Agilent E2290A 16-Bit Digital I/O

User's Manual and SCPI Programming Guide

Where to Find it - Online and Printed Information:

System installation (hardware/software)	VXIbus Configuration Guide*
	Agilent VIC*
Module configuration and wiring	This Manual
Register-Based Programming	This Manual
VXI*plug&play* programming	VXI*plug&play* Online Help
VXI*plug&play* example programs	VXI*plug&play* Online Help
VXI*plug&play* function reference	VXI*plug&play* Online Help
Soft Front Panel information	VXI*plug&play* Online Help
VISA language information	Agilent VISA User's Guide
Agilent VEE programming information	Agilent VEE User's Manual

**Supplied with Agilent Command Modules, Embedded Controllers, and VXLink.*

Agilent Technologies

Manual Part Number: E2290-90000
Printed in Malaysia E0806

Certification

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

AGILENT TECHNOLOGIES WARRANTY STATEMENT

PRODUCT: E2290A **DURATION OF WARRANTY:** 1 year

1. Agilent warrants Agilent hardware, accessories and supplies against defects in materials and workmanship for the period specified above (one year). If Aglent 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 Technologies, (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.

IEC Measurement Category II Overvoltage Protection

This is a measurement Category II product designed for measurements at voltages up to 300V from earth, including measurements of voltages at typical mains socket outlets. The product should not be used to make voltage measurements on a fixed electrical installation including building wiring, circuit breakers, or service panels.

E2290A 16-Ch Digital I/O M-Module User's Manual and Programming Guide

Agilent Technologies

Edition 1 Rev 2

Documentation History

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

Edition 1 . . . . . .June 1997

Edition 1 Rev 2 . . . . . .August 2006

Trademarks

Microsoft® is a U.S. registered trademark of Microsoft Corporation

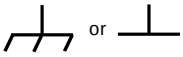
Windows NT® is a U.S. registered trademark of Microsoft Corporation

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation

Safety Symbols

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

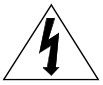
Indicates the field wiring terminal that must be connected to earth ground before operating the equipment—protects against electrical shock in case of fault.



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

Alternating current (AC)

Direct current (DC).



Indicates hazardous voltages.

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

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

WARNINGS

The following general safety precautions must be observed during all phases of operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the product. Agilent Technologies, Inc. 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 an Agilent Technologies Sales and Service Office for service and repair to ensure that safety features are maintained.

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

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

DECLARATION OF CONFORMITY

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

Manufacturer's Name: Agilent Technologies, Incorporated
Manufacturer's Address: *Measurement Product Generation Unit*
815 14th ST. S.W.
Loveland, CO 80537 USA

Declares, that the product

Product Name: 16 Bit Digital I/O M Module
Model Number: E2290A
Product Options: *This declaration covers all options of the above product(s).*

Conforms with the following European Directives:

The product herewith complies with the requirements of the Low Voltage Directive 73/23/EEC and the EMC Directive 89/336/EEC 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 55011:1998	*Group 1 Class A [1]*
	IEC 61000-4-2:1995+A1:1998 / EN 61000-4-2:1995	*4kV CD, 8kV AD*
	IEC 61000-4-3:1995 / EN 61000-4-3:1995	*3 V/m, 80-1000 MHz*
	IEC 61000-4-4:1995 / EN 61000-4-4:1995	*0.5kV signal lines, 1kV power lines*
	IEC 61000-4-5:1995 / EN 61000-4-5:1995	*0.5 kV line-line, 1 kV line-ground*
	IEC 61000-4-6:1996 / EN 61000-4-6:1996	*3V, 0.15-80 MHz*
	IEC 61000-4-11:1994 / EN 61000-4-11:1994	*I 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 3111-1:1994	
	IEC 950(1991) + A1(1992) + A2(1993) + A3(1994)	
	EN 60950(1992) + A1(1992) + A2(1993) + A3(1994)	
	CSA C22.2#950(1995)	
	UL 1950(1995)	

Supplemental Information:

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

September 5, 2000
Date

Jim White
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 Strabe 130, D 71034 Böblingen, Germany

Notes:

Notes:

Notes:

Chapter 1
Getting Started

What's in this Manual?

This manual contains a module description, configuration and wiring information, register maps, and specifications for the Agilent E2290A 16-Bit Digital I/O M-Module.

The Agilent E2290A is intended to be installed on an M-Module Carrier. When it is necessary to reference a particular carrier, the Agilent E2251 C-Size VXIbus M-Module Carrier will be used.

Module Description

The Agilent E2290A is a digital input/output M-Module containing 16 data/actuator lines. These data lines offer TTL compatible inputs and open-drain outputs up to 30 volts. Each I/O line also provides active pull-up and pull-down for actuating external devices such as:

- Relays and switches
- High frequency coax relays or microwave switches
- Programmable attenuators
- Optical Isolators

Agilent E2290A Features

- Sixteen Digital Inputs or Outputs. Data that is written (output) may also be read back. Bits may be addressed as two 8-bit ports (Port 0 and Port 1), one 16-bit port (Port 0), or individual bits can be accessed.
- Support of M-Module Specification Type C Interrupts for pattern matching. An interrupt occurs (if enabled) when the incoming data matches the pattern stored in the Compare Data Register.
- Debounce capability for data lines used as inputs. The debounce circuitry allows incoming data to be debounced for a period of 3 mSec before being latched into the Input Register. You can enable/disable debounce for the Input Register, however, debounce is always active for the input to the pattern match interrupt circuit.
- TTL compatible levels or open-drain outputs. The 16 data lines provide for TTL compatible I/O (open-drain outputs up to 30 Volts, requires external pull-up).
- Simplified Standard Commands for Programmable Instruments (SCPI) commands.
- There are no I/O handshaking lines/modes.

Output Circuitry

Figure 1-1 shows a simplified schematic of one bit's output circuitry and two example applications. For the SCPI commands, sourcing a "1" means that the output is driven high (2.9 Vdc or greater, sourcing up to 20 mA); sourcing a "0" means the output is driven low (0.4 Vdc or less, sinking up to 200mA).

Caution **Do not exceed the 30Vdc external voltage; doing so may damage the module.**

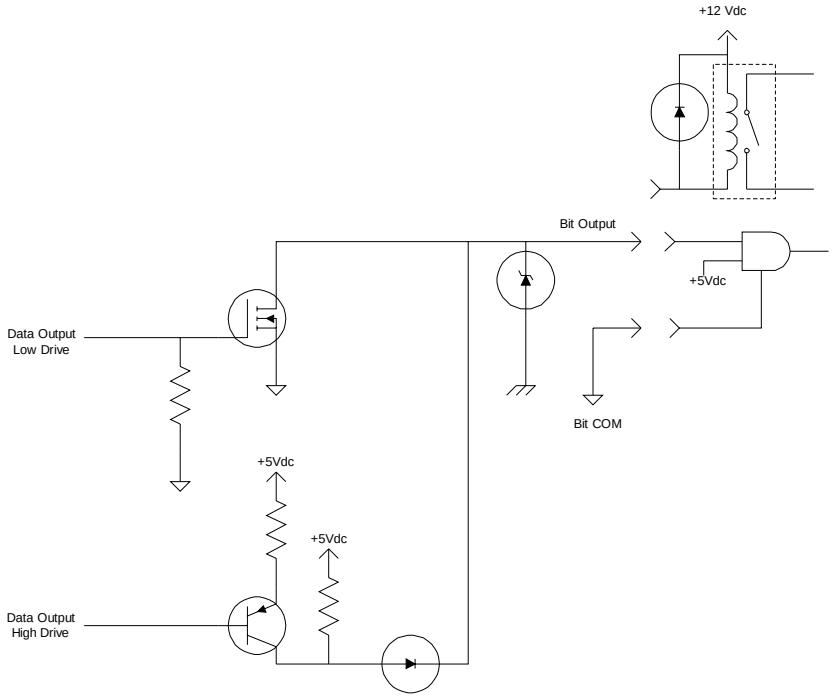


Figure 1-1. Simplified Digital Output Circuit

Input Circuitry

Figure 1-2 shows a simplified schematic of one bit's input circuitry and two example applications. Because the input bit has its own pull-up resistor, the bit can be used to detect switch closures to ground. Using external pull-ups, it can also be used with digital logic "high" levels of up to 30 Vdc. When used as digital inputs, a "HIGH" or "1" means a positive voltage (>1.8 Vdc) is present and "LOW" or "0" means a voltage of <0.8 Vdc is present.

Caution **Do not exceed the 30Vdc external voltage; doing so may damage the module.**

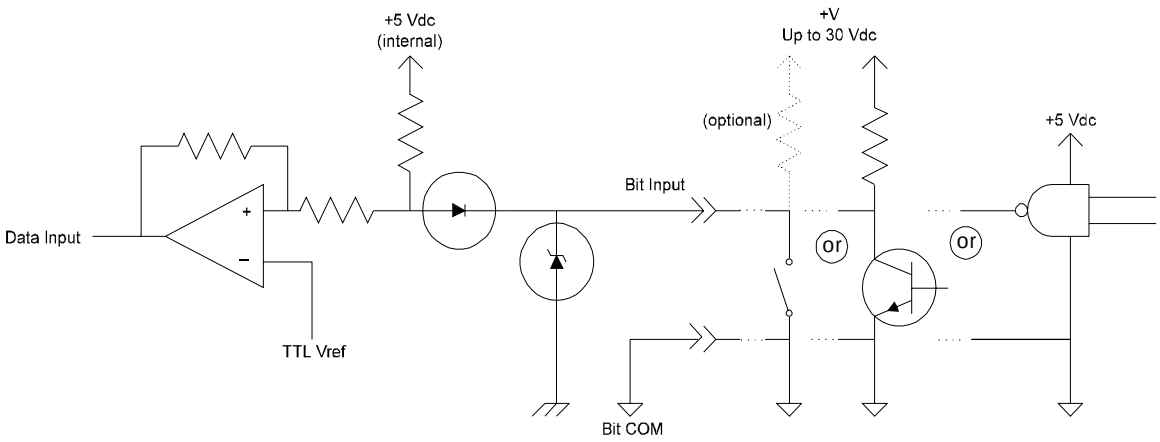


Figure 1-2. Simplified Digital Input Circuit

Wiring and Configuration

This section describes how to connect user wiring to Agilent E2290A.

Note The procedures in this section assume the M-Module(s) have already been installed into an M-Module Carrier. Since installation is dependent on the carrier used, instructions for installing M-Modules into the carrier are not included here. Refer to your M-Module carrier documentation for installation instructions.

