

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

**Artisan Technology Group**
(217) 352-9330 | sales@artisantg.com | artisantg.com

**Full-service, independent repair center**
with experienced engineers and technicians on staff.

**We buy your excess, underutilized, and idle equipment**
along with credit for buybacks and trade-ins.

**Custom engineering**
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

**Expert team | Trust guarantee | 100% satisfaction**

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *National Instruments PCI-MIO-16XE-50 / PCI-6011E* at our website: *Click HERE*

# DAQ

# PCI E Series Register-Level Programmer Manual

**Multifunction I/O Boards for PCI Bus Computers**

NATIONAL INSTRUMENTS™

November 1998 Edition
Part Number 341079B-01

**Internet Support**
E-mail: support@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

**Bulletin Board Support**
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

**Fax-on-Demand Support**
512 418 1111

**Telephone Support (USA)**
Tel: 512 795 8248
Fax: 512 794 5678

**International Offices**
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

**National Instruments Corporate Headquarters**
6504 Bridge Point Parkway  Austin, Texas 78730-5039  USA  Tel: 512 794 0100

© Copyright 1998 National Instruments Corporation. All rights reserved.

# Important Information

## Warranty

The PCI E Series boards are warranted against defects in materials and workmanship for a period of one year from the date of shipment, as evidenced by receipts or other documentation National Instruments will, at its option, repair or replace equipment that proves to be defective during the warranty period This warranty includes parts and labor

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period National Instruments does not warrant that the operation of the software shall be uninterrupted or error free

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty

National Instruments believes that the information in this manual is accurate The document has been carefully reviewed for technical accuracy In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition The reader should consult National Instruments if errors are suspected In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence Any action against National Instruments must be brought within one year after the cause of action accrues National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control

## Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation

## Trademarks

CVI™, DAQ-PnP™, DAQ-STC™, LabVIEW™, MITE™, NI-DAQ™, NI-PGIA™, RTSI™, and SCXI™ are trademarks of National Instruments Corporation

Product and company names listed are trademarks or trade names of their respective companies

## WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment

# Contents

## About This Manual


Organization of This Manual ..... xii
Conventions Used in This Manual ..... xii
Related Documentation ..... xiii
Customer Communication ..... xiii


## Chapter 1
## General Description


General Characteristics ..... 1-1


## Chapter 2
## Theory of Operation


Functional Overview ..... 2-1
PCI Interface Circuitry ..... 2-6
Analog Input and Timing Circuitry ..... 2-7
Analog Input Circuitry ..... 2-8
Data Acquisition Timing Circuitry ..... 2-11
Single-Read Timing ..... 2-11
Data Acquisition Sequence Timing ..... 2-12
Posttrigger and Pretrigger Acquisition ..... 2-18
Analog Triggering ..... 2-19
Analog Output and Timing Circuitry ..... 2-20
Analog Output Circuitry ..... 2-21
Analog Output Timing Circuitry ..... 2-22
Single-Point Output ..... 2-22
Waveform Generation ..... 2-23
Digital I/O Circuitry ..... 2-24
Timing I/O Circuitry ..... 2-24
RTSI Bus Interface Circuitry ..... 2-25


## Chapter 3
## Register Map and Descriptions


Register Map ..... 3-1
Register Sizes ..... 3-3
Register Descriptions ..... 3-3
Misc Register Group ..... 3-3
Serial Command Register ..... 3-4

Misc Command Register ..... 3-5
Status Register ..... 3-6
Analog Input Register Group ..... 3-7
ADC FIFO Data Register ..... 3-8
Configuration Memory Low Register ..... 3-9
Configuration Memory High Register ..... 3-11
Analog Output Register Group ..... 3-15
AO Configuration Register ..... 3-16
DAC FIFO Data Register ..... 3-18
DAC0 Direct Data Register ..... 3-19
DAC1 Direct Data Register ..... 3-20
DMA Control Register Group ..... 3-21
AI AO Select Register ..... 3-22
G0 G1 Select Register ..... 3-23
DAQ-STC Register Group ..... 3-24
FIFO Strobe Register Group ..... 3-24
Configuration Memory Clear Register ..... 3-24
ADC FIFO Clear Register ..... 3-24
DAC FIFO Clear Register ..... 3-24

# Chapter 4
# Programming

PCl Local Bus ..... 4-1
PCI Initialization for the IBM Compatible System ..... 4-2
Re-mapping the PCI E Series Board ..... 4-3
PCI Initialization for the Macintosh ..... 4-4
Windowing Registers ..... 4-5
Programming Examples ..... 4-5
Digital I/O ..... 4-7
Example 1 ..... 4-7
Example 2 ..... 4-7
Analog Input ..... 4-8
Example 1 ..... 4-9
Example 2 ..... 4-12
Example 3 ..... 4-14
Example Program ..... 4-15
Example 4 ..... 4-17
Programming the MITE for Different DMA Transfers ..... 4-20
Example 5 ..... 4-21
Example 6 ..... 4-23
Example 7 ..... 4-25
Example 8 ..... 4-27

Example 9 ..... 4-29
Analog Output ..... 4-31
Example 1 ..... 4-32
Example 2 ..... 4-34
Example 3 ..... 4-39
Example 4 ..... 4-41
Example 5 ..... 4-43
Example Program ..... 4-43
General-Purpose Counter/Timer ..... 4-45
Example 1 ..... 4-45
Example 2 ..... 4-47
Example 3 ..... 4-49
RTSI Trigger Lines Programming Considerations ..... 4-52
Analog Triggering ..... 4-52
Interrupt Programming ..... 4-56
Interrupt Sharing ..... 4-56
DMA Programming ..... 4-57
The Link Chaining Mode for DMA Transfer ..... 4-58

**Chapter 5**
**Calibration**

About the EEPROM ..... 5-1
Calibration DACs ..... 5-14
NI-DAQ Calibration Function ..... 5-17

**Appendix A**
**Customer Communication**

**Glossary**

**Index**

# Figures


Figure 2-1. PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-6071E Block Diagram ..... 2-1
Figure 2-2. PCI-MIO-16XE-10, PCI-6052E, and PCI-6031E Block Diagram ..... 2-2
Figure 2-3. PCI-6023E, PCI-6024E, and PCI-6025E Block Diagram ..... 2-3
Figure 2-4. PCI-6032E and PCI-6033E Block Diagram ..... 2-4
Figure 2-5. PCI-MIO-16XE-50 Block Diagram ..... 2-5
Figure 2-6. PCI Bus Interface Circuitry Block Diagram ..... 2-7
Figure 2-7. Analog Input and Data Acquisition Circuitry Block Diagram ..... 2-8
Figure 2-8. ADC Timing ..... 2-12
Figure 2-9. Timing of Scan in Example 1 ..... 2-14
Figure 2-10. Multirate Scanning of Two Channels ..... 2-15
Figure 2-11. Multirate Scanning of Two Channels with 1:*x* Sampling Rate ..... 2-15
Figure 2-12. Multirate Scanning of Two Channels with 3:1:1 Sampling Rate ..... 2-16
Figure 2-13. Multirate Scanning of Three Channels with 4:2:1 Sampling Rate ..... 2-16
Figure 2-14. Multirate Scanning without Ghost ..... 2-17
Figure 2-15. Occurrences of Conversion on Channel 1 in Example 3 ..... 2-17
Figure 2-16. Successive Scans Using Ghost ..... 2-17
Figure 2-17. Analog Output Circuitry Block Diagram ..... 2-20
Figure 2-18. DAQ-STC Counter Diagram ..... 2-24
Figure 2-19. RTSI Bus Interface Circuitry Block Diagram ..... 2-26

Figure 4-1. Analog Trigger Structure ..... 4-54
Figure 4-2. DMA Structure ..... 4-57
Figure 4-3. DMA Link Chaining Mode Structure ..... 4-59

Figure 5-1. EEPROM Read Timing ..... 5-2
Figure 5-2. Calibration AC Write Timing ..... 5-16


# Tables


Table 2-1. PGIA Gain Set Verses Board ..... 2-9
Table 2-2. Analog Input Configuration Memory ..... 2-18

Table 3-1. PCI E Series Register Map ..... 3-2
Table 3-2. PCI E Series Windowed Register Map ..... 3-3
Table 3-3. PGIA Gain Selection ..... 3-10
Table 3-4. Calibration Channel Assignments ..... 3-12
Table 3-5. Differential Channel Assignments ..... 3-13
Table 3-6. Nonreferenced Single-Ended Channel Assignments ..... 3-13
Table 3-7. Referenced Single-Ended Channel Assignments ..... 3-14
Table 3-8. Auxiliary Channel Assignments ..... 3-15
Table 3-9. Channel Assignments ..... 3-15

Table 5-1. PCI-MIO-16E-1, PCI-MIO-16E-4, PCI-6071E EEPROM Map .....5-3
Table 5-2. PCI-MIO-16XE-50 EEPROM Map .....5-5
Table 5-3. PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E EEPROM Map .....5-7
Table 5-4. PCI-6023E EEPROM Map .....5-9
Table 5-5. PCI-6024E and PCI-6025E EEPROM Map .....5-10
Table 5-6. PCI-6052E EEPROM Map .....5-12
Table 5-7. Type of CALDAC Used on Board .....5-14

# About This Manual

This manual describes the registers and register map of the PCI E Series boards and contains information concerning their register-level programming.

The DAQ-STC, a National Instruments system timing controller ASIC, is the timing engine that drives the PCI E Series boards. Consequently, the timing and programming sections in this manual repeat certain information from, or draw your attention to, sections in the *DAQ-STC Technical Reference Manual*. You must use your register-level programmer manual along with the *DAQ-STC Technical Reference Manual* for a complete understanding of PCI E Series board programming.

Unless otherwise noted, text applies to all boards in the PCI E Series. The PCI E Series boards are:

- PCI-MIO-16E-1
- PCI-MIO-16E-4
- PCI-MIO-16XE-10
- PCI-MIO-16XE-50
- PCI-6023E
- PCI-6024E
- PCI-6025E
- PCI-6031E (MIO-64XE-10)
- PCI-6032E (AI-16XE-10)
- PCI-6033E (AI-64XE-10)
- PCI-6052E
- PCI-6071E (MIO-64E-1)

The PCI E Series boards are high-performance multifunction analog, digital, and timing I/O boards for the PCI bus computers. Supported functions include analog input, analog output, digital I/O, and timing I/O.

# Organization of This Manual

The *PCI E Series Register-Level Programmer Manual* is organized as follows:

- Chapter 1, *General Description*, describes the general characteristics of the PCI E Series boards.
- Chapter 2, *Theory of Operation*, contains a functional overview of the PCI E Series boards and explains the operation of each functional unit making up the PCI E Series boards.
- Chapter 3, *Register Map and Descriptions*, describes in detail the address and function of each of the PCI E Series control and status registers.
- Chapter 4, *Programming*, contains programming instructions for operating the circuitry on the PCI E Series boards.
- Chapter 5, *Calibration*, explains how to calibrate the analog input and output sections of the PCI E Series boards by reading calibration constants from the EEPROM and writing them to the calibration DACs.
- Appendix A, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

# Conventions Used in This Manual

The following conventions are used in this manual:

<> Angle brackets containing numbers separated by an ellipsis represent a range of values associated with a bit or signal name—for example, DBIO<3..0>.

