@100:103)               ! Set channel list to be scanned.
INIT                           ! Start scan, close channel 100.
(trigger externally)           ! Advance channel list to next channel.
```

Example **Scanning Using Bus Triggers**

This example uses bus triggering (TRIG:SOUR BUS) to scan channels 100 through 103 of a single-module switchbox. The trigger source to advance the scan is the *TRG command (as set with TRIGger:SOURce BUS). When INIT is executed, the scan is started and channel 00 is closed. Then, each *TRG command advances the scan to the next channel.

```
TRIG:SOUR  BUS                 ! Set trigger source to BUS.
SCAN  (@100:103)               ! Set channel list to be scanned.
INIT                           ! Start scan, close channel 100.
loop statement                 ! Loop to scan all channels.
*TRG                           ! Advance channel list to next channel.
Increment loop                 ! Increment loop count.
```

TRIGger:SOURce?

TRIGger:SOURce? returns the current trigger source for the switchbox. Command returns: BUS, EXT, HOLD, IMM, ECLT0-1, or TTLT0-7 for sources BUS, EXTernal, HOLD, IMMediate, ECLTrg*n*, or TTLTrg*n*, respectively.

Example **Querying the Trigger Source**

This example sets external triggering and queries the trigger source. Since external triggering is set, TRIG:SOUR? returns "EXT".

```
TRIG:SOUR  EXT                 ! Set external trigger source.
TRIG:SOUR?                     ! Query trigger source.
```

SCPI Command Quick Reference

The following table summarizes the SCPI commands for the E8480A Module.

Command		Description
ABORt		Abort a scan in progress.
ARM	:COUNt *<number>* | MIN | MAX :COUNt? [MIN|MAX]	Multiple scans per INIT command. Query number of scans.
DIAGnostic	:EMERgency:CLEar *<card_num>* :EMERgency:STATus? *<card_num>* :EMERgency:TRIGger:STATe *<card_num>,<mode>* :EMERgency:TRIGger:STATe? *<card_num>,<mode>* :INTerrupt[:LINe] *<card_num>,<line_nu*m> :INTerrupt[:LINe]? *<card_num>* :INTerrupt:TIMer *<card_num>,<time>* :INTerrupt:TIMer? *<card_num>* :SCAN:DELay *<card_num>,<time>* :SCAN:DELay? *<card_num>* :TEST[:RELays]? :TEST:EEPRom? *<card_num>*	Clear the emergency state occurred on the specified module. Query to see whether an emergency trigger occurs or not. Enable/disable the "Emergency Reset" port. Query enable/disable setting for the "Emergency Reset" port. Set an interrupt line for the specified module. Query the interrupt line of the specified module. Set wait time after an open or close before interrupting. Query the interrupt timer. Set additional scan delay time. Query the scan delay time. Do diagnostic to find the specific error(s). Check the checksum of EEPROM on the specified module.
DISPlay	:MONitor:CARD *<card_num>* | AUTO :MONitor:CARD? :MONitor[:STATe] *<mode>* :MONitor[:STATe]?	Select a module in a switchbox to be monitored. Query which module is set by above command. Set the monitor state on or off. Query the monitor state setting.
INITiate	:CONTinuous ON | OFF :CONTinuous? [:IMMediate]	Enable/disable continuous scanning. Query continuous scan state. Start a scanning cycle.
OUTPut	:ECLTrg*n*[:STATe] ON | OFF | 1 | 0 :ECLTrg*n*[:STATe]? [:EXTernal][:STATe] ON | OFF | 1 | 0 [:EXTernal][:STATe]? :TTLTrg*n*[:STATe] ON |OFF | 1 | 0 :TTLTrg*n*[:STATe]?	Enable/disable the specified ECL trigger line pulse. Query the specified ECL trigger line state. Enable/disable the "Trig Out" port on the command module. Query the "Trig Out" port enable state. Enable/disable the specified TTL trigger line pulse. Query the specified TTL trigger line state.