WARNING **SHOCK HAZARD. Only service-trained personnel who are aware of the hazards involved should install, remove, or configure the modules. Before installing or removing any module or carrier, disconnect power from the mainframe and user wiring.**

Caution **MAXIMUM VOLTAGE/CURRENT. Maximum voltage that may be applied (any I/O line to chassis) to the Agilent E2290A is 30 VDC.**

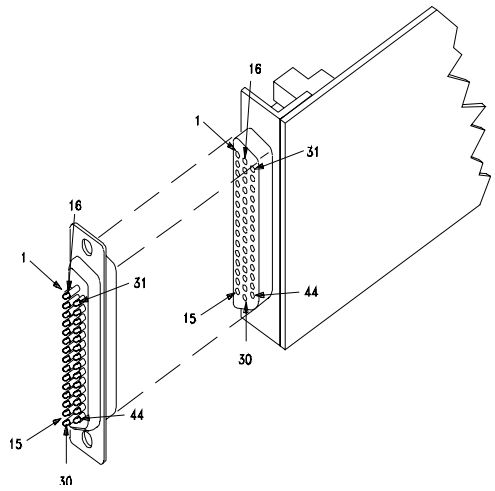
Caution **STATIC ELECTRICITY. Static electricity is a major cause of component failure. To prevent damage to the electrical components on an M-Module or the carrier, observe anti-static techniques whenever installing, removing, or working on a carrier or M-Module.**

Wiring to the Supplied Mating Connector

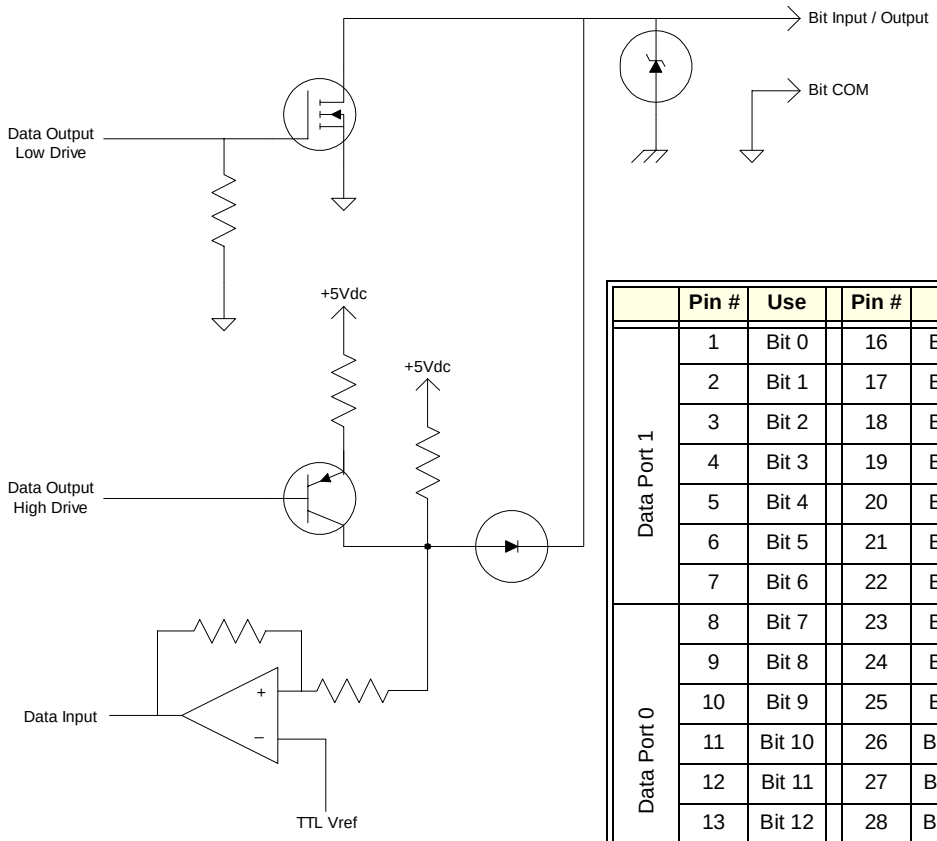
Figure 1-3 shows the 44-pin user connector, its pinout, and a simplified schematic for the input/output circuitry. Figure 1-4 shows how to wire and assemble the supplied connector and hood.

Note Do not use the supplied hood if the Agilent E2290A is installed in one of the two internal slots (M4 or M5) of the Agilent E2251A M-Module Carrier.

Agilent E2290A Digital I/O Wiring Information



CAUTION The maximum voltage is 30 VDC (any I/O line to chassis or common).



	Pin #	Use	Pin #	Use	Pin #	Use
Data Port 1	1	Bit 0	16	Bit 0 COM	31	Chassis
	2	Bit 1	17	Bit 1 COM	32	Chassis
	3	Bit 2	18	Bit 2 COM	33	Chassis
	4	Bit 3	19	Bit 3 COM	34	Chassis
	5	Bit 4	20	Bit 4 COM	35	Chassis
	6	Bit 5	21	Bit 5 COM	36	Chassis
	7	Bit 6	22	Bit 6 COM	37	Chassis
Data Port 0	8	Bit 7	23	Bit 7 COM	38	Chassis
	9	Bit 8	24	Bit 8 COM	39	Chassis
	10	Bit 9	25	Bit 9 COM	40	Chassis
	11	Bit 10	26	Bit 10 COM	41	Chassis
	12	Bit 11	27	Bit 11 COM	42	Chassis
	13	Bit 12	28	Bit 12 COM	43	Bit 15
	14	Bit 13	29	Bit 13 COM	44	Bit 15 COM
	15	Bit 14	30	Bit 14 COM		

Figure 1-3. Agilent E2290A User Connector and I/O Schematic

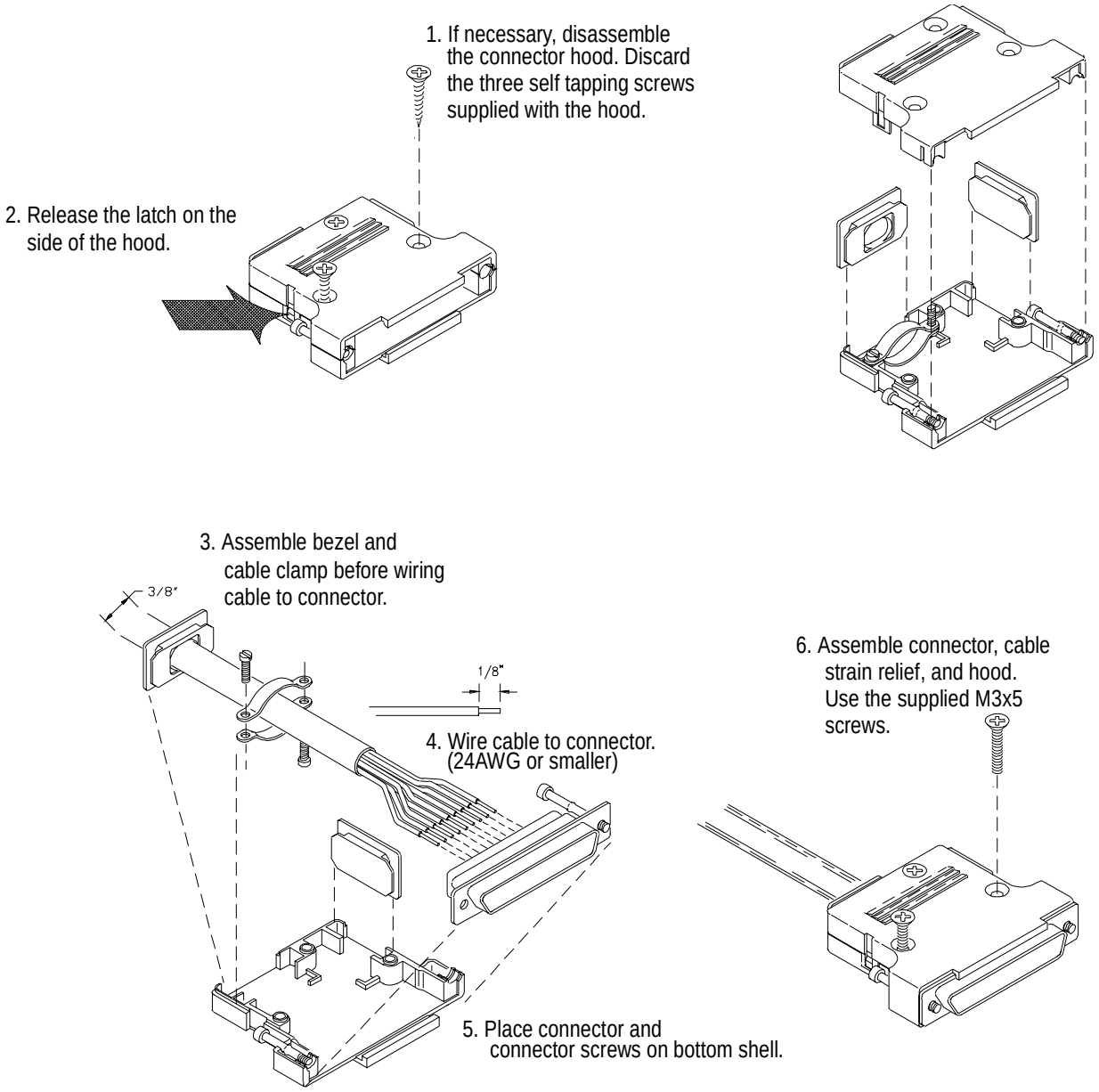


Figure 1-4. Assembling the Connector and Hood

Note Do not use the supplied hood if the Agilent E2290A is installed in one of the two internal slots (M4 or M5) of the Agilent E2251A M-Module Carrier.

Chapter 2
SCPI Programming

General Information

This chapter describes how to program the Agilent E2290A Digital I/O M-Module with the Standard Commands for Programmable Instruments (SCPI).

Note Do not do register writes if you are controlling the module by a high level driver such as SCPI or VXI*plug&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.

The following example programs were developed with the ANSI C language using the Agilent VISA extensions. The programs were written and tested in Microsoft® Visual C++ but should compile under any standard ANSI C compiler.

To run the programs you must have the Agilent SICL Library, the Agilent VISA extensions, and an Agilent 82340 or 82341 GPIB module installed and properly configured in your PC. An Agilent E1406 Command Module is required.

Example 1: Reset, Self Test, and Module ID

The following example reads the module ID string, performs module self-test, and displays the results.