☞ This icon to the left of bold italicized text denotes a note, which alerts you to important information.

| | |
|---|---|
| **bold** | Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows 95 tabs, or LEDs. |
| ***bold italic*** | Bold italic text denotes a note, caution, or warning. |
| *italic* | Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text from which you supply the appropriate word or value, as in Windows 3.*x*. |
| Macintosh | Macintosh refers to all Macintosh computers with the PCI bus, unless otherwise noted. |
| `monospace` | Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and for statements and comments taken from programs. |
| PC | PC refers to the IBM PC AT and compatible computers with the PCI bus. |

# Related Documentation

The following National Instruments manuals contain general information and operating instructions for the PCI E Series boards:

- *PCI E Series User Manual*
- *DAQ-STC Technical Reference Manual*
- *PCI-6023E/6024E/6025E User Manual*

# Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix A, *Customer Communication*, at the end of this manual.

# 1

# General Description

This chapter describes the general characteristics of the PCI E Series boards.

## General Characteristics

The PCI E Series boards are Plug and Play-compatible multifunction analog, digital, and timing I/O boards for the PCI bus computers. This family of boards features 12-bit and 16-bit ADCs with 16 and 64 analog inputs, 12-bit and 16-bit DACs with voltage outputs, eight TTL-compatible digital I/O, and two 24-bit counter/timers for timing I/O. Because the PCI E Series boards have no DIP switches, jumpers, or potentiometers, they are easily configured and calibrated using software. This feature is made possible by the National Instruments MITE bus interface chip to connect the boards to the PCI I/O bus. The MITE implements the PCI Local Bus Specification so that the DMA, interrupts, and base address are all software configurable.

**Note** ***Revision C and earlier versions of the PCI-MIO-16XE-50 use the MITE as the interface chip and do not support the DMA feature.***

The PCI E Series boards use the National Instruments DAQ-STC system timing controller for time-related functions. The DAQ-STC consists of three timing groups that control analog input, analog output, and general-purpose counter/timer functions. These groups include a total of seven 24-bit and three 16-bit counters and a maximum timing resolution of 50 ns.

A common characteristic with DAQ boards is that you cannot easily synchronize several measurement functions to a common trigger or timing event. The PCI E Series boards have the Real-Time System Integration (RTSI) bus to solve this problem. The RTSI bus consists of our RTSI bus interface and a ribbon cable to route timing and trigger signals between several functions on up to five DAQ boards in your PCI bus computer.

The PCI E Series boards can interface to an SCXI system so that you can acquire over 3,000 analog signals from thermocouples, RTDs, strain gauges, voltage sources, and current sources. You can also acquire or

generate digital signals for communication and control. SCXI is the instrumentation front end for plug-in DAQ boards.

Your PCI E Series board is completely software configurable. Refer to your *PCI E Series User Manual* if you have not already installed and configured your board.

# 2

# Theory of Operation

This chapter contains a functional overview of the PCI E Series boards and explains the operation of each functional unit making up the PCI E Series boards.

## Functional Overview

The block diagram in Figures 2-1 through 2-5 give a functional overview of each PCI E Series board.