[ROUTe:]	CLOSe *<channel _list>* CLOSe? *<channel _list>* OPEN *<channel_list>* OPEN? *<channel _list>* SCAN *<channel_list>*	Close channel(s). Query channel(s) closed. Open channel(s). Query channel(s) opened. Define channels for scanning.
STATus	:OPERation:CONDition? :OPERation:ENABle *<unmask>* :OPERation:ENABle? :OPERation[:EVENt]? :PRESet	Return the contents of the Operation Condition Register. Enable events in the Operation Event Register to be reported. Return the unmask value set by STAT:OPER:ENAB command. Return the contents of the Operation Event Register. Set Enable Register bits to 0.
SYSTem	:CDEScription? *<card_num>* :CPON *<card_num>* | ALL :CTYPe? *<card_num>* :ERRor? :VERSion?	Returns description of the module. Opens all channels on the specified module(s). Returns the module type. Returns error number/message in the error queue. Returns the version of the SCPI standard.
TRIGger	[:IMMediate] :SOURce *<source>* :SOURce?	Causes a trigger event to occur. Set trigger source to BUS, or EXT, or HOLD, or IMM, ECLTrg*n* or TTLTrg*n*. Query scan trigger source.

IEEE 488.2 Common Command Reference

The following table lists the IEEE 488.2 Common (*) Commands that apply to the E8480A module.

Command	Command Description
*CLS	Clears all status registers (see STATus:OPERation[:EVENt]?) and clears the error queue.
*ESE *<register value>*	Enable Standard Event.
*ESE?	Enable Standard Event Query.
*ESR?	Standard Event Status Register Query.
*IDN?	Instrument ID Query; returns identification string of the module.
*OPC	Operation Complete.
*OPC?	Operation Complete Query.
*RCL *<numeric state>*	Recalls the instrument state saved by *SAV. You must reconfigure the scan list.
*RST	Resets the module. Opens all channels and invalidates current channel list for scanning. Sets ARM:COUN 1, TRIG:SOUR IMM, and INIT:CONT OFF.
*SAV *<numeric state>*	Stores the instrument state but does not save the scan list.
*SRE *<register value>*	Service request enable, enables status register bits.
*SRE?	Service request enable query.
*STB?	Read status byte query.
*TRG	Triggers the module to advance the scan when scan is enabled and trigger source is TRIGger:SOURce BUS.
*TST?	Self-test. Executes an internal self-test and returns only the first error encountered. Does not return multiple errors. The following is a list of responses you can obtain where "cc" is the card number with the leading zero deleted. +0 if self test passes. +cc01 for firmware error. +cc02 for bus error (problem communicating with the module). +cc03 for incorrect ID information read back from the module's ID register. +cc05 for hardware and firmware have different values. Possibly a hardware fault or an outside entity is register programming the E8480A. +cc10 if an interrupt was expected but not received. +cc11 if the busy bit was not held for a sufficient amount of time.
*WAI	Wait to Complete.