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

    /* Interface address is 9, M-Module secondary address is 3*/
#define INSTR_ADDR "GPIB0::9::3::INSTR"

int main()
{
   ViStatus errStatus;                  /*Status from each VISA call*/
   ViSession viRM;                      /*Resource mgr. session */
   ViSession m_mod;                     /* M-module session  */
   char id_string[256];                 /*ID string*/
   char selftst_string[256];            /*self-test string*/
```

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

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

  /* Reset the M-Module */
errStatus = viPrintf(m_mod, "*RST\n");
if(VI_SUCCESS > errStatus){
    printf("ERROR: viviPrintf() returned 0x%x\n",errStatus);
    return errStatus;}

  /* Perform M-Module Self-Test */
errStatus = viQueryf(m_mod,"DIAG:TEST?\n","%t",selftst_string);
if (VI_SUCCESS > errStatus) {
    printf("ERROR: viPrintf() returned 0x%x\n",errStatus);
    return errStatus;}
printf("Self Test Result is %s\n",selftst_string);

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

  /* Close the M_Module Instrument Session */
errStatus = viClose (m_mod);
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 2: Source and Sense (Output and Input)

The following example program resets an Agilent E2290A 16-Bit Digital I/O M-Module. It demonstrates how to source (output) and sense (input) data from the M-Module. With a simple loop-back fixture (connects Port 0 bit 0 to Port 1 bit 0, Port 0 bit 1 to Port 1 bit 1, Port 0 bit 2 to Port 1 bit 2, etc.), this program can be used as a simple operational verification test. By writing a pattern to Port 0 and reading the pattern from Port 1 and then writing a pattern to Port 1 and reading the pattern from Port 0 both ports can be verified for read and write operations.

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

#define INSTR_ADDR "GPIB0::9::3::INSTR"  /* M-Module logical address */

int main()
{
    ViStatus errStatus;                  /*Status from each VISA call*/
    ViSession viRM;                      /*Resource mgr. session */
    ViSession E2290                      /* M-module session  */
    inst value;                          /* data read from port */

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

/* Open the M-Module instrument session  */
    errStatus = viOpen(viRM,INSTR_ADDR, VI_NULL,VI_NULL,&E2290);
    if (VI_SUCCESS > errStatus){
        printf ("ERROR: viOpen() returned 0x%x\n",errStatus);
        return errStatus;}

/* Reset the M-Module */
    errStatus = viPrintf(E2290, "*RST\n");
    if (VI_SUCCESS > errStatus){
        printf ("ERROR: viPrintf() returned 0x%x\n",errStatus);
        return errStatus;}

/* Write a value to Port 0 */
    errStatus = viPrintf(E2290,"SOUR:DIG:DATA0 0xAA\n");
        if (VI_SUCCESS > errStatus) {
        printf ("ERROR: viPrintf() returned 0x%x\n",errStatus);
        return errStatus;}

/* Read the value from Port 1 */
    errStatus = viQueryf(E2290,"MEAS:DIG:DATA1?\n","%i",value);
        if (VI_SUCCESS > errStatus) {
        printf ("ERROR: viPrintf() returned 0x%x\n",errStatus);
        return errStatus;}
printf (Value read from Port 1 is: %i\n",value);
```

```
/* Write a value to Port 1 */
    errStatus = viPrintf(E2290,"SOUR:DIG:DATA1 0x55\n");
        if (VI_SUCCESS > errStatus) {
        printf ("ERROR: viPrintf() returned 0x%x\n",errStatus);
        return errStatus;}

/* Read the value from Port 0 */
    errStatus = viQueryf(E2290,"MEAS:DIG:DATA0?\n","%i",value);
        if (VI_SUCCESS > errStatus) {
        printf ("ERROR: viPrintf() returned 0x%x\n",errStatus);
        return errStatus;}
printf ("Value returned from Port 0 is: %i\n",value);

/* Close the M_Module Instrument Session */
    errStatus = viClose (E2290);
        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 3: Pattern Match

You can set-up the Status Subsystem and have the Agilent E2290A Module interrupt (via SRQ) the system computer when a pattern match occurs. In general, you must enable bit 8 in the Status Operation Enable register; use the STATus:OPERation:ENABle 256 command. Enable the OPR bit (bit 7) in the Status Register with the *SRE 128 command; this allows the Operation Status register to generate the SRQ.

This program interrupts the system computer via SRQ when a pattern match occurs. The program defines a pattern for matching, enables the system to respond to SRQ interrupts, and then programs the Agilent E2290A Status Subsystem to generate an SRQ. The program then waits for the pattern match to occur. When the match occurs, the program disables further interrupts and clears the Agilent E2290A.

```
/* Pattern Match Interrupt Example
This program sets a pattern to match, enables service request event,
then waits until the match occurs. Created in Microsoft Visual C++   */


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

#define INSTR_ADDR "GPIB0::9::3"          /* E2290A logical address */

void main()
{
ViStatus errStatus;                       /* status from VISA call */
ViSession viRM;                           /* Resource Mgr. session */
viSession E2290;                          /* session for E2290A */
int val, event;

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

/* Open the Instrument Session */
errStatus = viOpen (viRM, INSTR_ADDR,VI_NULL,VI_NULL, &E2290);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viOpen() returned 0x%x\n",errStatus);
    return errStatus;}


/* Define Pattern for Matching */
errStatus = viPrintf (E2290, "SENS:EVEN:PATT %d\n",0xAAAA);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n",errStatus);
    return errStatus;}

/* Set Pattern Mask */
errStatus = viPrintf (E2290, "SENS:EVEN:ENAB %d\n",0xFFFF);
    if (VI_SUCCESS > errStatus){
```

```
    printf("ERROR: viPrintf() returned 0x%x\n",errStatus);
    return errStatus;}

/* Enable SRQ Event */
errStatus =
viEnableEvent(E2290,VI_EVENT_SERVICE_REQ,VI_QUEUE,VI_NULL);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viEnableEvent() returned 0x%x\n",errStatus);
    return errStatus;}

errStatus = viPrintf (E2290, "STAT:OPER:ENAB 256\n");
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n",errStatus);
    return errStatus;}

errStatus = viPrintf (E2290, "*SRE 128\n");
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n",errStatus);
    return errStatus;}

/* Wait for DAV event to occur */
printf("Waiting for a DAV Event to occur."\n);
errStatus =
viWaitOnEvent(E2290,VI_EVENT_SERVICE_REQ,VI_TMO_INFINITE,VI_
NULL);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viWaitOnEvent() returned 0x%x\n",errStatus);
    return errStatus;}

printf("Pattern Match Occurred."\n);

/* Disable the event */
errStatus = viDisableEvent(E2290,VI_EVENT_SERVICE_REQ,VIQUEUE);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viDisableEvent returned 0x%x\n",errStatus);
    return errStatus;}

/* Clear the E2290A */
errStatus = viPrintf (E2290, "*CLS\n");
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viPrintf() returned 0x%x\n",errStatus);
    return errStatus;}

/* Close Sessions */
errStatus = viClose (E2290);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viClose() returned 0x%x\n",errStatus);
    return 0;}

errStatus = viClose (viRM);
    if (VI_SUCCESS > errStatus){
    printf("ERROR: viClose() returned 0x%x\n",errStatus);
    return 0;}

}                                   /* End of main program */
```

Chapter 3
SCPI Command Reference

General Information

This chapter describes the Standard Commands for Programmable Instruments (SCPI) command set for the the Agilent E2290A Digital I/O Module. For examples of programming the module with SCPI commands, refer to Chapter 2.

Note Do not do register writes if you are controlling the module by a high level driver such as SCPI or the VXI*plug&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.

Command Fundamentals

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 the asterisk character (*), and may include one or more parameters. The command keyword is separated from the first parameter by a space character. Some examples of common commands are shown below:

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

SCPI Command Format

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

```
[SOURce:]
    DIGital:DATAn[:BYTE][:VALue]<data>
    DIGital:DATAn[:BYTE]:BITm 0 | 1
```

[SOURce:] is the root command, DIGital is a second level command, DATA is a third level command, [:BYTe] is a fourth level command (the square brackets [] indicate the command is optional). <*data*>, *n*, *m*, 0, and 1 are parameters..

Command Separator

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

```
SOURce:DIGital:DATAn:BYTE:VALUE<data>
```

Abbreviated Commands

The command syntax shows most commands as a mixture of upper and lower case letters. The upper case letters indicate the abbreviated spelling for the command. For shorter program lines, send the abbreviated form. For better program readability, you may send the entire command. The instrument will accept either the abbreviated form or the entire command. For example, if the command syntax shows `MEASure`, then `MEAS` and `MEASURE` are both acceptable forms. Other forms of `MEASure`, such as `MEASU` or `MEASUR` will generate an error. You may use upper or lower case letters. Therefore, `MEASURE`, `measure`, and `MeAsUrE` are all acceptable.

Implied Commands

Implied commands are those which appear in square brackets ([]) in the command syntax. (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 [SOUR:] subsystem shown below:

```
[SOURce:]
    DIGital:DATAn[:BYTE][:VALue]<data>
    DIGital:DATAn[:BYTE]:BITm 0 | 1
```

The root command [SOURce:] and the fourth level [:BYTE] are implied commands. To output the value 55 to Port 0, you can send either of the following command statements:

```
    SOUR:DIG:DATA0:BYTE:VALUE 55
```

or

```
    DIG:DATA0 55
```

Command Parameters

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 chooses a default value. 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;DIG:DATA0 55
```

or

```
DIG:DATA0 55;*SAV 2
```

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

```
DIG:DATA0 55;:DIG:DATA0?
```

The DIAGnostic SUbsystem provides a self test for the E2290A M-Module.

Syntax

```
DIAGnostic
        :INTerrupt[:LINE] <intr_line>
        :INTerrupt[:LINE]?
        :TEST?
```

DIAGnostic:INTerrupt[:LINe] <*intr_line*>

DIAGnostic:INTerrupt[:LINe] <*intr_line*> sets the VXIbus interrupt line the M-Module will use. The M-Module generates an interrupt after a channel closing or opening completes. Refer to your command module documentation for more information.

Note The VXIbus Interrupt Line is controlled by the VXIbus M-Module Carrier NOT by the M-Module. DIAGnostic:INTerrupt:LINE reprograms the Agilent E2251 M-Module Carrier. It will work properly only if the M-Module is installed in an Agilent E2251 M-Module Carrier.

Parameter

Parameter Name	Parameter Type	Range of Values	Default Value
<intr_line>	int16	0 through 7	1

Comments

- Normally, the VXIbus interrupt line does not need to be modified. Only one value (1 through 7) can be set at one time.
- The default value of <*number*> is 1 (lowest interrupt level).
- Setting the interrupt line to 0 disables the interrupt.

Reset Condition *RST does not affect the interrupt line.

Related Commands DIAGnostic:INTerrupt:LINE?

DIAGnostic:INTerrupt[:LINe]?

DIAGnostic:INTerrupt[:LINe]? returns the VXIbus interrupt line being used be the module.

Returned Data

Type	Range	Default
int16	0 through 7	1

Note

The VXIbus Interrupt Line is controlled by the VXIbus M-Module Carrier NOT by the M-Module. DIAGnostic:INTerrupt:LINE reprograms the Agilent E2251 M-Module Carrier. It will work properly only if the M-Module is installed in an Agilent E2251 M-Module Carrier.

Reset Condition

*RST does not affect the interrupt line.

Related Commands

DIAGnostic:INTerrupt:LINE

DIAGnostic:TEST?

DIAGnostic:TEST? performs a extensive self test on the module that will drive each bit HI and LO.

This command performs tests in the following order:

- Verify that the Status Register indicates the module is ready.
- Read/write test of the debounce/interrupt bits in the Status Register.
- Read/write test of the Pattern and Pattern Enable Registers.
- Test interrupts.
- Read/write of Output Enable Register (all bits 1 then all bits 0).
- Read/write of Output Register (all bits 1 then all bits 0).
- Read/write of a sliding bit through the Output Register.

DIAGnostic:TEST? returns a a number and a string indicating what failure (if any) occurred. The first four tests in the list are also performed by the *TST? command.

Caution

The extended self-test will drive HI and LO each bit on the module. Before performing the test, make certain that external devices will not be affected by these actions. It is recommended that external devices be disconnected from the module while executing DAIGnostic:TEST?.