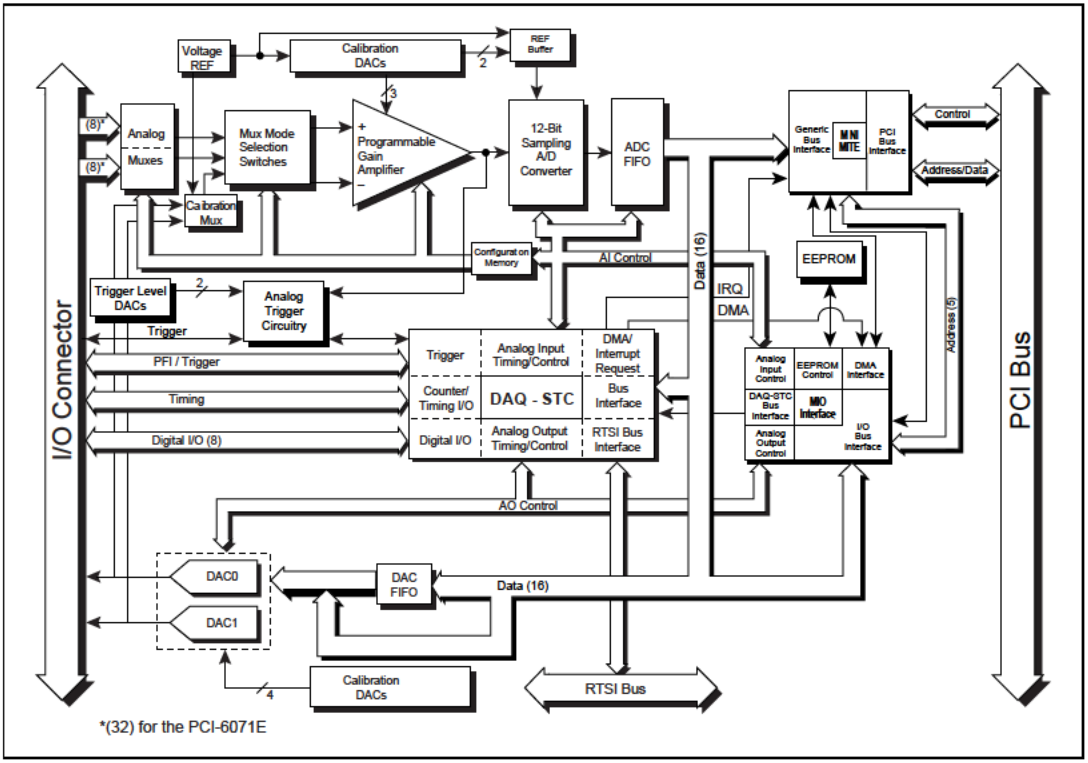



**Figure 2-1.** PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-6071E Block Diagram

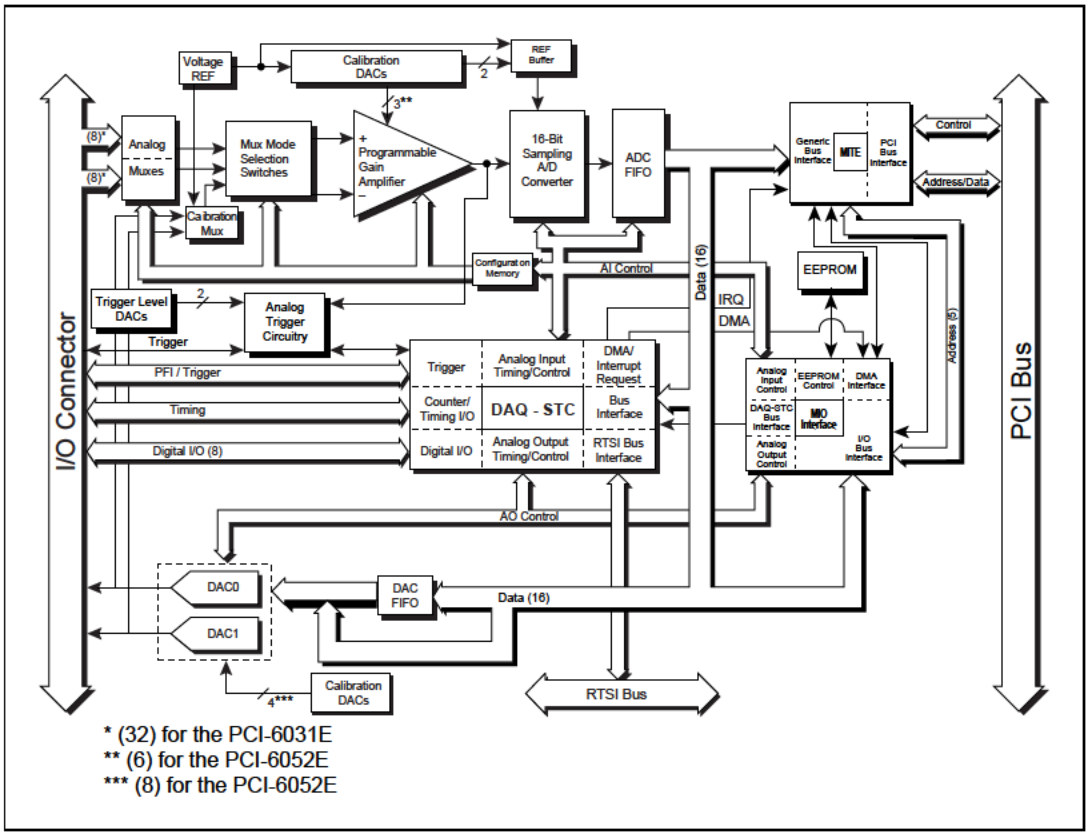



**Figure 2-2.** PCI-MIO-16XE-10, PCI-6052E, and PCI-6031E Block Diagram

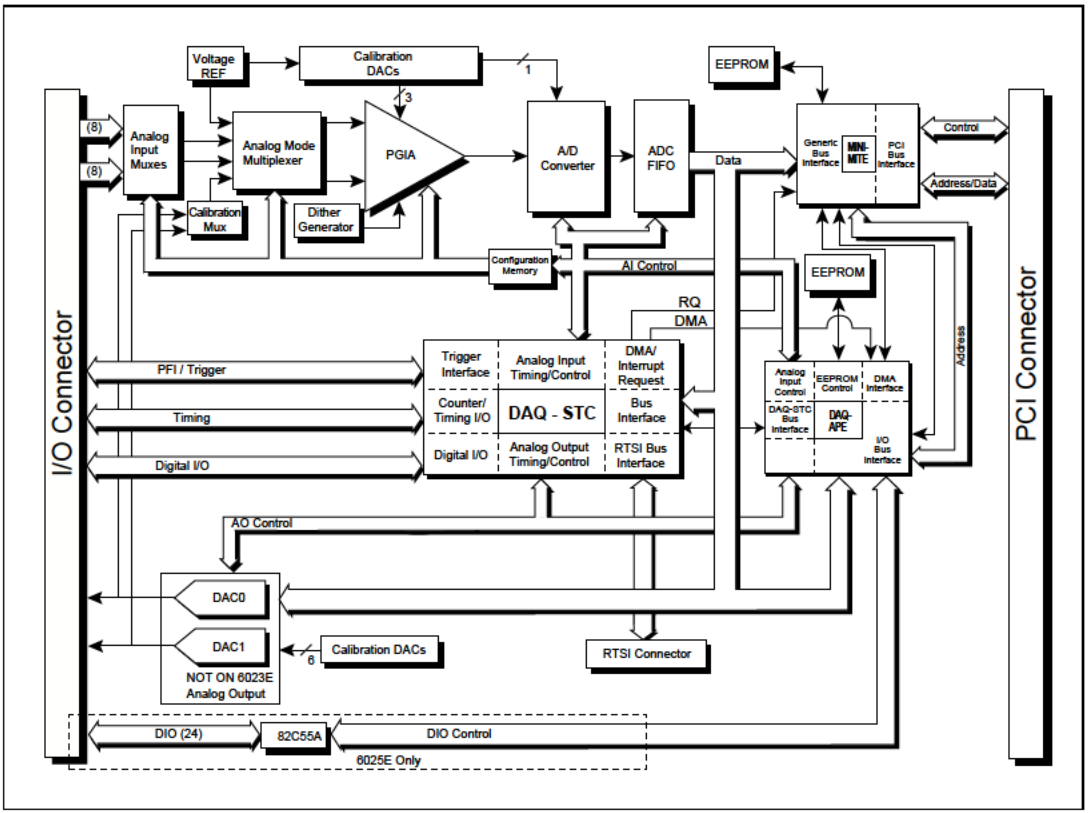



**Figure 2-3.** PCI-6023E, PCI-6024E, and PCI-6025E Block Diagram

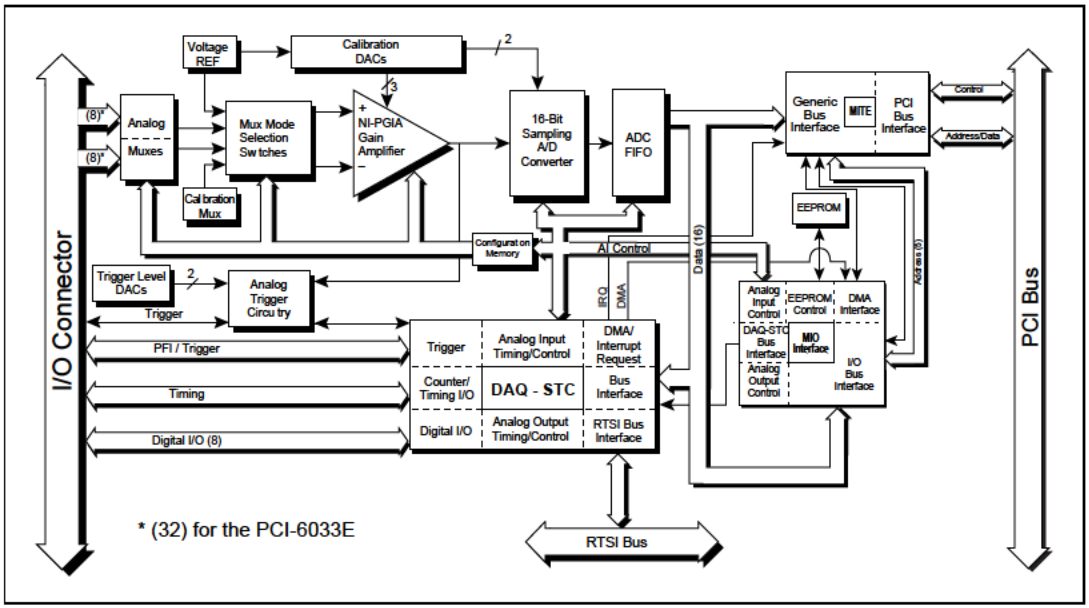



**Figure 2-4.** PCI-6032E and PCI-6033E Block Diagram

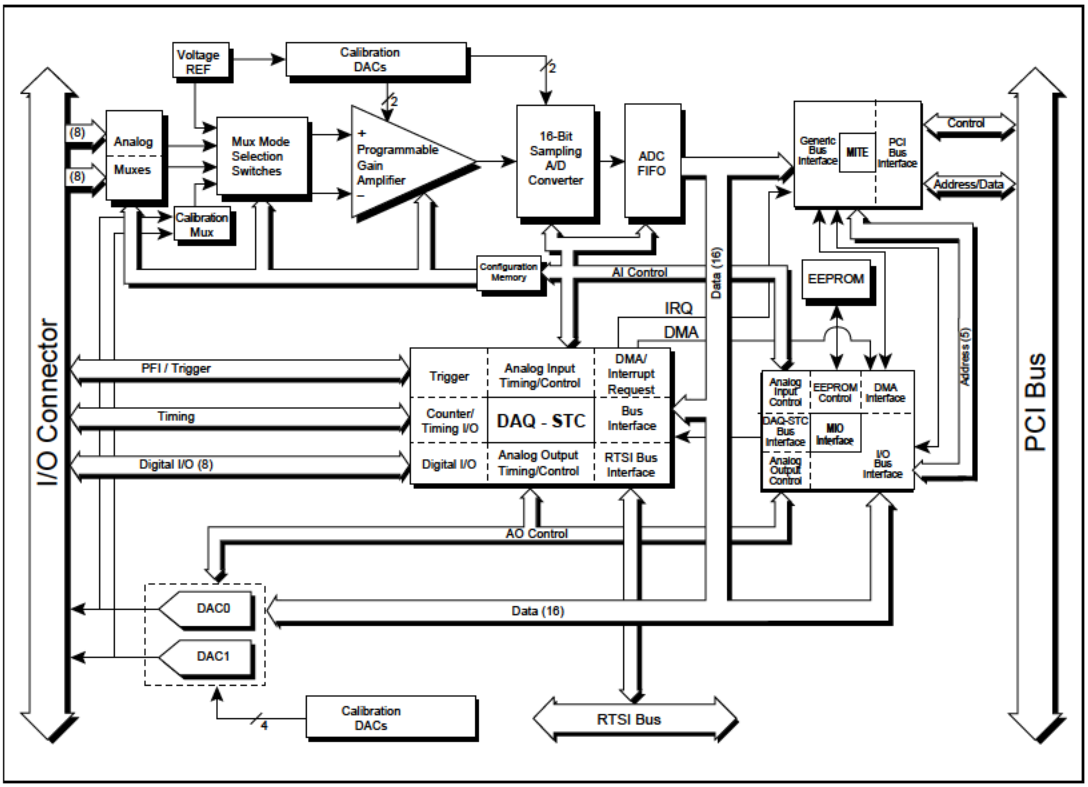



**Figure 2-5.** PCI-MIO-16XE-50 Block Diagram

The following major components make up the PCI E Series boards:

- PCI bus interface circuitry with Plug and Play capability (MITE)
- Analog input circuitry
- Analog trigger circuitry
- Analog output circuitry
- Digital I/O circuitry
- Timing I/O circuitry (DAQ-STC)
- RTSI bus interface circuitry

The internal data and control buses interconnect the components. Notice that the DAQ-STC is the timing engine that provides precise timing signals for the analog input and output operations. The timing I/O circuitry information in this manual is skeletal in nature and is sufficient in most cases. For register-level programming information, refer to the *DAQ-STC Technical Reference Manual*.

## PCI Interface Circuitry

The PCI E Series interface circuitry consists of a PCI interface chip and a digital control logic chip. The PCI interface chip provides a mechanism for the PCI E Series to communicate with the PCI bus. The digital control logic chip connects the PCI interface chip with the rest of the board. The PCI E Series is fully compliant with *PCI Local Bus Specification*, Revision 2.0. Therefore, the base memory address and the interrupt level for the board are stored inside the PCI interface chip at power on. You do not need to set any switches or jumpers. The PCI bus is capable of 8-bit, 16-bit, or 32-bit transfers, but PCI E Series boards use only 8-bit or 16-bit transfers.

The bus-mastering capabilities of the MITE provides high-speed data transfer between the board and system memory. The MITE contains three DMA channels that can be used simultaneously for data transfer with analog input, analog output, and the general-purpose counters. The MITE can control the PCI bus and transfer the data without interrupting the host processor.

The DAQ-STC can generate interrupts from over 20 sources and can route these interrupts to the INTA line on the PCI bus interface. Using two interrupt lines, such as INTB, INTC or INTD, is not permitted for the PCI E Series since each function in the PCI E Series does not have its own configuration space. PCI E Series boards have the DAQ-STC IRQOUT0 line connected to the MITE interrupt input. Therefore, when setting up interrupts you must route all interrupts through IRQOUT0. See the *DAQ-STC Technical Reference Manual* for more information.

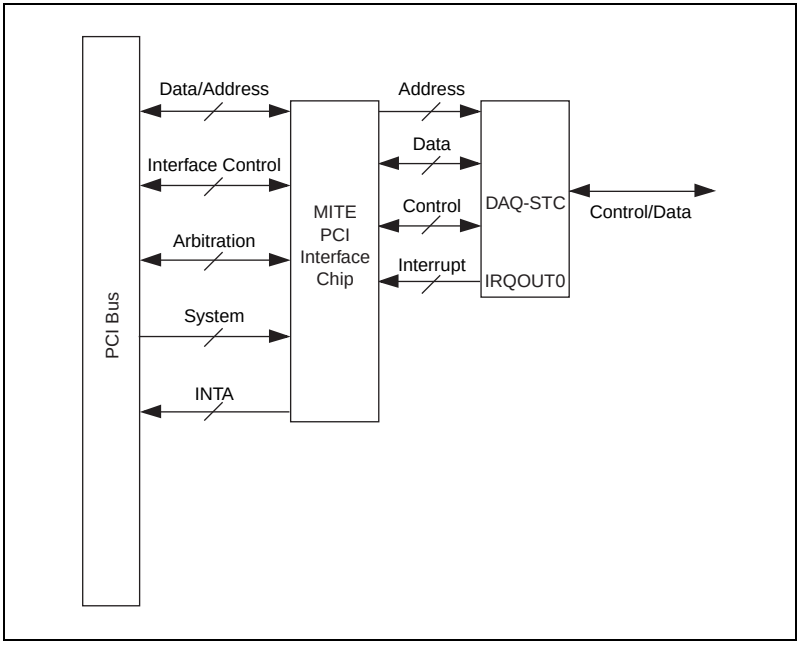



**Figure 2-6.** PCI Bus Interface Circuitry Block Diagram

## Analog Input and Timing Circuitry

The PCI E Series boards have 16 and 64 analog input channels and a timing core within the DAQ-STC that is dedicated to analog input operation. Figure 2-7 shows a general block diagram for the analog input circuitry.

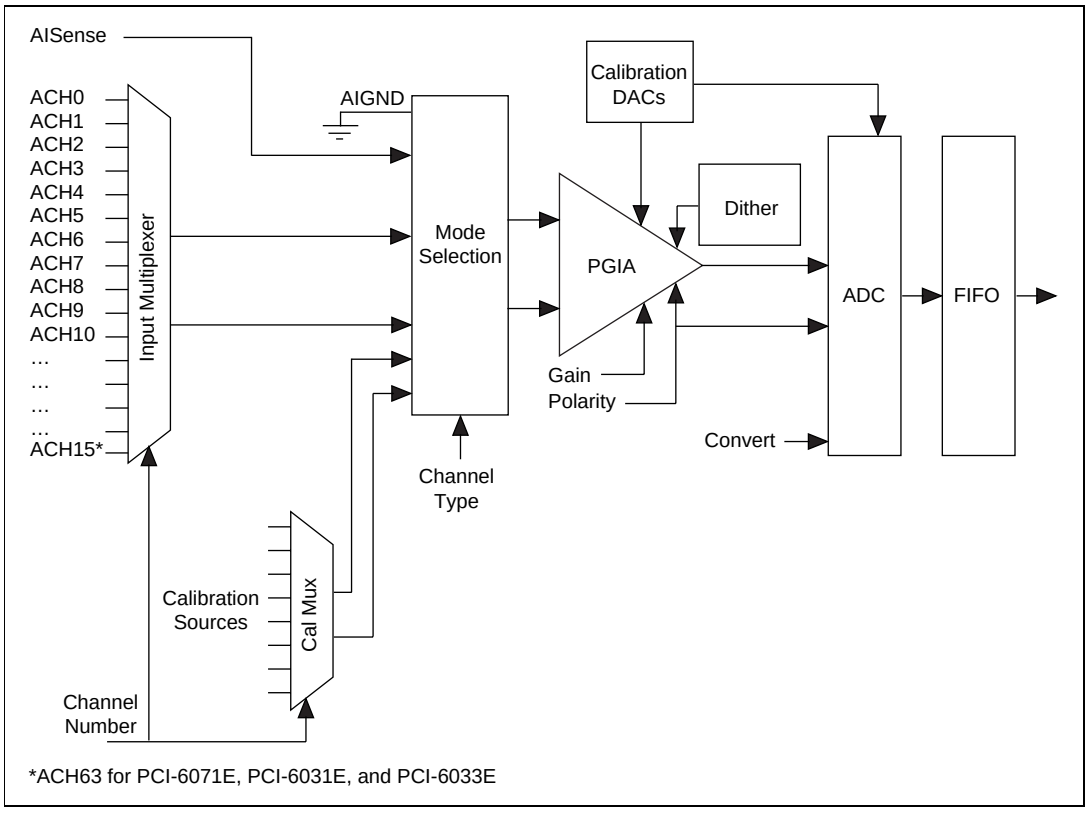



**Figure 2-7.** Analog Input and Data Acquisition Circuitry Block Diagram

## Analog Input Circuitry

The general model for analog input on the PCI E Series boards includes input multiplexer, multiplexer mode selection switches, a software-programmable gain instrumentation amplifier, calibration hardware, a sampling ADC, a 16-bit wide data FIFO, and a configuration memory.