Appendix A
E8480A Specifications

ITEMS		SPECIFICATIONS
GENERAL CHARACTERISTICS		
Module Size/Device Type:		C-Size 1-Slot, Register based, A16, slave only, P1 and P2 Connectors
Total Channels:		40 channels
Relays Type:		Non-latching Form A
Typical Relay Life:	No Load: At Rated Load (5 Vdc & 0.1A):	1×10^7 1×10^5
Power Requirements:	Peak Module Current: Dynamic Module Current:	3.5 A @ +5 V 0.1 A @ +5 V
Watts/slot:		86 W
Cooling/slot:		0.70 mm H_2O @ 6.9 Liter/sec for 10^oC rise
Operating Temperature:		0 - 55^oC
Operating Humidity:		65% RH, 0 - 40^oC
INPUT CHARACTERISTICS		
Maximum Voltage:		150 Vdc, 280 Vac_{rms}, 2500 Vpk
Maximum Current:	Non-inductive Per Channel: [a]	12 Adc @ 30 Vdc, or 12 Aac_{rms}
Maximum Power:	Per channel: Per Module:	360 W, or 3360 VA 2160 W, or 20160 VA
DC PERFORMANCE		
Initial Closed Channel Resistance:		0.1 Ω (typical)

Isolation resistance:
(between any two points)
	≤ (40^oC, 65% RH):
	≤ (25^oC, 40% RH):
		$> 10^8$ Ω
		$> 10^9$ Ω

Thermal Offset:	Per Channel:	< 10 µV
AC PERFORMANCE		
Capacitance:	Hi to Lo: Channel to Channel: Channel to Chassis:	< 200 pF < 200 pF < 200 pF
Bandwidth (-3 dB):		10 MHz
Crosstalk: **(Channel to Channel for Zl = Zs = 50Ω)**	< 10 KHz: < 100 KHz:	< -65 dB < -45 dB
Time to Close or Open a Channel:		15 ms (typical)

a. 8.0 Adc @ 35 Vdc; 3.5 Adc @ 40 Vdc; 1.5 Adc @ 50 Vdc; 0.8 Adc @ 70 Vdc; 0.3 Adc @ 150 Vdc.

Notes:

Appendix B
Register-Based Programming

About This Appendix

The Agilent E8480A High Power General Purpose (GP) Switch module is a register-based product which does not support the VXIbus word serial protocol. When a SCPI command is sent to the module, the instrument driver resident in the Agilent E1406A command module parses the command and programs the module at the register level.

Register-based programming is a series of reads and writes directly to the module registers. This increases throughput speed since it eliminates command parsing and allows the use of an embedded controller. Also, register programming provides an avenue for users to control a VXI module with an alternate VXI controller device and eliminate the need for using an E1406A command module.

This appendix contains the information you need for register-based programming. The contents include:

Register Addressing

Register addresses for register-based devices are located in the upper 25% of VXI A16 address space. Every VXI device (up to 256 devices) is allocated a 32 word (64 byte) block of addresses. Figure B-1 on page 88 shows the register address location within A16 as it might be mapped by an embedded controller. Figure B-2 on page 89 shows the location of A16 address space in the E1406A command module.

When you are reading from or writing to a register of the module, a hexadecimal or decimal register address needs to be specified. This address consists of a base address plus a register offset:

Register Address = Base Address + Register Offset

Base Address

The base address used in register-based programming depends on whether the A16 address space is outside or inside the E1406A command module.

A16 Address Space Outside the Command Module

When the E1406A command module is not part of your VXIbus system (Figure B-1), the module's base address is computed as:[1]

$$C000_h + (LADDR_h * 40_h)$$

***- or** (decimal)*

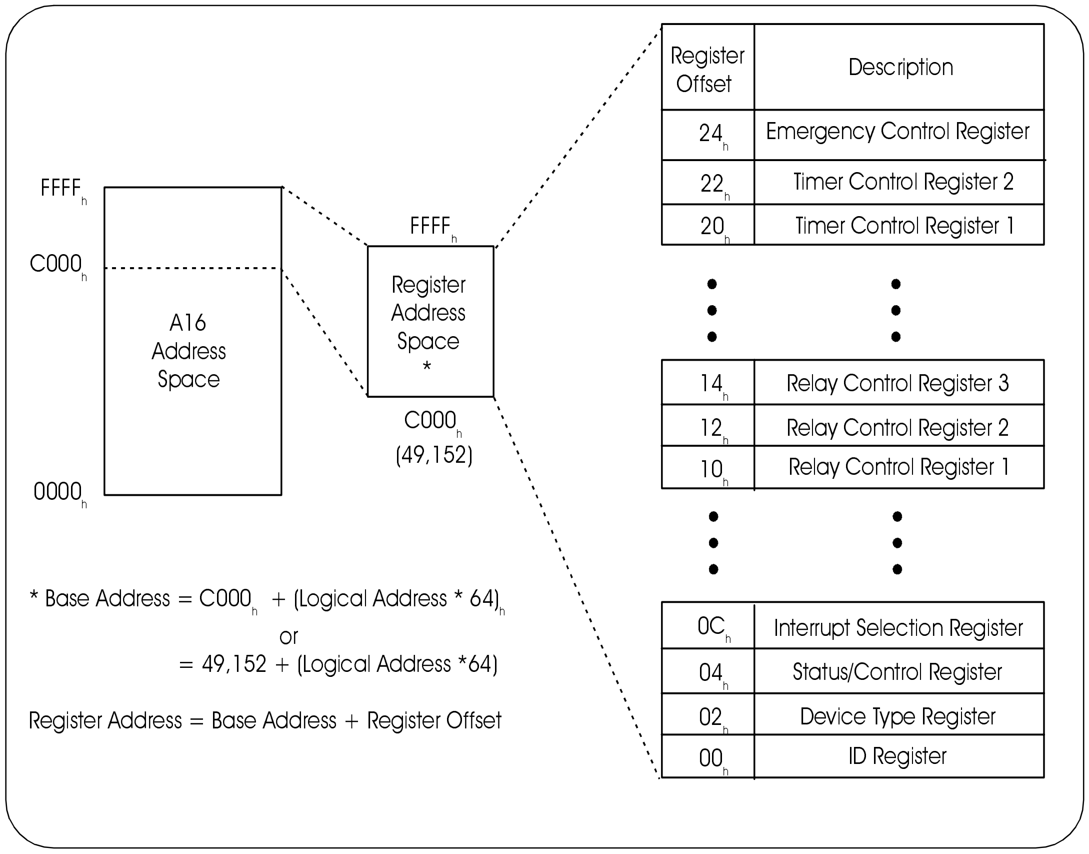
$$49{,}152 + (LADDR * 64)$$

where $C000_h$ (49,152) is the starting location of the VXI A16 addresses, LADDR is the module's logical address, and 64 (40_h) is the number of address bytes per register-based module. For example, the module's factory set logical address is 120 (78_h). If this address is not changed, the module will have a base address of:

$$C000_h + (78_h * 40_h) = C000_h + 1E00_h = DE00_h$$

***- or** (decimal)*

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



$FFFF_h$
$C000_h$
A16 Address Space
0000_h
$FFFF_h$
Register Address Space *
$C000_h$ (49,152)
* Base Address = $C000_h$ + (Logical Address * 64)$_h$
or
= 49,152 + (Logical Address *64)
Register Address = Base Address + Register Offset

Register Offset	Description
24_h	Emergency Control Register
22_h	Timer Control Register 2
20_h	Timer Control Register 1
• • •	• • •
14_h	Relay Control Register 3
12_h	Relay Control Register 2
10_h	Relay Control Register 1
• • •	• • •
$0C_h$	Interrupt Selection Register
04_h	Status/Control Register
02_h	Device Type Register
00_h	ID Register

Figure B-1. Registers within A16 Address Space