Returned Data

Type	Description of Numerical Response	Possible Strings Returned
int16, string	0 = self-test passsed	“Self Test passed”
	1 = ERROR: status register	“init or full bit wrong. Expect X got X”
	2 = ERROR: register readback	“Readback reg X failed, expect 0 got X”
	3: ERROR: interrupt	“Interrupt failed VISA error X”

Comments

- A query response of 1 indicates that the module is operating properly, a non-zero response measn an error occurred.
- The extended self test does not measure the actual output state to ensure it is high or low, it only queries the state of the Control Register circuitry. It may be possible to pass DIAGnostic:TEST? (return a 0) and still have output/input circuitry failures.

Reset Condition

*RST does not affect this query.

Related Commands

*TST?, DIAGnostic:INTeruppt:LINE

The DISPlay:MONitor Subsystem turns on the monitor mode. Parameters related to the state of the data are shown on an external terminal[1]. Refer to the Command Module's User's Guide for supported terminal types. The DISPlay:MONitor commands do not apply to any C-SCPI or VXI*plug&play* driver implementation.

Syntax

```
DISPlay:MONitor[:STATe] 0 | 1 | ON | OFF
:MONitor[:STATe]?
```

DISPlay:MONitor[:STATe] 0 | 1 | ON | OFF

Turns the Display Mode on or off.

Parameters None

Comments

- DISP:MON ON or DISP:MON 1 enables the terminal display of Port parameters. The parameters are updated to the terminal following each new command accessing a Port.
- A keyboard entry at the terminal sets DISP:MON OFF.
- ***RST Condition:** OFF.

Example

```
DISP:MON ON
```

turns the display mode on.

DISPlay:MONitor[:STATe]?

Returns the value of the Display Monitor State as either 0 (for OFF) or 1 (for ON).

Parameters None

Comments

- DISP:MON[:STAT]? returns a 1 if the monitor mode is on; or returns a 0 if the monitor mode is off.

Example

```
DISP:MON?
```

1.The display monitor is an RS-232 Terminal attached to an Agilent E1405B or E1406A/B Command Module and provides an interactive user interface to the Agilent E2290A.

The INPut:DEBounce Subsystem configures the input debounce circuitry The debounce circuitry allows the incoming data to be debounced for approximately 3mSec before being latched into the Input Register. You can enable/disable the debounce for the Input Register, however, debounce is always active for the pattern match interrupt.

Syntax

```
INPut:DEBounce[:STATe] 0 | 1 | ON | OFF
      :DEBounce[:STATe}?
```

INPut:DEBounce[:STATe] 0 | 1 | ON | OFF

Turns on or off the 3mSec input debounce circuitry for the Input Register.

Parameters None

Comments

- ***RST Condition:** Debounce is off.

INPut:DEBounce[:STATe]?

Returns an unsigned integer indicating the current state of the input register debounce circuitry. A “1” indicates the 3mS debounce circuitry is enabled (ON), a “0” indicates the debounce circuitry is disabled (OFF).

Parameters None

Comments

- ***RST Condition:** Debounce is off.

The MEASure commands return data corresponding to the current value of the input signals.

Syntax

```
MEASure
        :DIGital:DATAn[:BYTE]:BITm?
        :DIGital:DATAn[:BYTE][:VALue]?
        :DIGital:DATA0:WORD:BITm?
        :DIGital:DATA0:WORD[:VALue]?
```

Note With no load or signal applied to the input circuits, the MEASure commands will respond with a high on all bits due to the internal pull-ups. If you source a low (SOURce command subsystem) and then perform a MEASure command the response may still show a low. It takes approximately 5µS after the output is disabled for the pull-ups to bring the input circuit up to the minimum 1.8V for a high indication.

MEASure:DIGital:DATA*n*[:BYTE]:BIT*m*?

Returns the value of BIT *m* of the data for the specified 8-bit Port *n* as an unsigned integer of either 0 or 1.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0 or 1	0
BIT*m*	Numeric	0 - 7	none

Comments

- Port 0 represents the bits 8 through 15 of the input, Port 1 represents the bits 0 through 7 of the input.
- A returned “0” indicates the input is low (≤0.8Vdc). A returned “1” indicates the input is high (≥1.8Vdc).

MEASure:DIGital:DATA*n*[:BYTE][:VALue]?

Returns the current data for the specified 8-bit Port *n* as an unsigned integer between the values of 0 and 255 (00_h - FF_h).

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0 or 1	0

Comments

- For any bit, a returned "0" indicates the input is low (≤ 0.8Vdc). A "1" indicates the input is high (≥ 1.8Vdc).
- Port 0 represents the bits 8 through 15 of the input, Port 1 represents the bits 0 through 7 of the input.

MEASure:DIGital:DATA0:WORD:BIT*m*?

Returns the value of BIT *m* of the data for the combined 16-bit word as an unsigned integer of either 0 or 1.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0	0
BIT*m*	Numeric	0 - 15	none

Comments

- You must specify DATA0.
- A returned "0" indicates the input is low (≤ 0.8Vdc). A returned "1" indicates the input is high (≥ 1.8Vdc).

MEASure:DIGital:DATA0:WORD[:VALue]?

Returns the current data for all 16 input bits as a signed integer between the values of -32768 and 32767 (0000_h - $FFFF_h$).

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0	0

Comments

- You must specify DATA0.
- For any bit, a returned "0" indicates the input is low (≤ 0.8Vdc). A "1" indicates the input is high (≥ 1.8Vdc).

The SENSe Subsystem allows matching of digital input patterns. An interrupt is generated when a pattern match occurs.

Syntax

```
SENSe
    [:EVENt][:PATTern] <pattern>
    [:EVENt][:PATTern]?
    [:EVENt][:PATTern]:ENABle <mask>
    [:EVENt][:PATTern]ENABle?
```

SENSe[:EVENt][:PATTern] <*pattern*>

Specifies the input bit pattern that will generate an interrupt. This command applies only to the complete input port (16-bit Word).

Parameters

Parameter Name	Parameter Type	Range of Values	Default
<*pattern*>	Numeric	-32768 to +32767 (0000_h to $FFFF_h$)	0

Comments

- You can also specify which bits can cause the interrupt. Refer to the SENSe[:EVENt][:PATTern]:ENABle<*mask*> command.

SENSe[:EVENt][:PATTern]?

Returns interrupt pattern if previously set, otherwise returns all zeros.

Parameters None

Comments

- Returns a value in the range of -32768 to +32767 (0000_h to $FFFF_h$) which corresponds to the pattern sent in the SENSe[:EVENt][:PATTern] <*pattern*> command.

SENSe[:EVENt][:PATTern]:ENABle <*mask*>

Sets the bits to be used in the pattern match interrupt.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
<*mask*>	Numeric	-32768 to +32767 (0000_h to $FFFF_h$)	0

Comments

- Set the mask for those bits you want to cause an interrupt when the pattern matches.

Returns the value of the Enable mask.

Parameters None

Comments

- Returns an integer in the range -32768 to +32767 (0000_h to $FFFF_h$).

The SOURce Subsystem is used to output (source) digital data from the Agilent E2290A to external instrumentation or device-under-test..

Syntax

```
[SOURce]
        :DIGital:DATAn[:BYTe]:BITm 0 | 1
        :DIGital:DATAn[:BYTe]:BITm?
        :DIGital:DATAn[:BYTe][:VALue] <data>
        :DIGital:DATAn[:BYTe][:VALue]?
        :DIGital:DATA0:WORD:BITm 0 | 1
        :DIGital:DATA0:WORD:BITm?
        :DIGital:DATA0:WORD[:VALue] <data>
        :DIGital:DATA0:WORD[:VALue]?
        :CONTrol[:ENABle]?
```

[SOURce]:DIGital:DATA*n*[:BYTe]:BIT*m* 0 | 1

Outputs a value on byte Port *n* (0 or 1) at bit position *m* (0 - 7). Note that you can do this on a bit-by-bit basis; for example, bits 0, 2, 3, and 5 on Port 0 may be sourcing (output), while bits 1, 4, 6, and 7 are sensing (input).

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0 or 1	0
BIT*m*	Numeric	0 - 7	none

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).
- Port 0 represents the bits 8 through 15 of the input, Port 1 represents the bits 0 through 7 of the input.

[SOURce]:DIGital:DATA*n*[:BYTe]:BIT*m*?

Returns value of bit *m* (0 - 7) from byte port *n* (0 or 1) currently being output to Port *n* (0 or 1). This is a read-back function of the output register (not the actual state of the data lines) to verify the current output data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0 or 1	0
BIT*m*	Numeric	0 - 7	none

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).
- Port 0 represents the bits 8 through 15 of the input, Port 1 represents the bits 0 through 7 of the input.

[SOURce]:DIGital:DATA*n*[:BYTe][:VALue] <*data*>

Outputs an 8-bit data byte to Port *n* (0 or 1). Note that you can do this on a bit-by-bit basis; for example, bits 0, 2, 3, and 5 on Port 0 may be sourcing (output), while bits 1, 4, 6, and 7 are sensing (input).

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0 or 1	0
data	Numeric	0 - 255 (00_h - FF_h)	none

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).
- Port 0 represents the bits 8 through 15 of the input, Port 1 represents the bits 0 through 7 of the input.

[SOURce]:DIGital:DATA*n*[:BYTe][:VALue]?

Returns the 8-bit data byte value currently being output to Port *n* (0 or 1). This is a read-back function of the output register (not the actual state of the data lines) to verify the current output data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
DATA*n*	Numeric	0 or 1	0

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).
- Port 0 represents the bits 8 through 15 of the input, Port 1 represents the bits 0 through 7 of the input.

[SOURce]:DIGital:DATA0:WORD:BIT*m* 0 | 1

Outputs a value on 16-bit word Port 0 at bit position *m* (0 - 15). Note that you can do this on a bit-by-bit basis; for example, bits 0, 2, 3, and 5 may be sourcing (output), while the remaining 12 bits are sensing (input).

Parameters

Parameter Name	Parameter Type	Range of Values	Default
BIT*m*	Numeric	0 - 15	none

Comments

- This command outputs a single 16-bit value, Bits 0 through 7 represent the lower 8-bits, bits 8 - 15 represents the upper 8-bits.
- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).

[SOURce]:DIGital:DATA0:WORD:BIT*m*?

Returns value of bit *m* (0 - 15) from 16-bit word Port 0. This is a read-back function of the output register (not the actual state of the data lines) to verify the current output data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
BIT*m*	Numeric	0 - 15	none

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).

[SOURce]:DIGital:DATA0:WORD[:VALue] <*data*>

Outputs a 16-bit word of data.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
data	Numeric	-32768 to +32767 (0000_h - $FFFF_h$)	none

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).

[SOURce]:DIGital:DATA0:WORD[:VALue]?

Returns the value of 16-bit word from Port 0 currently being output from the 16-bit port. This is a read-back function of the output register (not the actual state of the data lines) to verify the current output data.

Parameters

None

Comments

- For any bit, a “0” indicates the output is low (≤ 0.4Vdc). A “1” indicates the output is high (≥ 2.9Vdc).