The configuration memory defines the parameters to use for each conversion. Each entry in the configuration memory includes channel type, channel number, bank, gain, polarity, dither, general trigger, and last channel. The configuration memory is a 512-entry deep FIFO that is initialized prior to the start of the acquisition sequence. It can be incremented after every conversion, allowing the analog input configuration to vary on a per conversion basis. Once the FIFO is empty,

the DAQ-STC asserts the FIFO retransmit signal, which restores the FIFO data to its original state.

The channel type field indicates the resource type to be used during the conversion and controls the multiplexer mode selection switches. These resources include calibration channels, analog input channels in differential, referenced single-ended, or nonreferenced single-ended mode, or a ghost channel. The ghost channel type indicates that a conversion should occur but that the data should not be stored in the data FIFO. This type is useful for multirate scanning, which is described later in this chapter.

The channel number indicates which channel of the specified type will be used during the conversion, while the bank field indicates which bank of 16 channels is active. This bank field is used on boards that have more than 16 channels. These bits control the input multiplexers.

The programmable gain instrumentation amplifier (PGIA) serves two purposes on the PCI E Series boards. The PGIA applies gain to the input signal, amplifying an analog input signal before sampling and conversion to increase measurement resolution and accuracy. This gain is determined by the gain field in the configuration memory. It also provides polarity selection for the input signal, which is also controlled by the configuration memory. In unipolar mode, the input range includes only positive voltages. In bipolar mode, the input signal may also be a negative voltage. The PGIA provides gains shown in Table 2-1.

**Table 2-1.** PGIA Gain Set Verses Board

| Gain | PCI-MIO-16E-1<br>PCI-MIO-16E-4<br>PCI-6071E<br>PCI-6052E | PCI-MIO-16XE-50 | PCI-MIO-16XE-50<br>PCI-6031E<br>PCI-6032E<br>PCI-6033E | PCI-6023E<br>PCI-6024E<br>PCI-6025E |
|---|---|---|---|---|
| 0.5 | ✓ | — | — | ✓ |
| 1 | ✓ | ✓ | ✓ | ✓ |
| 2 | ✓ | ✓ | ✓ | — |
| 5 | ✓ | — | ✓ | — |
| 10 | ✓ | ✓ | ✓ | ✓ |
| 20 | ✓ | — | ✓ | — |

**Table 2-1.** PGIA Gain Set Verses Board (Continued)

| Gain | PCI-MIO-16E-1<br>PCI-MIO-16E-4<br>PCI-6071E<br>PCI-6052E | PCI-MIO-16XE-50 | PCI-MIO-16XE-50<br>PCI-6031E<br>PCI-6032E<br>PCI-6033E | PCI-6023E<br>PCI-6024E<br>PCI-6025E |
|---|---|---|---|---|
| 50 | ✓ | — | ✓ | — |
| 100 | ✓ | ✓ | ✓ | ✓ |

The dither circuitry adds approximately 0.5 LSB rms of white Gaussian noise to the signal being converted by the ADC. This addition is useful for applications, such as calibration, involving averaging to increase the resolution of the board to more than the resolution of the ADC. In such applications, which are often lower frequency in nature, adding the dither decreases noise modulation and improves differential linearity. Dither should be disabled for high-speed applications not involving averaging because it would only add noise. When taking DC measurements, such as when calibrating the board, you should enable dither and average about 1,000 points to take a single reading. This process removes the effects of quantization, reduces measurement noise, and improves resolution. Notice that dither cannot be disabled on the PCI-MIO-16XE-50, PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, PCI-6052E, or PCI-6033E.

The last channel bit is used to indicate that this is the last conversion in a scan. The DAQ-STC will end the scan on the conversion with this bit set.

The PCI E Series boards use sampling, successive approximation ADCs with 12 or 16 bits of resolution with maximum conversion rates between 50 µs and 800 ns. The converter can resolve its input range into 4,096 different steps for the 12-bit ADC and 65,536 for the 16-bit ADC. The input range of the 12-bit boards is ±5 V in bipolar mode and 0 to +10 V in unipolar mode. These modes correspond to ranges of –2,048 to 2,047 in unipolar mode and 0 to 4,095 in bipolar mode. The input range of the 16-bit boards is ±10 V in bipolar mode and 0 to +10 V in unipolar mode. These modes correspond to ranges of –32,768 to 32,767 in bipolar mode and 0 to 65,535 in unipolar mode.

The PCI E Series boards include a 16-bit wide FIFO to buffer the analog input data. This buffering will increase the maximum rate that the analog input can sustain during continuous acquisition. The FIFO is 2,048 words deep on the PCI-MIO-16XE-50, and 512 words deep on the other PCI E Series boards. The DAQ-STC shifts the data into the FIFO from the ADC when the conversion is complete. This buffering allows the ADC to begin a new conversion even though the data has not yet been read from the

board. This buffering also provides more time for the software or DMA to respond and read the analog input data from the board. If the FIFO is full and another conversion completes, an error condition called *FIFO overflow* occurs and the data from that conversion is lost. The FIFO *not empty*, *half-full*, and *full* flags are available to generate interrupts or DMA requests for the data transfer.

Measurement reliability is assured through the onboard calibration circuitry of the board. This circuitry uses an internal, stable 5 V reference that is measured at the factory against a higher accuracy reference; its value is then stored in the EEPROM. With this stored reference value, the board can be recalibrated at any time under any number of different environmental conditions in order to remove errors caused by time and temperature drift. The EEPROM stores calibration constants that can be read and then written to calibration DACs that adjust input offset, output offset, and gain errors associated with the analog input section. When the board leaves the factory, the upper one-fourth of the EEPROM is protected and cannot be overwritten. The lower three-fourths is unprotected, and the top fourth of that can be used to store alternate calibration constants for the different conditions under which you use the board.

## Data Acquisition Timing Circuitry

This section describes the different methods of acquiring A/D data from a single channel or multiple channels.

From this section through the end of this manual, you are assumed to have a working knowledge of the DAQ-STC features. These features are explained in the *DAQ-STC Technical Reference Manual*. If you have not read the functional description of each DAQ-STC module, you must do so before completing this register-level programmer manual.

### Single-Read Timing

To acquire data from the ADC, initiate a single conversion and read the resulting value from the ADC FIFO buffer after the conversion is complete. You can generate a single conversion in three different ways—apply an active pulse to the CONVERT* pin of the I/O connector, generate a falling edge on the sample-interval counter of the DAQ-STC, or strobe the

appropriate bit in a register in the PCI E Series register set. Any one of these operations will generate the timing shown in Figure 2-8.

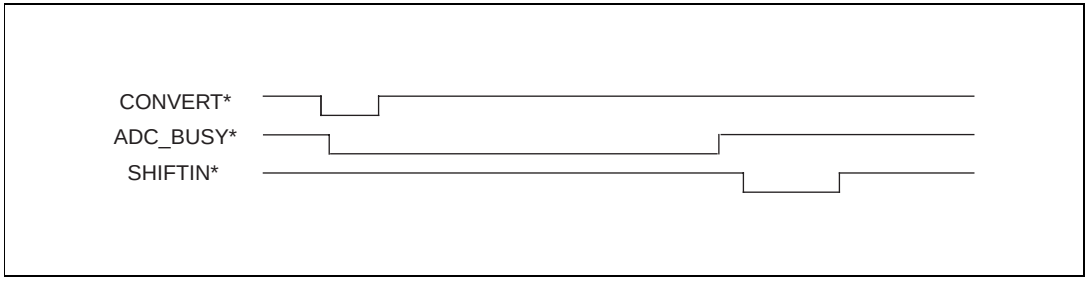



**Figure 2-8.** ADC Timing

When SHIFTIN* shifts the ADC value into the ADC FIFO buffer, the AI_FIFO_Empty_St bit in the status register is cleared, which indicates that valid data is available to be read. Single conversion timing of this type is appropriate for reading channel data on an *ad hoc* basis. However, if you need a sequence of conversions, the time interval between successive conversions is not constant because it relies on the software to generate the conversions. For finely timed conversions that require triggering and gating, you must program the boards to automatically generate timed signals that initiate and gate conversions. This is known as a data acquisition (DAQ) sequence.

## Data Acquisition Sequence Timing

The following counters are used for a data acquisition sequence:

- Scan interval (SI) 24 bits
- Sample interval (SI2) 16 bits
- Divide by (DIV) 16 bits
- Scan counter (SC) 24 bits

This section presents a concise summary of only the most important features of your board. For a complete description of all the analog input modes and features of the PCI E Series boards, refer to the *DAQ-STC Technical Reference Manual.*

The most basic timing signal in the analog input model is the CONVERT* signal. A group of precisely timed CONVERT* pulses is a SCAN. The sequence of channels selected in each conversion in a SCAN is programmed in the configuration memory prior to starting the operation. The SI2 counter is a 16-bit counter in the DAQ-STC. This counter

determines the interval between CONVERT* pulses. It can be programmed for a maximum interval of 3.3 ms and a minimum interval of 50 ns. If alternate slow timebases are used, the maximum interval is 0.65 s. Each time the counter reaches terminal count (TC), a CONVERT* pulse is generated. Alternatively, CONVERT* pulses could be given externally.

A SCAN sequence is started by the START pulse, which is generated by the TC of the SI counter. This counter is a 24-bit counter that determines the time between the start of each SCAN. The minimum duration is 50 ns and the maximum duration is 0.8 s when the internal 20 MHz timebase is used. If the internal 100 kHz timebase is used, the maximum is 167 s. The START pulse triggers the SI2 counter to generate CONVERT* pulses. With each conversion, the configuration memory advances by one and selects the next set of analog input conditions—channel number, gain, polarity, etc. A STOP pulse ends the SCAN sequence. This STOP could be generated in two ways—either by using the LASTCHANNEL bit in the configuration memory or by programming the 16-bit DIV counter to count the number of conversions per SCAN and using the terminal count of the DIV counter as a STOP pulse.

The SC is a 24-bit counter that counts the number of scans. The data acquisition sequence can be programmed to stop when the terminal count of this counter is reached. Notice that the START and STOP signals could also be supplied externally.