1. Numbers with a subscripted "h" are in hexadecimal format. Numbers without the subscripted "h" are in decimal format.

A16 Address Space Inside the Command Module

When the A16 address space is inside the Agilent E1406A command module (Figure B-2), the module's base address is computed as:[1]

$1FC000_h + (LADDR_h * 40_h)$

***- or** (decimal)*

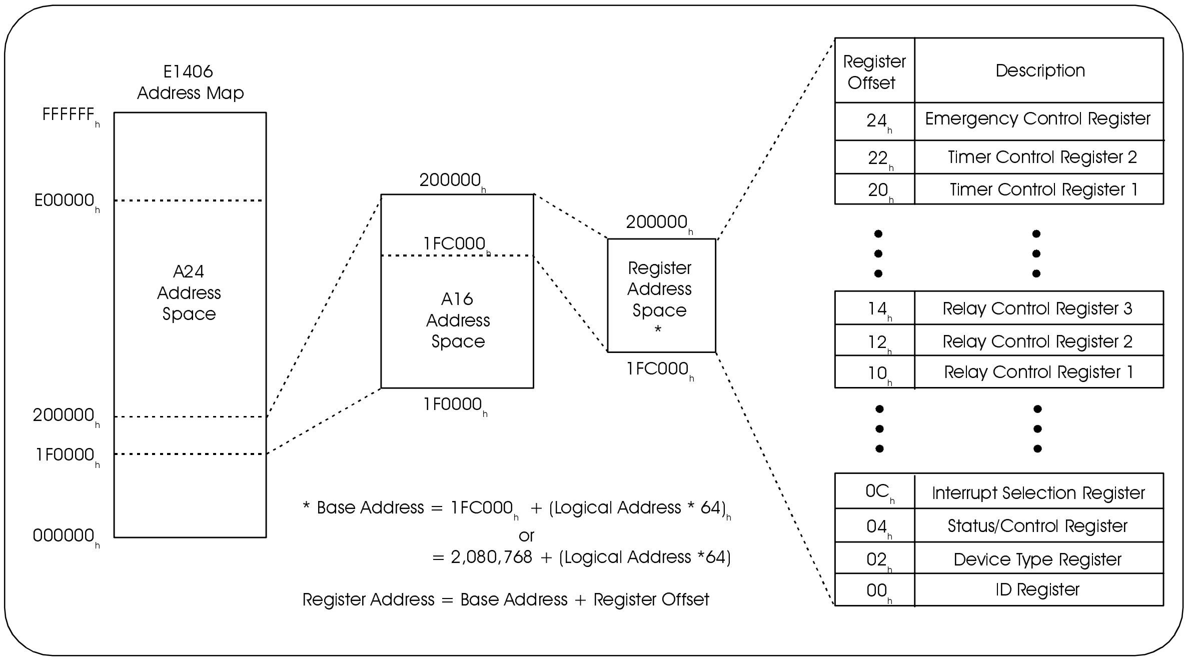
$2{,}080{,}768 + (LADDR * 64)$

where $1FC000_h$ (2,080,768) is the starting location of the register addresses, LADDR is the module's logical address, and 64 (40_h) is the number of address bytes per register-based device. Again, the module's factory set logical address is 120 (78_h). 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	Description
24_h	Emergency Control Register
22_h	Timer Control Register 2
20_h	Timer Control Register 1
• • •	• • •
14_h	Relay Control Register 3
12_h	Relay Control Register 2
10_h	Relay Control Register 1
• • •	• • •
$0C_h$	Interrupt Selection Register
04_h	Status/Control Register
02_h	Device Type Register
00_h	ID Register

Figure B-2. Registers within Command Module's A16 Address Space

1. Numbers with a subscripted "h" are in hexadecimal format. Numbers without the subscripted "h" are in decimal format.

Register Offset

The register offset is the register’s location in the block of 64 address bytes. 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
2,088,448 + 4 = 2,088,452

Registers Description

The E8480A module contains 10 registers as shown in Table B-1 on page 91. You can write to the writable (W) registers and read from the readable (R) registers. This section contains a description of the registers followed by a bit map of the registers in sequential address order.

NOTE *Undefined register bits (shown as "x" in the Tables) return as "1" when the register is read, and have no effect when written to.*

Table B-1. Module Registers

Registers	Addr. Offset	R/W	Register Address
Manufacturer ID Register	00_h	R	base + 00_h
Device Type Register	02_h	R	base + 02_h
Status/Control Register	04_h	R/W	base + 04_h
Interrupt Selection Register	$0C_h$	R/W	base + $0C_h$
Relay Control Register 1 (for Channels 00-15)	10_h	R/W	base + 10_h
Relay Control Register 2 (for Channels 16-31)	12_h	R/W	base + 12_h
Relay Control Register 3 (for Channels 32-39)	14_h	R/W	base + 14_h
Timer Control Register 1	20_h	R/W	base + 20_h
Timer Control Register 2	22_h	R/W	base + 22_h
Emergency Control Register	24_h	R/W	base + 24_h

ID Register

The Manufacturer Identification Register is at offset address 00_h. Reading the register returns $FFFF_h$ indicating the manufacturer is Agilent Technologies and the module is an A16 register-based device.