Returns value of Output Enable Register, returns 16-bit value only.

Parameters None

Comments

- The Output Enable Register indicates whether the Ports (0 and 1) are configured as output or input ports. The lower 8-bits represent Port 0, the upper 8-bits represent Port 1. A 1 in the register bits indicate the port is sourcing (outputting); a 0 in the register bits indicates the port in inputting.
- The Enable Register is set when you either execute a SOURce command or a MEASure command.

The STATus subsystem controls the SCPI-defined Operation and Questionable Status registers, Standard Event register, and the Status Byte register. Each is comprised of a condition register, an event register, an enable mask, and transition filters.

Note Transition filters are always set for positive edge transitions. When an event occurs, the condition is set and the event register bit is set true. If the event condition is cleared, the event status register remains set. The event status register is cleared upon reading that register.

Each status register works as follows: when a condition occurs, the appropriate bit in the condition register is set or cleared. The contents of the events register and the enable mask are logically ANDed bit-for-bit; if any bit of the result is set, the summary bit for that register is set in the status byte. The status byte summary bit for the Operation status register is bit 7; for the Questionable Signal status register, bit 3; and for the Standard Event registers is bit 5.

Syntax

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

The STATus system contains five registers, two of which are under IEEE 488.2 control: the Event Status Register (*ESE?) and the Status Byte Register (*STB?). The Operational Status bit (OPR), Service Request bit (RQS), 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 Event Status 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 3-1.

Note The Questionable Status Condition, Event, and Enable registers exist for SCPI compliance only. No status bits are defined or reported in these registers.

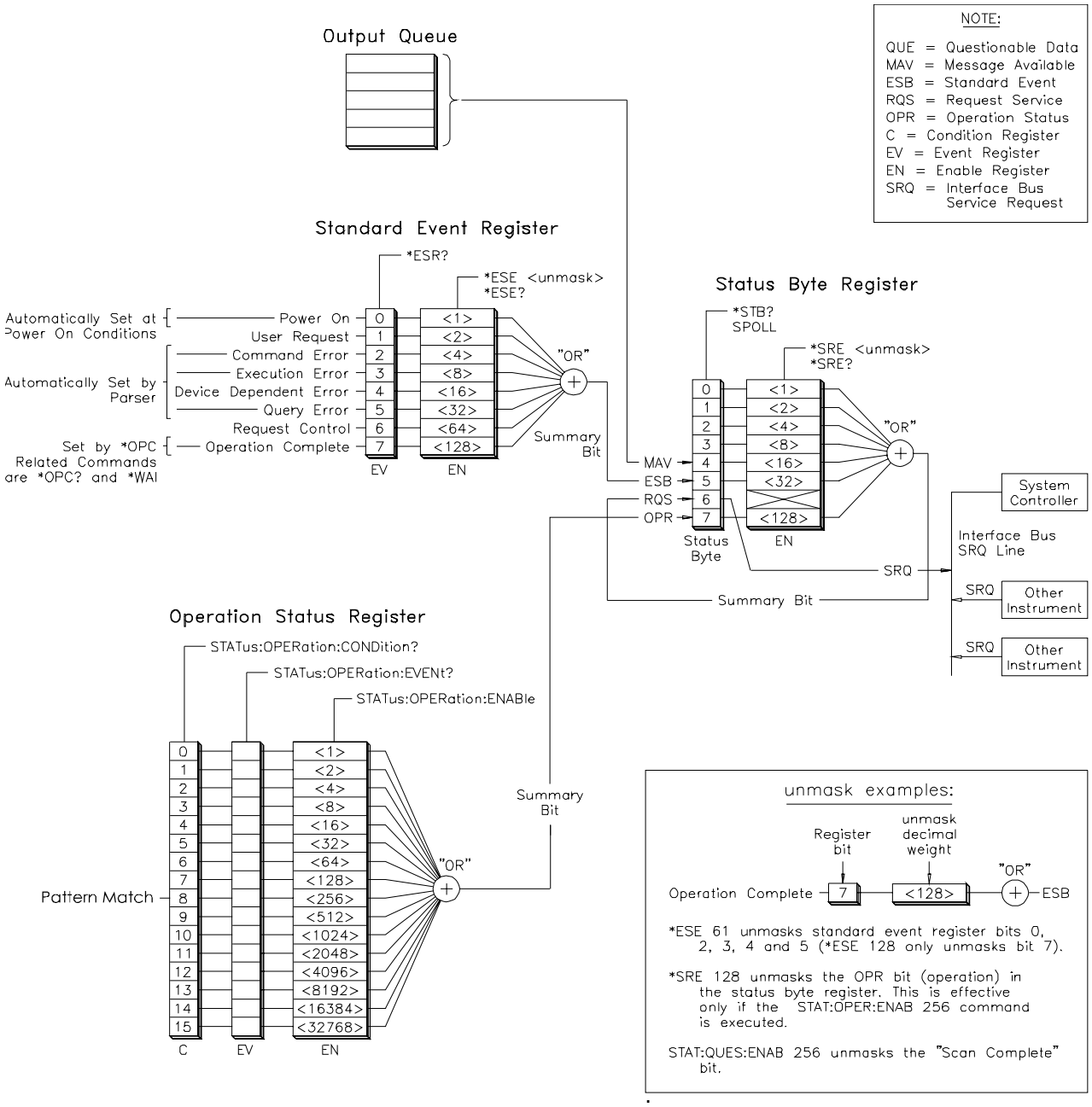


Figure 3-1. Agilent E2290A Status System Register Diagram

STATus:OPERation:CONDition?

Returns the value of the Operation Status Condition Register as a signed 16 bit integer.

Parameters None

Comments

- *RST clears all Status Operation Conditions.
- *CLS does not affect the contents of the of the Status Operation Conditions.
- The STATus:PRESet command does not affect the Status Operation Conditions.

STATus:OPERation:ENABle <*mask*>

Sets the value of the OPERation Status Enable Register.

Parameters

Parameter Name	Parameter Type	Range of Values	Default
<*mask*>	numeric	-32768 to +32767 (0000_h to $FFFF_h$)	0

Comments

- <*mask*> determines which OPERation Status conditions are summed. See Figure 3-1. The events detected in the Port Summary Status Register are reported in bit 8 of the Operation Status Register which in turn is reported in bit 7 of the Status Byte Register.
- *RST and *CLS do not affect the value of the enable mask.
- STATus:PRESet sets the value of the enable mask to 0.

Example

```
STAT:OPER:ENAB 0xFFFF
```

Enable all bits of the Operation Status Enable Register

STATus:OPERation:ENABle?

Returns the value of the OPERation Status Enable Register as a signed 16 bit integer.

Parameters None

Comments

- The only defined bit is bit 8 which is the summary of the Data Available and Edge Status for Ports 0, 1, 2, and 3. See Figure 3-1.

STATus:OPERation[:EVENt]?

Returns the value of the OPERation Status Event Register as a signed 16 bit integer and then clears the register to 0.

Parameters None

Comments

- *RST does not affect the contents of the Status Operation Event Register.
- *CLS clears the contents of the Status Operation Event Register.
- STAT:PRESet does not affect the contents of the Status Operation Event Register but does disable reporting the summary of this register in the Status Byte Register (STB?).

STATus:PRESet

Presets the Status system registers and conditions.

Parameters None

Comments

- Resets the following registers and conditions:

Register	Action		Register	Action
Status Byte	none		QUES Status enable	presets to 0
Standard Event event	none		OPER Status condition	none
Standard Event enable	presets to 0		OPER Status event	none
QUES Status Condition	none		OPER Status enable	presets to 0
QUES Status Event	none			

STATus:QUEStionable:CONDition?

Always returns a 0.

Note The Questionable Status Condition, Event, and Enable registers exist for SCPI compliance only. No status bits are defined or reported in these registers.

Parameters None

Comments

- No bits are defined.
- *RST clears all Status Questionable Conditions.
- *CLS does not affect the contents of the Status Questionable Conditions.
- The STAT:PRESet command does not affect the Status Questionable Conditions.

STATus:QUEStionable:ENABle <*mask*>

Sets the value of the QUEStionable Status Enable Register.

Note The Questionable Status Condition, Event, and Enable registers exist for SCPI compliance only. No status bits are defined or reported in these registers.

Parameters None

Comments

- No bits are defined.
- *RST and *CLS do not affect the value of the enable mask.
- The STAT:PRESet command sets the value of the enable mask to 0.

STATus:QUEStionable:ENABle?

Returns the value of the QUEStionable Status Enable Register as a signed 16 bit integer.

Note The Questionable Status Condition, Event, and Enable registers exist for SCPI compliance only. No status bits are defined or reported in these registers.

Parameters None

Comments

- No bits are defined.

STATus:QUEStionable[:EVENt]?

Returns the value of the QUEStionable Status Event Register as a signed 16 bit integer and then clears the register to 0.

Note The Questionable Status Condition, Event, and Enable registers exist for SCPI compliance only. No status bits are defined or reported in these registers.

Parameters None

Comments

- No bits are defined.
- This is a destructive read so that all register bits are cleared after the read is executed.
- *RST does not affect the contents of the Status Questionable Event Register.
- *CLS clears the contents of the Status Questionable Event Register.
- STAT:PRESet does not affect the contents of the Status Questionable Event Register but does disable reporting the summary of this register in the Status Byte register (STB?)

The SYSTem Subsystem returns module-specific information. This information includes module type and description, and error messages.

Syntax

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

SYSTem:ERRor?

Queries the error register for the error value and string to identify the error. The errors are held in an error buffer and read on a First-In-First-Out basis.

Parameters None

Comments

- Returns the error number and string. If no errors are in the error buffer, the command returns:
  `+0,"No error"`
- *CLS clears the error buffer.
- *RST does not affect the error buffer

Example

```
SYST:ERR?
```

Requests the error messages.

SYSTem:VERSion?

Returns the SCPI version to which this module complies.

Parameters None

Comments

- Returns a decimal value in the form: YYY.R where YYY is the year and R is the revision number within that year. Since there is no SCPI subsystem defined for Digital I/O, the version returned will be:
  `1990.0`

The following table lists the IEEE 488.2 Common Commands listed by functional group that can be executed by the Agilent E2290A Digital Input / Output Module. However, commands are listed alphabetically in the reference. Example are shown in the reference when the command has parameters or returns a non-trivial response; otherwise, the command string is as shown in the table. For additional information, refer to IEEE Standard 488.2-1987.