Example 1: To acquire 50 scans, with each scan consisting of one sample on channel 0 at gain 50, one sample on channel 5 at gain 2, and one sample on channel 3 at gain 10, with a SCAN interval of 100 µs and a sample interval of 10 µs, program your configuration memory as follows:

1. Channel 0, gain 50
2. Channel 5, gain 2
3. Channel 3, gain 10, last channel

You should program SI2 for 10 µs, SI for 100 µs, and SC for 50 scans.

Figure 2-9 shows the timing for each scan in Example 1.

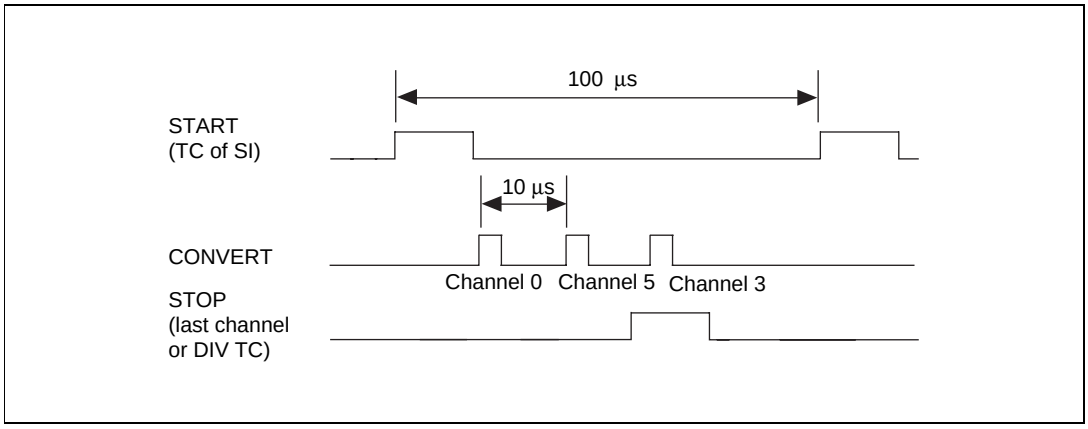



**Figure 2-9.** Timing of Scan in Example 1

The START pulse starts each scan. The first CONVERT pulse samples channel 0, the second CONVERT pulse samples channel 5, and the third CONVERT pulse samples channel 3. The STOP pulse ends the scan.

Example 1 allows you to sample all three channels at a rate of 10 kS/s per channel (100 µs sample interval period). To achieve different rates for different channels, you must do multirate scanning.

### Multirate Scanning without Using Ghost

Example 2: To sample channel 0 at 10 kS/s and channel 1 at 5 kS/s, both at gain 1 with 50 scans, program the configuration memory as follows:

1. Channel 0, gain 1
2. Channel 1, gain 1, last channel
3. Channel 0, gain 1, last channel

Program SI for 100 µs, SI2 for 10 µs. Figure 2-10 shows the timing sequence for two scans.

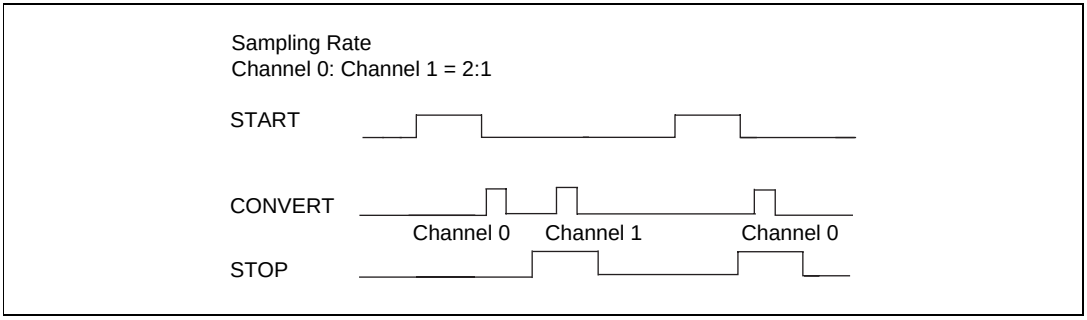



**Figure 2-10.** Multirate Scanning of Two Channels

This sequence of two scans is repeated 25 times to complete the acquisition. Notice that channel 0 is sampled once every 100 µs. Hence, its sampling rate is 10 kS/s, whereas channel 1 is sampled once every 200 µs. Its rate is 5 kS/s. Similarly, you could implement any 1:*x* ratio of sampling rates. The effective scan interval of the slower channel will be at $\frac{1}{x}$ the rate of the faster channel. This implementation requires *x* scan sequences in the configuration memory.

Also, you can implement a 1:1:*x* or 1:*x*:*mx* ratio for three channels, where *m* is a non-negative integer. Figures 2-11, 2-12, and 2-13 show timing sequences for different ratios. In these figures, the numbers above the CONVERT pulses indicate the channels sampled in that conversion.

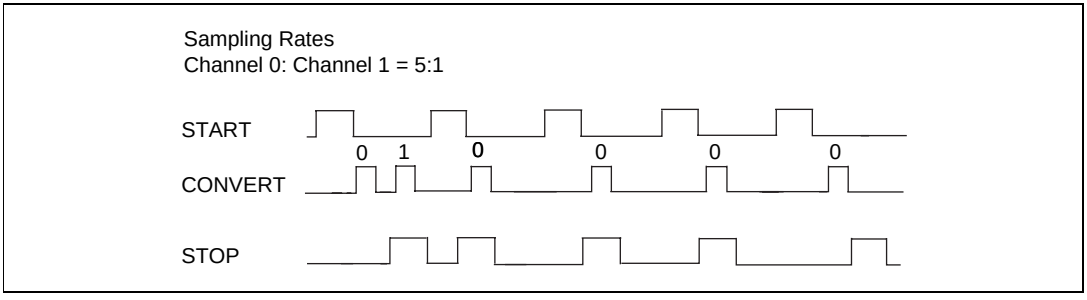



**Figure 2-11.** Multirate Scanning of Two Channels with 1:*x* Sampling Rate

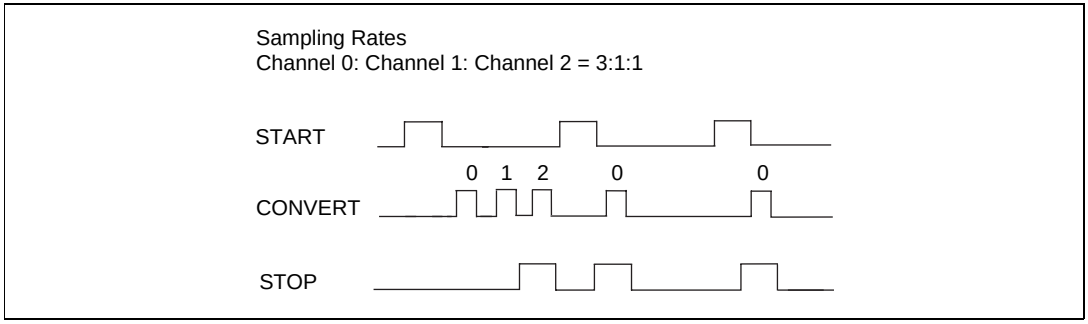



**Figure 2-12.** Multirate Scanning of Two Channels with 3:1:1 Sampling Rate

Here, channel 0 is sampled three times, whereas channels 1 and 2 are sampled once every three scans.

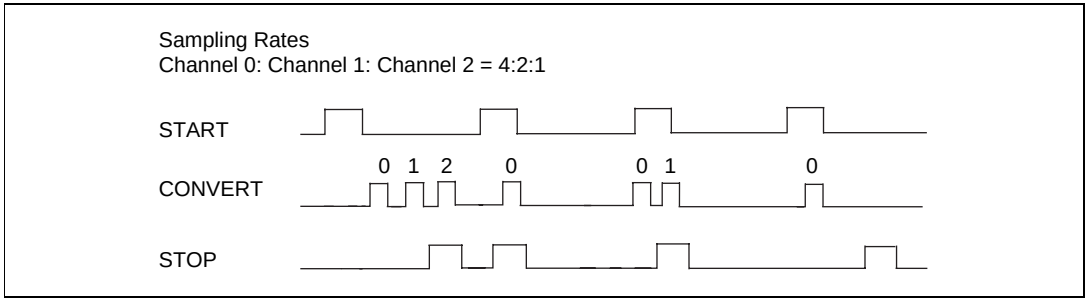



**Figure 2-13.** Multirate Scanning of Three Channels with 4:2:1 Sampling Rate

### Multirate Scanning Using Ghost

If the ghost option in the configuration memory is set, that conversion occurs but the data is not stored in the analog input FIFO. In other words, a conversion is performed and the data is thrown away. By using this option, multirate scanning with ratios such as x:y are possible, within the limits imposed by the size of the configuration memory.

Figures 2-14 and 2-16 illustrate the advantages of using the ghost feature. Figure 2-15 shows Example 3 timing, and Figure 2-16 shows the same example using ghost.

Example 3: channel 1: channel 0 = 2:3 (without ghost).

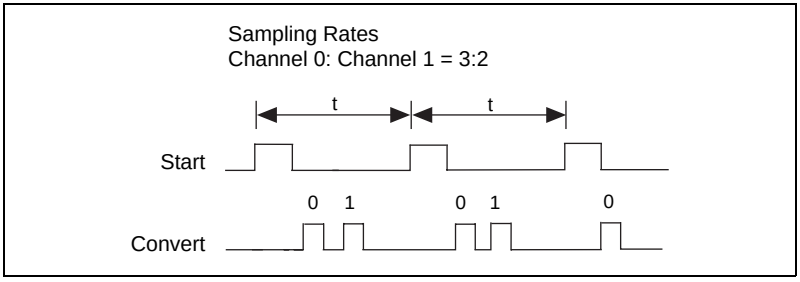



**Figure 2-14.** Multirate Scanning without Ghost

In Example 3, channel 0 is sampled correctly. Although channel 1 is sampled twice, it does not yield a 50% duty cycle. This type of acquisition will result in imprecise rates. Figure 2-15 shows the relative occurrences of convert pulses in Figure 2-14.

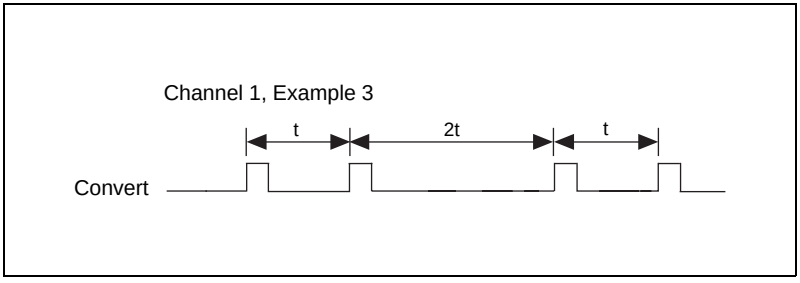



**Figure 2-15.** Occurrences of Conversion on Channel 1 in Example 3

To rectify the problem, use ghost as illustrated in Figure 2-16.