base + 00_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	x															
Read	Manufacturer ID - returns $FFFF_h$ in Agilent Technologies A16 only register-based card															

Device Type Register

The Device Type Register is at offset address 02_h. Reading the register returns $02D0_h$ indicating that the device is an E8480A module.

base + 02_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	x															
Read	$02D0_h$															

Status/Control Register

The Status/Control Register is at offset address 04_h. It is used to control the module and inform the user of its status.

base + 04_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write [a]	x									IRQ E/D	x				S	R
Read [b]	x	MS	x						B	IRQ E/D	x		1	P	x	

a. Writing to the reserved bits ("x") will cause no action. We recommend writing "1" to these bits.

b. Reading from the reserved bits ("x") will return as "1". Do not rely on these value for card operation.

Reading the Status/Control Register

When reading the status/control register, the following bits are of importance:

- **Self-test Passed (bit 2) -** Used to inform the user of the self-test status. "1" in this field indicates the module has successfully passed its self-test, and "0" indicates that the module is either executing or has failed its self-test.

- **Interrupt Status (bit 6) -** Used to inform the user of the interrupt status. "0" indicates that the interrupt is enabled, and "1" indicates that the interrupt is disabled. The interrupt generated after a channel has been closed can be disabled.

- **Busy (bit 7) -** Used to inform the user of a busy condition. "0" indicates that the module is busy, and "1" indicates that the module is not busy. Each relay requires about 20 ms execution time during which time the module is busy.

- **Modid Select (bit 14) -** "0" in this bit indicates that the module is selected by a high state on the P2 MODID line, and "1" indicates it is not selected via the P2 MODID line.

As an example, if a read of the Status Register (base + 04_h) returns "FFBF (1111111110111111)", it indicates that the module is not busy (bit 7 = 1) and the interrupt is enabled (bit 6 = 0).

Writing to the Status/Control Register

When writing to the status/control register, the following bits are of importance:

- **Soft Reset (bit 0) -** Writing a "1" to this bit will force the module to reset (all channels open).

NOTE *When writing to the registers it is necessary to write "0" to bit 0 after the reset has been performed before any other commands can be programmed and executed. SCPI commands take care of this automatically.*

- **Sysfail Inhibit (bit 1) -** Writing a "1" to this bit will disable the module from driving the SYSFAIL line (all channels open). The Slot-0 module can detect the failed module via this line.

- **Interrupt Enable/Disable (bit 6) -** Writing a "1" to this bit will disable the module from sending an interrupt request (generated by operating relays). Writing a "0" to this bit will enable the module's interrupt capability.

NOTE *Typically, interrupts are only disabled to "peek-poke" a module. Refer to your command module's operating manual before disabling the interrupt.*

Interrupt Selection Register

The Interrupt Selection Register is at offset address $0C_h$. It is used to set the interrupt level of the module and inform the user of the current interrupt level of the module.

base + $0C_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	x													Interrupt Level		
Read	x													Interrupt Level		

- You can set the interrupt level of the module by writing to **Interrupt Level Bits (bits 0-2)** of the register. Writing bits 2-0 with 001, 010, 011, 100, 101, 110, or 111 will set the interrupt level to 1 through 7 which corresponds to the VXI backplane lines 1-7. The highest interrupt level is 7, and the lowest level is 1 (default value).

- Reading the register will return the current interrupt level of the module. The returned value 001, 010, 011, 100, 101, 110, or 111 in Bits 2-0 corresponds to interrupt level 1 through 7.

Relay Control Registers

There are three relay control registers used to control the 40 channel relays of the module. All these registers are readable/writable (R/W) registers.