Command	Title	Description
*CLS	Clear Status Registers	Clears all STATus event registers and clears the error queue.
*ESE <mask>	Event Status Enable	Sets the bits in the Event Status Enable Register. <mask> has a range of 0 through 255 and must be entered in decimal format.
*ESE?	Event Status Enable Query	Returns the current programmed value of the Event Status Enable Register.
*ESR?	Event Status Register Query.	Queries and clears contents of the Standard Event Status Register.
*IDN?	Identification query	Returns the (unquoted) identification string: HEWLETT-PACKARD,E2290A,0,*revision*
*OPC	Operation Complete	This command always immediately sets the operation complete bit (bit 0) in the Standard Event Register because there are never any pending operations.
*OPC?	Operation Complete Query	This command always returns a 1 since there are never any pending operations.
*RCL<*state*>	Recalls stored instrument state from memory	Recalls the specified stored instrument state where <*state*> has a value of 0 through 9.
*RST	Resets the module	Resets the module to the settings shown in the "Power-On and Reset State" table following the individual common command descriptions.
*SAV<*state*>	Save state to memory	Saves the present instrument state in the specified memory location (1 to 9). Refer to *RCL.
*SRE <*mask*>	Service Request Enable	Sets the bits in the Service Request Enable Register. <mask> has a range of 0 through 255 and must be entered in decimal format.
*SRE?	Service Request Enable Query	Returns the current programmed value of the Service Request Enable Register.
*STB?	Status Byte	Returns the current value of the Status Byte Register.
*TST?	Self-Test	Returns "0" if self-test passed. Returns "1" if Ready bit in the Status Register is not equal to “1”. Returns “2” if write to Configuration register does not read back the correct value. Returns “3” if write to Compare Enable Register does not read back the correct value. Returns “4” if write to Compare Data Register does not return correct value. Returns “5” if module interrupt system failed. Instrument state returned to the power-on / reset state after *TST?
*WAI	Wait to Complete	Prevents execution of commands until the No Operation Pending message is true. Since each command is fully executed at the time of execution, the No Operation Pending message is always true and the *WAI command always immediately executes when received.

Command System	Description
DIAGnostic:INTerrupt:LINE DIAGnostic:INTerrupt:LINE? DIAGnostic:TEST?	Specifies VXI Interrupt Line. Queries VXI Interrupt Line. Performs an extended self test on the module.
DISPlay:MONitor[:STATe]<*state*> DISPlay:MONitor[:STATe]?	Turns display mode on or off. Returns value of Disp. Monitor State (0 or 1).
INPut:DEBounce[:STATe] ON | OFF INPut:DEBounce[:STATe]?	Turns on or off the 3mS debounce circuit. Returns value indicating current state of debounce circuit.
MEASure:DIGital:DATA*n*:WORD[:VALue]? MEASure:DIGital:DATA*n*:WORD:BIT*m*? MEASure:DIGital:DATAn[:BYTE][:VALue]? MEASure:DIGital:DATA*n*[:BYTE]:BIT*m*?	Returns value from 16-bit word input (specify DATA0). Returns value of Bit m from 16-bit word (specify DATA0). Returns value for specified Port *n* as a signed integer. Returns value of Bit *m* from Port *n*.
SENSe[:EVENt][:PATTern] <*pattern*> SENSe[:EVENt][:PATTern]? SENSe[:EVENt][:PATTern]:ENABle <*mask*>	Set interrupt pattern, applies only to Port 0 (word port). Returns interrupt pattern if set. Sets bit mask for pattern interrupt.
[SOURce]:DIGital:DATA*n*[:BYTE]:BIT*m* 0 | 1 [SOURce]:DIGital:DATA*n*[:BYTE]:BIT*m*? [SOURce]:DIGital:DATA*n*[:BYTE][:VALue] <*data*> [SOURce]:DIGital:DATA*n*[:BYTE][:VALue]? [SOURce]:DIGital:DATA*n*:WORD:BIT*m* 0 | 1 [SOURce]:DIGital:DATA*n*:WORD:BIT*m*? [SOURce]:DIGital:DATA*n*:WORD[:VALue] <*data*> [SOURce]:DIGital:DATA*n*:WORD[:VALue]? [SOURce]:CONTrol[:ENABle]?	Outputs a value on Port *n* (0 or 1) at bit position *m* (0 to 7). Returns the value of bit *m* (0 to 7) from Port *n* (0 or 1). Outputs data byte to Port *n* (0 or 1). Returns value of data byte output to Port *n* (0 or 1). Outputs a value to Bit *m* (0 to 15) on Word Port 0. Returns value of Bit *m* (0 to 15) from Word Port 0. Outputs data word (16-bit) to Port 0. Returns value of data word (16-bit) from Port 0. Returns value of Enable Register (16-bit).
STATus:OPERation:CONDition? STATus:OPERation:ENABle <*mask*> STATus:OPERation:ENABle? STATus:OPERation[:EVENt]? STATus:PRESet STATus:QUEStionable:CONDition? STATus:QUEStionable:ENABle <*mask*> STATus:QUEStionable:ENABle? STATus:QUEStionable[:EVENt]?	Returns value of Operation Status Condition Register as 16 bit int. Sets the value of the OPERation Status Enable Register. Returns value of OPERation Status Enable Register as 16 bit integer. Returns value of OPERation Status Event Register as 16 bit integer. Presets the Status system registers and conditions. Returns value of Questionable Status Condition Register as 16 bit integer. Sets the value of the QUEStionable Status Enable Register. Returns value of QUEStionable Status Enable Register as 16 bit integer. Returns value of QUEStionable Status Event Register as 16 bit integer.
SYSTem:ERRor? SYSTem:VERsion?	Queries error register for error value and string. Returns SCPI version.

Chapter 4
Register Programming

Introduction

This chapter describes how to program the Agilent E2290A Digital I/O M-Module at the register level in an Agilent E2251 Carrier installed in a VXIbus mainframe. Register programming is recommended only if you are unable to use the module's higher-level VXI*plug&play* or D-SCPI driver. D-SCPI is documented in this manual. For information on using the VXI*plug&play* driver, refer to the on-line help.

Block Diagram Description

In order to register program the Agilent E2290A, it is important to understand its operation at the block diagram level. Figure 4-1 shows a simplified block diagram of the module.

Module Control

This block contains all of the logic for the module including all registers, interrupt control, and carrier interface.

ID EEPROM

The EEPROM holds sixty-four 16-bit words of M-Module ID data and VXI M-module data. Refer to "ID EEPROM Register" on page 63 for EEPROM contents.

Input Circuit

The input comparator maintains correct TTL high and low levels by shifting the input voltages to compensate for the forward voltage drop of the blocking diode. A reference voltage of +1.9Vdc is applied to the inverting input of the comparator. When the input is in the range of 0Vdc to +4.3Vdc the blocking diode is forward biased, and its forward voltage drop is added to the applied voltage. For example, when 0Vdc is applied to the data line, +0.7Vdc is present on the comparator's non-inverting input. Similarly, when the input signal is greater than +1.2Vdc, a voltage greater than +1.9 Vdc is applied to the comparator's non-inverting input causing its output to go high. When the applied voltage is less than +1.2Vdc, a voltage less than +1.9Vdc is applied to the comparator causing its output to be low.

The pull-up resistor on the comparator's non-inverting input allows external ground connections and open circuits to be detected.

High Drive Circuit

The high drive circuit (active when sourcing a high, 2.9 Vdc) consists of a blocking diode and a PNP transistor circuit for each line. The transistor is in a high impedance state when the line is not output-enabled or the line is driving low.

Low Drive Circuit

The low drive circuit (active when sourcing a low, 0.4Vdc) consists of one MOSFET, and a pull-down resistor. The MOSFET is in a high impedance state when the line is not output-enabled or the line is driving high. During a low level output, the MOSFET is turned on creating a low impedance path from the channel input to channel common. The MOSFET can sink up to 200mA.

Ground and Power Supply Conditioning

This block filters the +5Vdc power to produce the VCC power necessary for the logic circuity and isolates the various grounds used by the module. The module does not use ±12Vdc power.

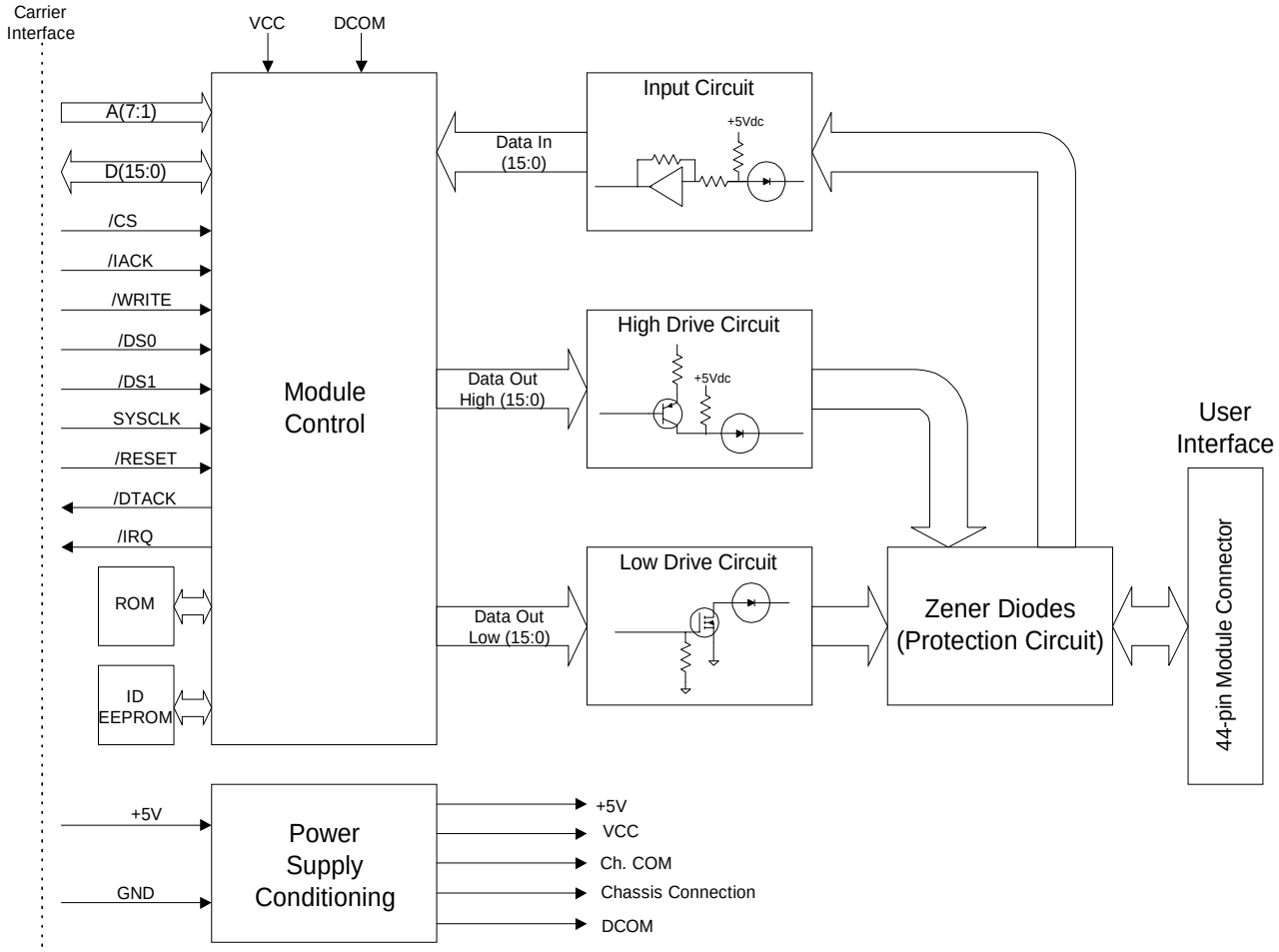


Figure 4-1. Agilent E2290A Simplified Block Diagram

Register Addressing in the VXIbus Environment

Logical Address

Each module in a VXIbus (VXI) system, whether VXI or M-Module, must have a unique logical address. The Agilent E2251 Carrier provides a logical address for each installed M-Module. Refer to the Agilent E2251 Installation and Getting Started Guide for details (if you are using a different carrier, refer to that carrier's documentation for register-based addressing information).