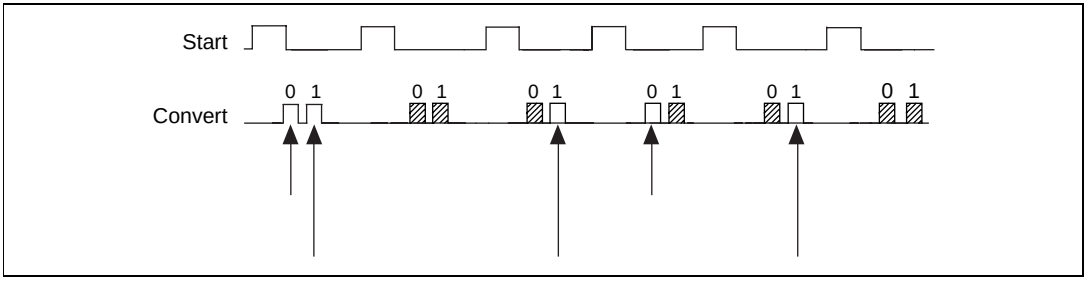



**Figure 2-16.** Successive Scans Using Ghost

The shaded conversions are ghost conversions. The short arrows indicate channel 0 samples and the long arrows indicate channel 1 samples that are actually stored in the FIFOs.

Table 2-2 shows what the configuration memory should look like.

**Table 2-2.** Analog Input Configuration Memory

| Channel | Ghost | Last Channel |
|---|---|---|
| 0 | — | — |
| 1 | — | last channel |
| 0 | ghost | — |
| 1 | ghost | last channel |
| 0 | ghost | — |
| 1 | — | last channel |
| 0 | — | — |
| 1 | ghost | last channel |
| 0 | ghost | — |
| 1 | — | last channel |
| 0 | ghost | — |
| 1 | ghost | last channel |

Now both channel 0 and channel 1 are sampled with 50% duty cycle.

## Posttrigger and Pretrigger Acquisition

Whereas a data acquisition operation normally ends when the SC counts down to zero, it can be initiated either through software, by strobing a bit in a control register in the DAQ-STC, or by suitable external triggers.

There are two internal trigger lines—START1 and START2. These appear at the connector on pins called PFI0/Trig1 and PFI1/Trig2, respectively. START1 is used as a trigger line in the posttrigger mode. Since the START1 pulse can be generated through software by strobing a bit, all of the examples discussed so far can be generally categorized as posttrigger acquisition. In the classic posttrigger mode case, the data acquisition circuitry is armed by software but does not start acquiring data until a pulse is given on the START1 line. Then the acquisition starts and ends when the SCAN counter counts down to zero.

In the pretrigger mode, data is acquired before and after the trigger. In this mode, both START1 and START2 lines are used. There are two counts for

the SCAN counter—the pretrigger count and the posttrigger count. To begin with, the pretrigger count is loaded into the SC. Acquisition is then started through either software by strobing a bit, or through hardware by externally pulsing the START1 pin of the connector. Acquisition then starts and data is acquired. When the SC counts down to zero, the scans continue and data still gets acquired. During this time, the board waits for a START2 pulse but the SC does not count. Also, the SC gets loaded with the posttrigger count. When a START2 pulse is received, the SC once again starts counting. When it reaches zero, the operation ends. Refer to the *Acquisition Level Timing and Control* section of Chapter 2 in the *DAQ-STC Technical Reference Manual* for timing examples of pretrigger and posttrigger modes.

Variations:

- Posttrigger and pretrigger modes can be retriggerable. This means that after one posttrigger or pretrigger operation is over, the SC gets reloaded and the board sits waiting for an additional START1. This can continue indefinitely and can be disabled through software.
- A special mode in the DAQ-STC allows continuous software-initiated acquisition to continue indefinitely. In this mode, the SC gets reloaded each time it counts down to zero. The acquisition can be stopped by disarming the SC. When the SC counts down to zero, acquisition stops.

## Analog Triggering

All PCI E Series boards except the PCI-MIO-16XE-50, PCI-6023E, PCI-6024E, and PCI-6025E have an analog trigger in addition to the digital triggers. To use analog triggering to start an acquisition sequence, select either the PFI0/Trig1 input on the I/O connector or one of the analog input pins. An analog input pin allows you to apply gain to the external signal for more flexible triggering conditions. The INT/EXTTRIG bit in the Misc. Command Register selects the source. PFI0/Trig1 pin has an input voltage range of ±10 V.

An 8-bit serial DAC sets each of the high and low thresholds in the PCI-MIO-16E-1, PCI-6071E, and the PCI-MIO-16E-4. For the PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E, a 12-bit serial DAC sets each of the high and low thresholds. These thresholds are within plus/minus full scale. The selected input is compared against each of these thresholds by a comparator. The outputs of the comparators are connected to the DAQ-STC analog trigger inputs 0 and 1. Within the DAQ-STC, these signals can be routed to any of the internal timing signals.

You can trigger on either a positive or negative slope or on absolute values.

Refer to the *DAQ-STC Technical Reference Manual* and the *NI-DAQ Function Reference Manual* for more information on analog triggering. For a detailed description of these modes, and timing diagrams, and for a description of other modes not discussed here, refer to the *DAQ-STC Technical Reference Manual*.

## Analog Output and Timing Circuitry

The PCI E Series boards (except the PCI-6023E, PCI-6032E and PCI-6033E) have two analog output channels and a timing core within the DAQ-STC that is dedicated to analog output operation. Figure 2-17 shows a general block diagram for the analog output circuitry.

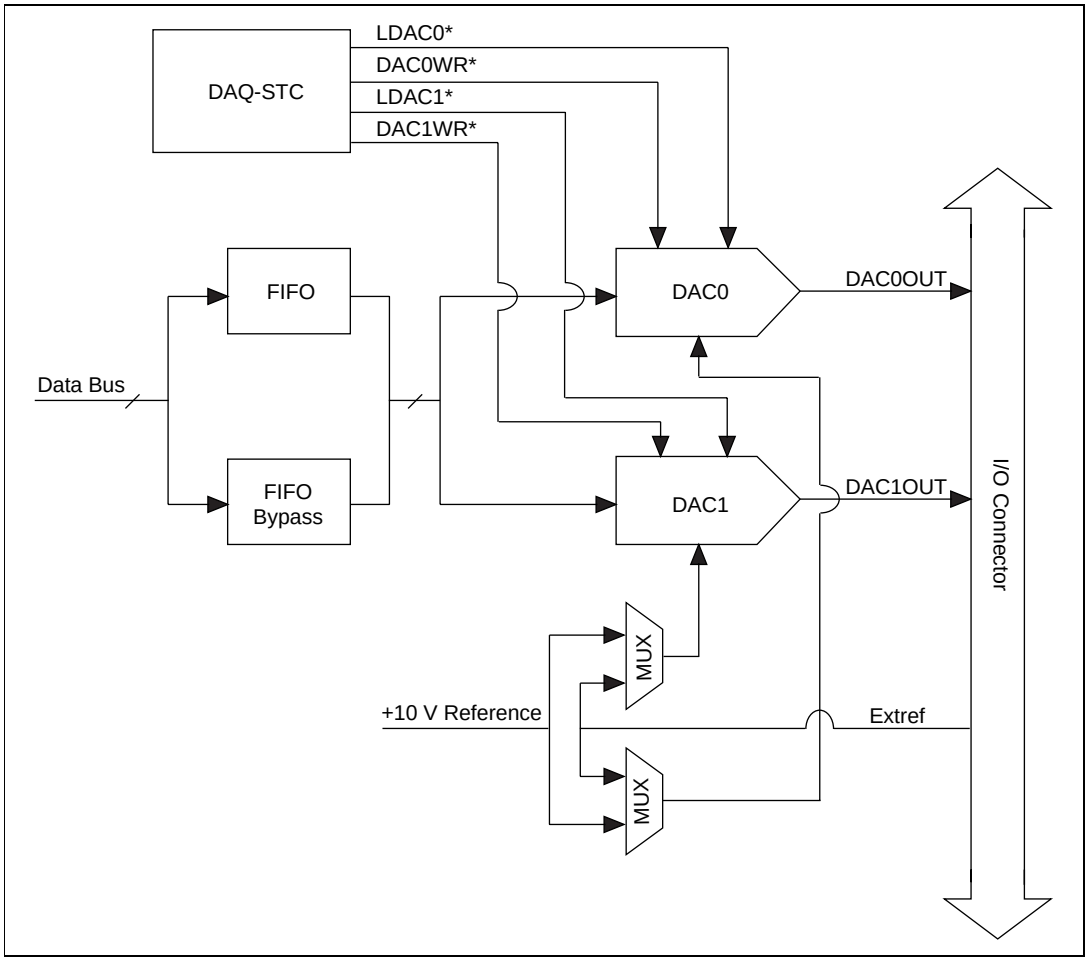



**Figure 2-17.** Analog Output Circuitry Block Diagram

## Analog Output Circuitry

The general model for analog output on the PCI E Series boards includes two channels of double-buffered analog output with programmable polarity, reference source, and reglitching circuit, as well as a FIFO to buffer the data. However, not all of the PCI E Series boards contain every one of these features.

Each analog output channel contains a 12-bit DAC, an amplification stage, and an onboard voltage reference, except for the PCI-MIO-16XE-10, PCI-6052E, and PCI-6031E, which have a 16-bit DAC. The output voltage will be proportional to the voltage reference (Vref) multiplied by the digital code loaded into the DAC. Note that the output will be set to 0 V on power up. The polarity, reference source, and reglitching circuit are all configured in the AO Configuration Register.

The voltage reference source for each DAC is selectable from the onboard reference or a voltage supplied at the EXTREF pin on the I/O connector, except for the PCI-MIO-16XE-50, PCI-MIO-16XE-10, PCI-6024E, PCI-6025E, and PCI-6031E which only supports the onboard reference. The onboard reference is fixed at +10 V. The external reference can be either a DC or an AC signal. If you apply an AC reference, the analog output channel acts as a signal attenuator and the AC signal appears at the output attenuated by the digital code. For unipolar output the voltage is simply attenuated. Four quadrant multiplication occurs in bipolar output, where the signal will not only be attenuated but also inverted for negative digital codes.

The DAC output can be configured to produce either a unipolar or bipolar output range, except for the PCI-MIO-16XE-50, which supports only bipolar output. A unipolar output has an output range of 0 to +Vref –1 LSB V. A bipolar output has an output voltage range of –Vref to Vref –1 LSB V). For unipolar output, the data written to the DAC is interpreted in straight binary format. For bipolar output, the data is interpreted as two’s complement format. One LSB is the voltage increment corresponding to an LSB change in the digital code word. For a 12-bit DAC, 1 LSB = (Vref)/4,096 in unipolar mode, and 1 LSB = (Vref)/2,048 in bipolar mode. For a 16-bit DAC, 1 LSB = (Vref)/65,536 in unipolar mode, and 1 LSB = (Vref)/32,768 in bipolar mode.

Using the 12-bit DAC and onboard 10 V reference will produce an output voltage range of 0 to 9.9976 V in steps of 2.44 mV for unipolar output and an output voltage range of –10 to +9.9951 V in steps of 4.88 mV for bipolar operation. Using 16-bit DAC and onboard 10 V reference will produce an output range of 0 to 9.9998 V in steps of 153 µV for unipolar output and an

output voltage range of –10 to 9.9997 V in steps of 305 μV for bipolar operation.

In normal operation, a DAC output will glitch whenever it is updated with a new value. The glitch energy differs from code to code and appears as distortion in the frequency spectrum. Each analog output of the PCI-MIO-16E-1, PCI-6052E, and PCI-6071E contains a reglitch circuit that generates uniform glitch energy at every code rather than large glitches at the major code transitions. This uniform glitch energy appears as a multiple of the update rate in the frequency spectrum. Notice that this reglitch circuit does not eliminate the glitches; it only makes them more uniform in size.