-- Relay Control Register 1 for Channel 00-15
-- Relay Control Register 2 for Channel 16-31
-- Relay Control Register 3 for Channel 32-39

Relay Control Register for Channels 00 - 15 (base + 10_h)

base + 10_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write / Read	ch 15	ch 14	ch 13	ch 12	ch 11	ch 10	ch 09	ch 08	ch 07	ch 06	ch 05	ch 04	ch 03	ch 02	ch 01	ch 00

Relay Control Register for Channels 16 - 31 (base + 12_h)

base + 12_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write / Read	ch 31	ch 30	ch 29	ch 28	ch 27	ch 26	ch 25	ch 24	ch 23	ch 22	ch 21	ch 20	ch 19	ch 18	ch 17	ch 16

Relay Control Register for Channels 32 - 39 (base + 14_h)

base + 14_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write / Read	x								ch 39	ch 38	ch 37	ch 36	ch 35	ch 34	ch 33	ch 32

The numbers shown in the above register maps indicate the channel numbers of the module.

- Writing a "1" to one bit will close related channel, and writing a "0" will open the channel. For example, to close channel 02, you need to write a "1" to bit 2 of the Relay Control Register (base +10_h) to close channel 02 and all other bits are set to "0".

- Reading the channel bit indicates to get the state of the relay driver circuit only. It cannot detect a defective relay. A bit that is "1" represents the related channel relay is closed. A bit that is "0" indicates the related channel relay is open.

- When the module is powered on or reset, all the channel relays are open and when you read from these registers, all the bits are zero.

Timer Control Registers

Each relay on the E8480A module requires about 15 ms settling time during which time the module is busy. There are two registers used to provide a programmable timer for the relay settling time. They are:

-- Timer Control Register 1 (base + 20_h)

-- Timer Control Register 2 (base + 22_h)

Timer Control Register 1 (base + 20_h)

base + 20_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Set Time															
Read	Read Time															

Timer Control Register 2 (base + 22_h)

base + 22_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	x								Set Time							
Read									Read Time							

As shown in above table, totally 24-bits of the two Timer Control Registers can be used to preset the relay settling time. Since the system clock is 16 MHz, each step of the timer is 62.5 nanoseconds. The maximum programmable timer can be set to:

$$62.5 \text{ ns} * \text{FFFFFF}_h = 1.0486 \text{ second}$$

The settling time is calculated based on the following formula:

$$\text{Settling Time} = 62.5 \text{ ns} * (\text{FFFFFF}_h - xxyyyy_h)$$

where $yyyy_h$ is the value written to the Timer Control Register 1 and xx_h to the Timer Control Register 2.

For example, if you want to preset the relay settling time to 15 ms, you should write "$567F_h$" to Timer Control Register 1 (base + 20) and "FC_h" to Timer Control Register 2 (base + 22). Since:

$$xxyyyy_h = \text{FFFFFF}_h - (15 \text{ ms} / 62.5 \text{ ns})_h = \text{FC567F}_h$$

These two registers can also be read back. The returned value can be used to calculate the settling time set for the relays.

NOTE *For register-based programming, you must set the relay settling time each time the module is powered up. We highly recommend you set the time to 15 ms for the E8480A relays.*

Emergency Control Register

The Emergency Control Register is at offset address 24_h. It is used to enable or disable the "Emergency Reset" port on the module’s front panel to accept external emergency trigger signal. This register can also be read back.

base + 24_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	x															E/D
Read	x								Emer.	x						E/D

Writing to the Emergency Control Register

When writing to the Emergency Control register, only bit 0 of the register is of use:

- Writing a "1" to bit 0 will disable the "Emergency Reset" port on the front panel of the module. By default, it is disabled (bit 0 = 1).