A16/A24 Memory Mapping

The VXI Specification allows for only 64 bytes of address space in A16 memory. However, the M-Module Specification defines 256 bytes of address space. To resolve this conflict, the Agilent E2251 Carrier provides two memory segments for each installed M-Module. The first is in the VXI A16 memory space and contains the standard VXI registers. The second memory segment is in the VXI A24 memory space and contains all other M-Module registers (these registers are described starting on page 56). Figure 4-2 on page 51 shows the A16/A24 mapping for a typical M-Module.

Note

The M-Module's ID word (from the ID EEPROM) is mapped into the VXI Manufacturer ID Register at address 00_h and the M-Module's VXI Device Type word is mapped into the VXI Device Type Register at address 02_h

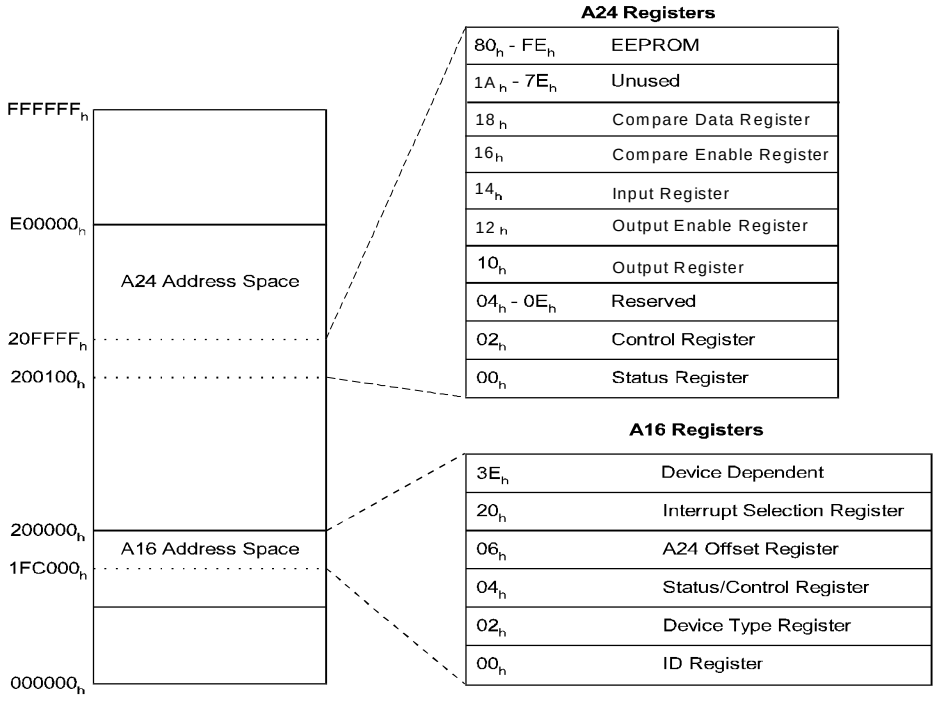


Figure 4-2. A16/A24 Address Mapping

Determining a Module's A16 Base Address

To access a register in A16 memory, you must specify a hexadecimal or decimal register address. This address consists of a base address plus a register offset. The A16 base address depends on whether or not you are using an Agilent E1406 Command Module.

A16 Address Space Inside the Command Module

When <u>using</u> an Agilent E1406 Command Module, the base address is computed as:

$1FC000_h + (LADDR_h \cdot 40_h)$
or (decimal)
2,080,768 + (LADDR · 64)

Where:
$1FC000_h$ (2,080,768) is the A16 starting address
LADDR is the module's logical address
40_h (64) is the number of address bytes allocated per module

For example, if the M-Module has a logical address of 78_h (120) the A16 base address becomes:

$1FC000_h + (78_h \cdot 40_h) = 1FC000_h + 1E00_h = 1FDE00_h$
or (decimal)
2,080,768 + (120 · 64) = 2,080,768 + 7680 = 2,088,448

A16 Address Space Outside the Command Module

When an Agilent E1406 Command Module is <u>not</u> part of your system, the base address is computed as:

$C000_h + (LADDR_h \cdot 40_h)$
or (decimal)
49,152 + (LADDR · 64)

Where:
$C000_h$ (49,152) is the A16 starting address
LADDR is the module's logical address
40_h (64) is the number of address bytes allocated per module

For example, if the M-Module has a logical address of 78_h (120) the A16 base address becomes:

$C000_h + (78_h \cdot 40_h) = C000_h + 1E00_h = DE00_h$
or (decimal)
49,152 + (120 · 64) = 49,152 + 7680 = 56,832

Addressing A16 Registers

As shown in Figure 4-2 on page 51, VXI registers for an M-Module are mapped into A16 address space. To access one of these registers, add the A16 base address to the register offset. For example, an M-Module's VXI Status/Control Register has an offset of 04_h. To access this register (assuming the system does not have an Agilent E1406 Command Module), use the register address:

$1FDE00_h + 04_h = 1FDE04_h$
or (decimal)
2,088,488 + 4 = 2,088,452

Addressing A24 Registers

As shown in Figure 4-2 on page 51, most of the registers for an M-Module are mapped into A24 address space. To access one of these registers:

1. Obtain the A24 base address by reading the VXI Offset Register (06_h) in A16 memory.

2. Add the A24 base address to the register offset (see Table 4-2).

For example, if the A24 base address is 200100_h, to access the Output Register (10_h):

$200100_h + 10_h = 200110_h$
or (decimal)
2,097,408 + 16 = 2,097,424

Program Example

The following C language program demonstrates how to program at the register level. The program reads the ID, Device Type, Status, and A24 registers then sets bits 00 and 02 to a HI state. This program was written and tested in Microsoft Visual C++ but should compile under any standard ANSI C compiler.

To run this program you must have the Agilent SICL library, the Agilent VISA library, an GPIB interface module installed in your PC, and an Agilent E2406 Command Module.

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

ViSession viRM,m_mod;
int main()
{

unsigned short id_reg,dt_reg ;              /* ID & Device Type Registers */
unsigned short stat_reg, a24_offset ;       /* Status Register & A24 offset
                                               register */
short value;

ViStatus errStatus;                         /*Status from each VISA call*/

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

    /* Open the M-Module instrument session ; Logical Address = 8 */
errStatus = viOpen(viRM,"GPIB-VXI0::8",VI_NULL,VI_NULL,&m_mod);
if (VI_SUCCESS > errStatus){
   printf("ERROR: viOpen() returned 0x%x\n",errStatus);
   return errStatus;}

    /* read  and print the module's ID Register  */
errStatus = viIn16(m_mod,VI_A16_SPACE,0x00,&id_reg);
if (VI_SUCCESS > errStatus){
   printf("ERROR: viIn16() returned 0x%x\n",errStatus);
   return errStatus;}
printf("ID register = 0x%4X\n", id_reg);

    /* read  and print the module's Device Type Register   */
errStatus = viIn16(m_mod,VI_A16_SPACE,0x02,&dt_reg);
if (VI_SUCCESS > errStatus){
   printf("ERROR: viIn16() returned 0x%x\n",errStatus);
   return errStatus;}
printf("Device Type register = 0x%4X\n", dt_reg);
```

```
    /* read and print the module's Status Register  */
errStatus = viIn16(m_mod,VI_A16_SPACE,0x04,&stat_reg);
if (VI_SUCCESS > errStatus){
   printf("ERROR: viIn16() returned 0x%x\n",errStatus);
   return errStatus;}
printf("Status register = 0x%hx\n", stat_reg);

    /* read and print the module's A24 Offset Register  */
errStatus = viIn16(m_mod,VI_A16_SPACE,0x06,&a24_offset);
if (VI_SUCCESS > errStatus){
   printf("ERROR: viOpen() returned 0x%x\n",errStatus);
   return errStatus;}
printf("A24 Offset register value = 0x%hx\n", a24_offset);

    /* Drive Bits 00 and 02 to HI State */
errStatus = viOut16(m_mod,VI_A24_SPACE,0x10,0x05);
if (VI_SUCCESS > errStatus){
   printf("ERROR: viOut16() returned 0x%x\n",errStatus);
   return errStatus;}