The PCI-MIO-16E-1, PCI-MIO-16XE-10, PCI-6031E, PCI-6052E, and PCI-6071E include 2 kword-deep FIFOs to buffer the analog output data. The PCI-MIO-16E-4 has 512 word-deep FIFOs. This buffering will increase the maximum rate that the analog output can sustain for waveform generation. It can also be used to store a complete waveform which can be output repetitively without any further data transfer to the FIFO. The PCI-MIO-16XE-50, PCI-6024E, and PCI-6025E has a zero-depth virtual FIFO.

## Analog Output Timing Circuitry

This section describes the different methods of setting the analog output voltage, including single-point updating and waveform generation. The DAQ-STC provides the timing signals necessary to write to the DACs and update them.

The DACs are double buffered and can be individually configured for immediate update or timed update mode. In immediate update mode, the double buffering of the DAC is disabled and the value written to the DAC will appear immediately on the output after the write completes. In the timed update mode, the double buffering is enabled. When double buffering is enabled, writing the digital value loads that value into the buffer stage of the DAC. When an update signal is received, the analog output of the DAC changes.

### Single-Point Output

The data values can be written directly to the DACs under software control without the use of the timing engine provided in the DAQ-STC. This is typically useful for setting the analog outputs to DC levels, where precise timing of the output change is not important. Writing directly to the DACs is accomplished by writing the desired value to the DAC<0..1> Direct Data

Register. This action will store the value in the first buffer of the DAC. If the DAQ-STC is programmed for immediate updating mode, the value will also be applied to the analog output. If the DAQ-STC is programmed for timed updating mode, the appropriate update signal, LDAC0* or LDAC1*, must be asserted by writing to the appropriate DAQ-STC register.

## Waveform Generation

The timing engine in the DAQ-STC can be used for waveform generation, where the update signals connected to the DACs can be generated at precise intervals. In this model, the DAQ-STC will generate the update signals and then transfer the data values for the next update to the first buffer of the DACs.

The update interval counter (UI) is 24 bits wide and generates the update pulses. These update pulses can be routed to the DACs. The update counter (UC) is 24 bits wide and counts the number of updates. The value loaded into this counter defines a buffer. The buffer counter (BC) determines how many times a buffer should be repeated. These counters define an analog output sequence.

The DAQ-STC will transfer the data values for the next update after each update pulse. This data transfer will be from the analog output FIFO. Some of the boards have large 2 kword FIFOs and some have 512 word FIFOs; others are zero-deep virtual FIFOs. The large FIFOs are true FIFOs, where data can be written to the FIFOs as long as they are not full, and can be read from when they are not empty. In this case, the DAQ-STC will transfer the data from the FIFO to the DACs when the FIFO is not empty. If the FIFO is empty, the DAQ-STC will delay the transfer until more data is written into the FIFO.

The zero-deep virtual FIFOs are not true FIFOs, but they do provide a software compatible method for waveform generation. When the DAQ-STC generates an update signal and is ready to transfer data, the board will clear the FIFO full flag. This FIFO full flag can be used by interrupts or DMA to transfer data to the virtual FIFO. The virtual FIFO will not actually buffer the data, instead it will transfer it directly to the first buffer of the destination DAC. After the data has been transferred, the board will set the FIFO full flag, indicating that no more data is required. Note that the half-full and empty flags do not provide useful information for data transfer and should not be used.

The large FIFOs can also be used to generate repetitive waveforms at very high speeds and without using any bus bandwidth. The FIFOs can be loaded with the desired waveform and the DAQ-STC can be programmed

to retransmit the same FIFO data to the DACs over and over. When the FIFO is empty, the retransmit signal is asserted, which restores that FIFO data to its original state.

When both DACs are used in waveform generation, the data in the FIFO is interleaved; in other words, every alternate sample belongs to one DAC.

## Digital I/O Circuitry

The PCI E Series boards have eight digital I/O lines. Each of the eight digital lines can be individually programmed to be input or output, if used in parallel. For serial data transfer, DIO4 is used as the serial data in pin, and DIO0 is used as the serial data out pin.

You can use handshaking with the EXTSTROBE* pin to do either parallel or serial data transfer. Refer to the *DAQ-STC Technical Reference Manual* for more details on the DIO features.

The external strobe signal EXTSTROBE* is a general-purpose strobe signal. Software can set the output of the EXTSTROBE* pin to either a high or low state. EXTSTROBE* is not necessarily part of the digital I/O circuitry, but is included here because you can use it to latch digital output from the PCI E Series into an external device.

## Timing I/O Circuitry

The DAQ-STC has two 24-bit general purpose counter/timers. The counters are numbered *0* and *1* and are diagrammed as shown in Figure 2-18.

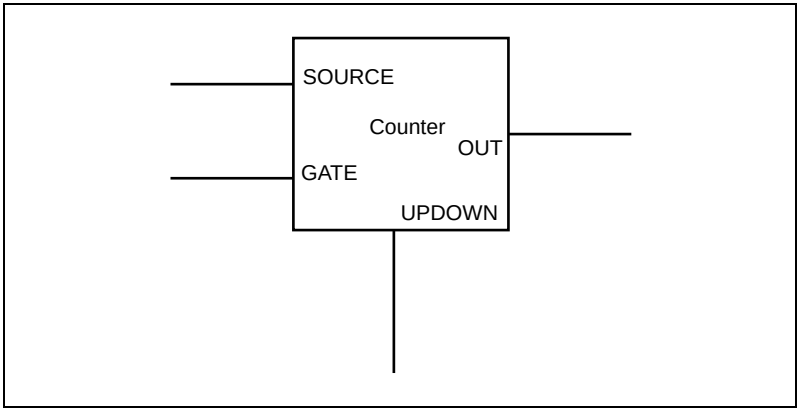



**Figure 2-18.** DAQ-STC Counter Diagram

The counter counts the transitions, or edges, on the SOURCE line when the GATE is enabled and generates timing signals at its OUT output pin. The UPDOWN pin determines the direction of counting. Active polarities of these pins are software selectable in the DAQ-STC. Notice that on the PCI E Series boards only the SOURCE, GATE, and OUT pins are brought out to the I/O connector. The UPDOWN pin for counter 0 is internally connected to DIO6, and to DIO7 for counter 1. To drive the UPDOWN pin from the connector, you must tri-state the corresponding DIO line.

The SOURCE of these counters can be selected to be one of the 10 PFI lines, the seven RTSI lines, the internal 20 MHz or 100 kHz timebases, or the TC of the other counter. With the last option, the two counters can be concatenated. Similarly, the GATE can also be selected from a variety of different sources. The sources for both of these signals are described in the *DAQ-STC Technical Reference Manual*.

## RTSI Bus Interface Circuitry

The PCI E Series is interfaced to the National Instruments RTSI bus. The RTSI bus has seven trigger lines and a system clock line. You can wire all National Instruments PCI E Series boards with RTSI bus connectors inside the PCI bus computer and share these signals. Figure 2-19 shows a block diagram of the RTSI bus interface circuitry.

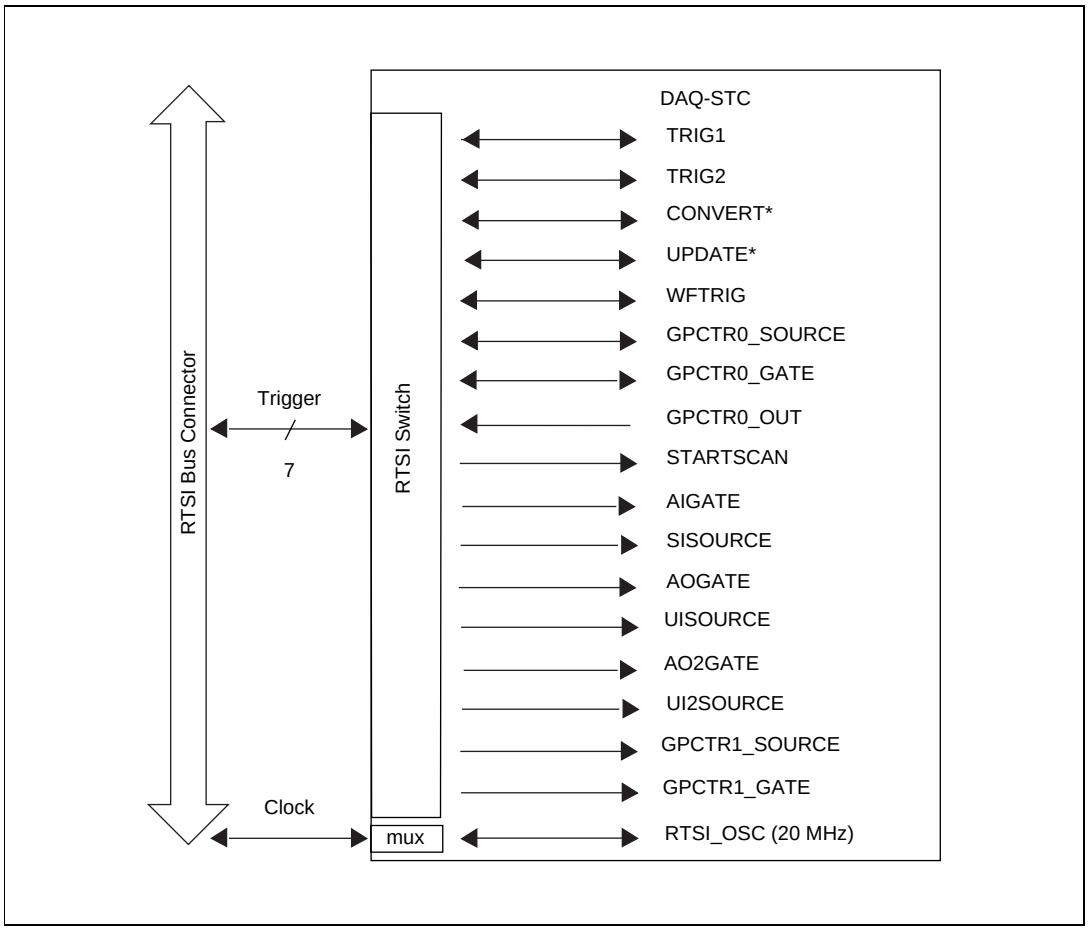



**Figure 2-19.** RTSI Bus Interface Circuitry Block Diagram

The RTSI functionality is provided in the DAQ-STC chip. This RTSI trigger module consists of seven 12-to-1 multiplexers that drive seven RTSI Trigger bus signals and four 8-to-1 multiplexers that drive the four RTSI board signals.

Any of the seven RTSI trigger bus signals can be driven by eight internally generated timing signals and the four RTSI board signals. Similarly, the four RTSI board signals can be driven by any of the RTSI trigger bus signals. Of the four RTSI board signals, only one is used. By programming certain bits in the DAQ-STC, you can drive the CONVERT pulse onto RTSI_BRD0 and then onto any of the TRIGGER lines.

# 3

# Register Map and Descriptions

This chapter describes in detail the address and function of each of the PCI E Series board control and status registers.

If you plan to use an application software package such as NI-DAQ, LabVIEW, or LabWindows/CVI with your PCI E Series board, you need not read this chapter. However, you gain added insight into your PCI E Series board by reading this chapter.