- Writing a "0" to this bit will enable the "Emergency Reset" port on the front panel of the module. When enabled, the "Emergency Reset" port can accept a TTL low voltage or a +5V negative-going pulse to force the module to open all channel relays. Furthermore, all relays on the module can not be operated any more unless the current emergency state is cleared.
- Once the emergency occurs, you need to clear the emergency state to recover the operation on the module relays. This can be done by writing a "1", then a "0" to this bit.

Reading the Emergency Control Register

When reading the Emergency Control register, the following bits are of use:

- **Bit 0 (E/D) -** Used to inform the user of the state of the "Emergency Reset" port whether enabled or disabled. "1" in this field indicates the port is disabled, and "0" indicates the port is enabled.
- **Bit 6 (Emer.) -** Used to inform the user of the emergency status on the "Emergency Reset" port whether happened or not. "1" indicates an emergency happened, and "0" indicates no emergency happened.

Appendix C
Error Messages

Table C-1 lists the error messages associated with the E8480A High Power General Purpose Switch Module when programmed with SCPI commands. See the appropriate mainframe manual for a complete list of error messages.

Table C-1. Error Messages

Number	Error Message	Potential Cause(s)
-211	Trigger ignored	Trigger received when scan not enabled. Trigger received after scan complete. Trigger too fast.
-213	INIT Ignored	Attempting to execute an INIT command when a scan is already in progress.
-224	Illegal parameter value	Attempting to execute a command with a parameter not applicable to the command.
-240	Hardware error	Command failed due to a hardware problem.
-310	System error	Too many characters in the channel list expression.
1500	External trigger source already allocated	Assigning an external trigger source to a switchbox when the trigger source has already been assigned to another switchbox.
2000	Invalid card number	Addressing a module (card) in a switchbox that is not part of the switchbox.
2001	Invalid channel number	Attempting to address a channel of a module in a switchbox that is not supported by the module (e.g., channel 99 of a module).
2006	Command not supported on this card	Sending a command to a module (card) in a switchbox that is unsupported by the module.
2008	Scan list not initialized	Executing an INIT command without a channel list defined.
2009	Too many channels in channel list	Attempting to address more channels than available in the switchbox.
2011	Empty channel list	Channel lists contains no valid channels.
2012	Invalid Channel Range	Invalid channel(s) specified in SCAN <*channel_list*> command. Attempting to begin scanning when no valid channel list is defined.
2600	Function not supported on this card	Sending a command to a module (card) in a switchbox that is not supported by the module or switchbox.
2601	Missing parameter	Sending a command requiring a *channel_list* without the *channel_list*.

Notes:

Appendix D
Relay Life

Relay Life

Electromechanical relays are subject to normal wear-out. Relay life depends on several factors. The effects of loading and switching frequency are briefly discussed below:

Relay Load. In general, higher power switching reduces relay life. In addition, capacitive/inductive loads and high inrush currents (for example, turning on a lamp or starting a motor) reduces relay life. *Exceeding specified maximum inputs can cause catastrophic failure.*

Switching Frequency. Relay contacts heat up when switched. As the switching frequency increases, the contacts have less time to dissipate heat. The resulting increase in contact temperature also reduces relay life.

End-of-Life Detection

A preventive maintenance routine can prevent problems caused by unexpected relay failure. The end of the life of the relay can be determined by using one or more of the three methods described below. The best method (or combination of methods), as well as the failure criteria, depends on the application in which the relay is used.

Contact Resistance. As the relay begins to wear out, its contact resistance increases. When the resistance exceeds a predetermined value, the relay should be replaced.

Stability of Contact Resistance. The stability of the contact resistance decreases with age. Using this method, the contact resistance is measured several (5-10) times, and the variance of the measurements is determined. An increase in the variance indicates deteriorating performance.

Number of Operations. Relays can be replaced after a predetermined number of contact closures. However, this method requires knowledge of the applied load and life specifications for the applied load.

Notes:

Notes:

Agilent Technologies

Manual Part Number: E8480-90001
Printed in U.S.A. E0301