    /* Close the M-Module Instrument Session */
errStatus = viClose (m_mod);
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;
}
```

Registers in A16 Address Space

Table 4-1 lists the five registers in the A16 memory space. The following paragraphs describe each register.

Table 4-1. VXIbus A16 Memory Instrument Registers

Address Mapping	Registers
00_{16}	VXI ID Register
02_{16}	VXI Device Type Register
04_{16}	VXI Status/Control Register
06_{16}	VXI Offset Register
20_{16}	M-Module Interrupt Control Register

VXI ID Register

The ID Register is a read only register at address 00_h (MSB) and 01_h (LSB).

b+00$_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Undefined															
Read	Device Class		Address Space		Manufacturer ID											

- **Device Class:** this field should always be 11 indicating a register-based device.
- **Address Space:** 00 indicating A16/A24 device
- **Manufacturer ID**: 4095 (decimal) for Agilent Technologies M-Modules

VXI Device Type Register

The Device Type Register is a read only register at address 02_h (MSB) and 03_h (LSB). Reading this register returns a unique identifier for each M-Module.

b+02$_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Undefined															
Read	Required Memory				M-Module Model Code											

- **Required Memory**: 1111_h indicating 256 byte block required.
- **M-Module Model Code**: $F260_h$ for the Agilent E2290A.

VXI Status/Control Register

The Status/Control Register is a read/write register (address 04_h and 05_h) that controls the module and indicates its status.

b+04$_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write (Control)	A24 Enable	Reserved													Sysfail Inhibit	Reset
Read (Status)	A24 Active	MODID*	M-Module Device Dependent										Ready	Passed	Device Dependent	

- **A24 Enable.** A 1 in this field means access to the devices A24 registers is enabled.
- **Sysfail Inhibit.** Writing a 1 disables the M-Module from driving the SYSFAIL* line.
- **Reset.** Writing a 1 to this field forces the M-Module to reset.
- **A24 Active.** A 1 in this field indicates the M-Module's registers in A24 memory space can be accessed. Default = 1.
- **MODID*.** A 1 in this field indicates that the M-Module is not selected via the P2 MODID line. A 0 indicates the M-Modules is selected by a high state on the P2 MODID line.
- **Ready.** A 1 in this field indicates that the M-Module is ready to accept commands. A 0 indicates the M-Module is busy and not ready to accept commands.
- **Passed.** A 1 in this field indicates the M-Module passed its self test successfully. A 0 indicates the M-Module is either executing or has failed its self test.

VXI Offset Register

The Offset Register (address 06_h and 07_h) contains the value of the base address for accessing registers in the A24 address space.

b+06$_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	A24 Space Base address for those M-Modules needing A24 memory															
Read	A24 Space Base address for those M-Modules needing A24 memory															

Interrupt Selection Register

The Interrupt Selection Register (base + 20_h) specifies which VXI interrupt line the M-Module will use. M-Modules may generate interrupts to indicate that a SCPI command has completed. These interrupts are sent to and acknowledged by the Agilent Command Module or other system controller via one of seven VXI backplane interrupt lines. Different controllers treat the interrupt lines differently, and you should refer to your controller's documentation to determine how to set the interrupt level. Agilent Command Modules configured as VXI Resource Managers treat all interrupt lines as having equal priority. For interrupters using the same line, priority is determined by which slot they are installed in; lower-numbered slots have higher priority than higher-numbered slots. Agilent Command Modules service line 1 by default, so it is normally correct to leave the interrupt level set to the factory default of IRQ1.

b+20_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Reserved												INT	VXI Interrupt Line		
Read (default value)	Reserved												INT 1	VXI Interrupt Line 0	0	1

If your controller's documentation instructs you to change the interrupt level, you need to specify the level in the VXI Interrupt Selection Register. To cause the M-Module to interrupt on one of the VXI interrupt lines, write to the appropriate bits (refer to table below). To disable the module's interrupt, set the bits to 000. Selecting other than the default interrupt line 1 is not recommended. Reading the default value of this register returns the value $XXX9_h$.

Bits 2 - 0	Selected Interrupt Line
000	NONE (Interrupt Disabled)
001	IRQ1 (default)
010	IRQ2
011	IRQ3
100	IRQ4
101	IRQ5
110	IRQ6
111	IRQ7

M-Module specifications define three types of interrupts. The INT bit (bit 3) determines which M-Module interrupt style is supported. If INT is set to a 0, the M-Module supports interrupt types A and B. If INT is set to a 1, the M-Module supports interrupt type C (this is the default).

Type A Interrupts The interrupting M-Module removes the interrupt request upon a register access (software method) to the interrupting M-Module (such as reading the Status Register). DTACK* is not asserted during interrupt acknowledge.

Type B Interrupts The interrupting M-Module removes the interrupt request via a hardware method (on IACK* going low) but provides no vector information for the interrupt. This is the same as Type C interrupts except that no vector is supplied and DTACK* is not asserted.

Type C Interrupts The interrupting M-Module removes the interrupt request via a hardware method and provides an interrupt vector on the data bus and DTACK* is asserted during the interrupt acknowledge cycle. The M-Module removes the interrupt request by IACK* going low.

In VXI specifications however, only two types of interrupts are defined; RORA (Release on Register Access) and ROAK (Release on Acknowledge). The Agilent E2251A Carrier converts M-Module Type A interrupts to RORA and Types B and C interrupts to ROAK (default).

RORA Interrupts The interrupting device provides its logical address on the data bus (D0 - D7) during the interrupt acknowledge cycle that was initiated in response to its interrupt request. It does not remove the interrupt request until its Status/Control register is accessed.

ROAK Interrupts The interrupting device removes the interrupt request upon the presence of a properly addressed interrupt acknowledge cycle and provides its logical address on the data bus (D0 - D7). A cause/status byte is also placed on the data bus (D15 - D8)

Registers in A24 Address Space

Table 4-2 shows the register definitions for the Agilent E2290A.

Table 4-2. Agilent E2290A Registers

Word Address (Offset from A24 Base)	Register Name	Register Type
00_h	Status Register	Read Only
02_h	Control Register	Read/Write
04_h - $0E_h$	Reserved	NA
10_h	Output Register	Read/Write
12_h	Output Enable Register	Read/Write
14_h	Input Register	Read Only
16_h	Compare Enable Register	Read/Write
18_h	Compare Data Register	Read/Write
$1A_h$ - $7E_h$	Unused	NA
80_h - FE_h	ID EEPROM	Read/Write

Status Register

This register monitors the module's Ready and Interrupt Status states.

b+00_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Reserved															
Read	Reserved														Int. Status	Ready

Reset Condition -- Bit 00 = logic "1", Bit 01 = logic "0".

Bit Definitions

Ready -- When set to logic "1", the M-Module is ready to accept read or write instructions. When set to logic "0", the M-Module is busy and will not accept any new instructions.

Interrupt Status -- This bit mirrors the module's interrupt (IRQ) line. When set to logic "1", an interrupt is being asserted. When set to logic "0", interrupt is not being asserted.

Control Register

This register controls module reset (soft reset), interrupt enabling, and debounce enabling.

b+02_h	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Reserved													Deb. Cont.	Int. Enable	Reset
Read	Reserved													Deb. Cont.	Int. Enable	Reset

Reset condition -- Bits 00 - 03 = logic "0".

Bit Definitions

Reset -- Writing a logic “1” to this bit initiates a soft reset. Writing a logic “0” to this bit terminates soft reset.

Interrupt Enable -- Writing a logic “1” to this bit enables the M-Module to interrupt upon a pattern match (see "Interrupt Registers" on page 62). Writing a logic “0” to this bit disables interrupting.

Debounce Control -- Writing a logic “1” to this bit enables the debounce circuitry. When enabled, input data will incur a 3 ms debounce time period prior to being latched into the Input Register. Writing a logic “0” to this bit disables the debounce circuitry--input data will be immediately latched into the Input Register.

Output Register

This register acts as a latch for data being output from the carrier to the device under test.

$b+10_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Read	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00

Reset Condition -- Bits 15 - 00 = logic “0”

Bit Definitions

D00 - D15 -- These are the 16 data output bits corresponding to the data output lines D00 - D15.

Output Enable Register

This register enables/disables output data lines.

$b+12_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Read	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00

Reset Condition -- Bits 15 - 00 = logic “0”

Bit Definitions

D00 - D15 -- These are the 16 output enable bits corresponding to the data output lines D00 - D15. When you write a logic “1” to a specific bit, the corresponding data line is enabled to output data. When you write a logic “0” to a specific bit, the corresponding data line is disabled from outputting data.

Input Register

This register acts as a latch for data being input to the M-Module from the device under test.

$b+14_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Reserved															
Read	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00

Reset Condition -- Bits 15 - 00 = logic “0”

Bit Definitions

D00 - D15 -- These are the 16 data input bits corresponding to the data input lines D00 - D15.

Note

If the module is reset, with no loads or signals applied, the input register will contain the value $FFFF_h$ because of the internal pull-up resistors. If you source a low and then perform and then read the input register the response may still show a low. It takes approximately 5µS after the output is disabled for the pull-ups to bring the input circuit up to the minimum 1.8V for a high indication.

Interrupt Registers

An interrupt occurs (if enabled) when an input data pattern matches the pattern stored in the Compare Data Register. You control which data lines will be used in the comparison by enabling/disabling bits in the Compare Enable Register. Additionally, the Interrupt Enable bit in the Control Register must be high for an interrupt to occur.

Compare Enable Register

This register enables/disables bits for interrupt pattern matching.

$b+16_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Read	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00

Reset Condition -- Bits 15 - 00 = logic “0”

Bit Definitions

D00 - D15 -- These are the 16 compare enable bits corresponding to the bits in the Compare Data Register. When you write a logic “1” to a specific bit in the Compare Enable Register, that bit will be used in the pattern comparison. When you write a logic “0” to a specific bit, that bit will not be used in the pattern comparison.

Compare Data Register

This register is where you store a data pattern for interrupt pattern matching.

b+18$_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00
Read	D15	D14	D13	D12	D11	D10	D09	D08	D07	D06	D05	D04	D03	D02	D01	D00

Reset Condition -- Bits 15 - 00 = logic “0”

Bit Definitions

D00 - D15 -- These bits contain the pattern to be matched in interrupt pattern comparison.

ID EEPROM Register

The ID EEPROM Register allows you to access the contents of the ID EEPROM. The ID EEPROM contains sixty-four 16-bit words of M-Module ID data and VXI M-Module data.

Note

This register is intended to be used by the higher-level software driver. The software driver must perform a series of many reads and writes to this register to perform the required functions within the EEPROM. **When register programming, it is much easier to read the module ID data from the VXI registers (A16 memory area) instead of reading the ID EEPROM Register.** A16 addressing is discussed earlier in this chapter. Do NOT attempt to read the ID EEPROM. Do not attempt to read the EEPROM Registers.

b+80$_h$ - FE$_h$	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write	Unused															
Read														Chip Select	Clock	Data In/Out

Reset Condition -- Bits 15 - 08 = logic “1”, Bits 07 - Bit 00 = logic “0”.

Caution

Do not attempt to write to Bit 00 of the ID EEPROM register. You could possibly write-over the contents of the ID EEPROM.

Bit Definitions

Data In/Out -- Reading this bit returns the value returned from the Data Out pin of the ID EEPROM.

Clock -- Writing a logic “1” to this bit forces the SK pin of the ID EEPROM high and writing a logic “0” drives it low. This bit is used as a clock to the ID EEPROM for reading data out. Reading this bit always returns logic “0”.

Chip Select -- Writing a logic “1” to this bit selects the ID EEPROM. Writing a logic “0” to this bit deselects the EEPROM. Reading this bit always returns logic “0”.

Table 4-3. ID EEPROM Contents

Word Number	Description	ID EEPROM Contents
0	Sync Code	5346_h
1	M-Module Number (binary code)	$069A_h$ (binary-coded 1690)
2	Revision Number (binary code)	0001_h
3	Module Characteristics	0868_h
4 - 7	Reserved	0000_h
8 - 15	Module Dependent	0000_h
16	VXI Sync Code	$ACBA_h$
17	VXI ID	$CFFF_h$ (Agilent Technologies)
18	VXI Device Type	$F260_h$
19 - 31	Reserved	0000_h
32 - 63	Module Dependent	0000_h

Appendix A
Specifications

M-Module Specification Compliance

The Agilent E2290A complies with the Mezzanine M-Module Specification. The Agilent E2290A also supports:

- A08, D16 accesses
- INTC interrupts
- IDENT capability

Agilent E2290A Specifications

Carrier Interface Specifications

The Agilent E2290A incorporates the standard 40-pin, 20x2 row connector interface to the carrier board for power and data/control, but does not have the 24-pin optional connector for carrying user-connections back onto the carrier board.

I/O Lines Specifications

Maximum Output Voltage: +30 VDC (I/O to Chassis or common)

Output Characteristics:
Vout (hi): ≥ 2.9 V @ I_{source} ≤ 20 mA
Vout(lo): ≤ 0.4 V @ I_{sink} ≤ 200 mA

Input Characteristics:
Vin(hi): ≥ 1.8 V
Vin(lo): ≤ 0.8 V

User Connector

44-pin D-Sub connector.

Power Requirements

	I_{PM} (A)	I_{DM} (A)
+5VDC	0.385	0.350
+12VDC	0	0
-12VDC	0	0

Symbols

A

B

C

D

E

F

T

U

V

W