## Register Map

Table 3-1 shows the register map for the PCI E Series boards and gives the register name, the register offset address, the type of the register (read-only, write-only, or read-and-write), and the size of the register in bits. Obtain the actual register address by adding the appropriate register offset to the memory base address of the PCI E Series board.

Registers are grouped in the table by function. Each register group is introduced in the order shown in Table 3-1, then described in detail, including a bit-by-bit description.

The DAQ-STC has 180 different registers. The more frequently used registers have been given lower offset addresses and are shown in Table 3-1 as the DAQ-STC Register Group. The advantage of having lower offset addresses is that they can be accessed directly or through the DAQ-STC windowed mode. You can access the remaining registers in the DAQ-STC only in the windowed mode. In this mode, the address of the register is first written to the Window Address Register. The data to be written is then written to the Window Data Register. Using windowed mode has the advantage of reducing the address space of the board from 180 to 32 bytes. However, this reduced address space is at the cost of two write operations to write data to or two read operations to read data from a register.

**Table 3-1.** PCI E Series Register Map

| Register Name | Offset Address | | Type | Size |
|---|---|---|---|---|
| | Hex | Decimal | | |
| Misc Register Group | | | | |
| Serial Command | 0D | 13 | Write-only | 8-bit |
| Misc Command | 0F | 15 | Write-only | 8-bit |
| Status | 01 | 1 | Read-only | 8-bit |
| Analog Input Register Group | | | | |
| ADC FIFO Data Register | 1C | 28 | Read-only | 16-bit |
| Configuration Memory Low | 10 | 16 | Write-only | 16-bit |
| Configuration Memory High | 12 | 18 | Write-only | 16-bit |
| Analog Output Register Group | | | | |
| AO Configuration | 16 | 22 | Write-only | 16-bit |
| DAC FIFO Data | 1E | 30 | Write-only | 16-bit |
| DAC0 Direct Data | 18 | 24 | Write-only | 16-bit |
| DAC1 Direct Data | 1A | 26 | Write-only | 16-bit |
| DMA Control Register Group | | | | |
| AI AO Select | 09 | 9 | Write-only | 8-bit |
| G0 G1 Select | 0B | 11 | Write-only | 8-bit |
| DAQ-STC Register Group | | | | |
| Window Address | 0 | 0 | Read-and-write | 16-bit |
| Window Data | 2 | 2 | Read-and-write | 16-bit |
| Interrupt A Acknowledge | 4 | 4 | Write | 16-bit |
| Interrupt B Acknowledge | 6 | 6 | Write | 16-bit |
| AI Command 2 | 8 | 8 | Write | 16-bit |
| AO Command 2 | A | 10 | Write | 16-bit |
| G0 Command | C | 12 | Write | 16-bit |
| G1 Command | E | 14 | Write | 16-bit |
| AI Status 1 | 4 | 4 | Read | 16-bit |
| AO Status 1 | 6 | 6 | Read | 16-bit |
| G Status | 8 | 8 | Read | 16-bit |
| AI Status 2 | A | 10 | Read | 16-bit |
| AO Status 2 | C | 12 | Read | 16-bit |
| DIO Parallel Input | E | 14 | Read | 16-bit |

**Table 3-2.** PCI E Series Windowed Register Map

| Register Name | Offset Address | | Type | Size |
|---|---|---|---|---|
| | Hex | Decimal | | |
| FIFO Strobe Register Group | | | | |
| Configuration Memory Clear | 52 | 82 | Write-only | 16-bit |
| ADC FIFO Clear | 53 | 83 | Write-only | 16-bit |
| DAC FIFO Clear | 54 | 84 | Write-only | 16-bit |

## Register Sizes

Two different transfer sizes for read-and-write operations are available on the computer—byte (8-bit) and word (16-bit). Table 3-1 shows the size of each PCI E Series register. For example, reading the ADC FIFO Data Register requires a 16-bit (word) read operation at the selected address, whereas writing to the Misc Command Register requires an 8-bit (byte) write operation at the selected address. For proper board operation you must adhere to these register size accesses. Avoid performing a byte access on a word location or performing a word access on a byte location; these are invalid operations. The register sizes are very important.

# Register Descriptions

This section discusses each of the PCI E Series registers in the order shown in Table 3-1. Each register group is introduced, followed by a detailed bit description. The individual register description gives the address, type, word size, and bit map of the register, followed by a description of each bit.

The register bit map shows a diagram of the register with the MSB shown on the left (bit 15 for a 16-bit register, bit 7 for an 8-bit register), and the LSB shown on the right (bit 0). A square represents each bit and contains the bit name. An asterisk (*) after the bit name indicates that the bit is inverted (negative logic).

In many of the registers, several bits are labeled *Reserved*. When a register is read, these bits may appear set or cleared but should be ignored because they have no significance. When you write to a register, set these bits to zero.

## Misc Register Group

The three registers making up the Misc Register Group include two command registers that control the serial DACs, EEPROM, and analog trigger source, and one status register that includes EEPROM information.

Bit descriptions of the three registers making up the Misc Register Group are given on the following pages.

## Serial Command Register

The Serial Command Register contains six bits that control PCI E Series serial EEPROM and DACs. The contents of this register are cleared upon power up and after a reset condition.

**Address:** Base address + 0D (hex)

**Type:** Write-only

**Word Size:** 8-bit

**Bit Map:**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | SerDacLd2 | SerDacLd1 | SerDacLd0 | EEPromCS | SerData | SerClk |

| Bit | Name | Description |
|---|---|---|
| 7–6 | Reserved | Reserved—Always write 0 to these bits. |
| 5 | SerDacLd2 | Serial DAC Load2—This bit is used to load the third set of serial DACs with the serial data previously shifted into the DACs. |
| 4 | SerDacLd1 | Serial DAC Load1—This bit is used to load the second set of serial DACs with the serial data previously shifted into the DACs. |
| 3 | SerDacLd0 | Serial DAC Load0—This bit is used to load the first set of serial DACs with the serial data previously shifted into the DACs. |
| 2 | EEPromCS | EEPROM Chip Select—This bit controls the chip select of the onboard EEPROM used to store calibration constants. When EEPromCS is set, the chip select signal to the EEPROM is enabled. |
| 1 | SerData | Serial Data—This bit is the data for the onboard serial devices—the calibration EEPROM and the serial DACs. This bit should be set to the desired value prior to the active write to the SerClk bit. |
| 0 | SerClk | Serial Clock—This bit is the clock input to the onboard serial devices. In order to access these devices, this bit is used to clock data in or out as appropriate. See Chapter 5, *Calibration*, for more information. |

## Misc Command Register

The Misc Command Register contains one bit that controls the PCI E Series analog trigger source. The contents of this register are cleared upon power up and after a reset condition.

**Address:** Base address + 0F (hex)

**Type:** Write-only

**Word Size:** 8-bit

**Bit Map:**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Int/Ext Trig | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved |

| Bit | Name | Description |
|---|---|---|
| 7 | Int/Ext Trig | Internal/External Analog Trigger—This bit controls the analog trigger source. If this bit is set, the output of the PGIA2 is selected as the trigger source. If this bit is cleared, the TRIG1 signal from the I/O connector is selected as the trigger source. |
| 6–0 | Reserved | Reserved—Always write 0 to these bits. |

## Status Register

The Status Register is used to indicate the status of the calibration EEPROM output.

**Address:** Base address + 01 (hex)

**Type:** Read-only

**Word Size:** 8-bit

**Bit Map:**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | PROMOUT |

| Bit | Name | Description |
|---|---|---|
| 7–1 | Reserved | Reserved—Ignore returned bits. |
| 0 | PROMOUT | EEPROM Output Data—This bit reflects the serial output data of the serial EEPROM. |

## Analog Input Register Group

The three registers making up the Analog Input Register Group control the analog input circuitry and can be used to read the ADC FIFO contents. Reading the ADC FIFO Data Register location transfers data from the PCI E Series ADC data FIFO to the computer. Writing to the Configuration Memory Low and Configuration Memory High Register locations sets up channel configuration information for the analog input section. This information is necessary for single conversions as well as for single- and multiple-channel data acquisition sequences.

## ADC FIFO Data Register

The ADC FIFO Data Register returns the oldest ADC conversion value stored in the ADC FIFO. Reading the ADC FIFO removes that value and leaves space for another ADC conversion value to be stored. Values are shifted into the ADC FIFO whenever an ADC conversion is complete unless the GHOST bit is set in that entry of the Configuration Memory.

The ADC FIFO is emptied when all values it contains are read. The empty, half-full, and full flags from the ADC data FIFO are available in a status register in the DAQ-STC. These flags indicate when the FIFO is empty, half-full, or full, respectively. Whenever the FIFO is not empty, the stored data can be read from the ADC FIFO Data register.

The values returned by reading the ADC Data Register are available in two different binary formats: straight binary, which generates only positive numbers, or two's complement binary, which generates both positive and negative numbers. The binary format used is determined by the mode in which the ADC is configured. Following is the bit pattern returned for either format:

**Address:** Base address + 1C (hex)

**Type:** Read-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

| Bit | Name | Description |
|---|---|---|
| 15–0 | D<15..0> | Data bits 15 through 0—These bits are the result of the ADC conversion. The boards with a 12-bit ADC return values ranging from 0 to 4,095 decimal (0x0000 to 0x0FFF) when the ADC is in unipolar mode, and –2,048 to 2,047 decimal (0xF800 to 0x07FF) when the ADC is in bipolar mode. The boards with a 16-bit ADC return values ranging from 0 to 65,535 decimal (0x0000 to 0xFFFF) when the ADC is in unipolar mode and 32,768 to 32,767 decimal (0x8000 to 0x7FFF) when the ADC is in bipolar mode. The mode is controlled by the Unip/Bip bit in the Configuration Memory Low Register. |

## Configuration Memory Low Register

The Configuration Memory Low Register works with the Configuration Memory High Register to control the input channel selection multiplexers, gain, range, and mode settings. The values written to these registers are placed into the Configuration Memory, which is sequenced through during an acquisition. This register contains seven of these bits. The contents of the Configuration Memory are emptied by a control register in the DAQ-STC.

**Address:** Base address + 10 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| LastChan | Reserved | Reserved | GenTrig | Reserved | Reserved | DitherEn | Unip/Bip |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | Reserved | Gain2 | Gain1 | Gain0 |

| Bit | Name | Description |
|---|---|---|
| 15 | LastChan | Last Channel—This bit should be set in the last entry of the scan sequence loaded into the channel configuration memory. More than one occurrence of the LastChan bit is possible in the configuration memory list for the interval-scanning mode. For example, there can be multiple scan sequences in one memory list. |
| 14–13, 11–10, 7–3 | Reserved | Reserved—Always write 0 to these bits. |
| 12 | GenTrig | General Trigger—This bit synchronizes actions in the DAQ-STC with the scan list. When this bit is set, an active low trigger pulse is sent into the RTSI_BRD0 input of the DAQ-STC during the CONVERT signal. This trigger is used at the application level and can perform such actions as time stamping and starting waveform generation. |
| 9 | DitherEn | Dither Enable—This bit controls the dither circuitry feeding the analog input. If this bit is set, approximately ±0.5 LSB of white Gaussian noise is added to the input signal. This bit is reserved on the PCI-MIO-16XE-50, |

PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E and should be set to 1. Dither cannot be disabled on the PCI-MIO-16XE-50, PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E.

| | | |
|---|---|---|
| 8 | Unip/Bip | Channel Unipolar/Bipolar—This bit configures the ADC for unipolar or bipolar mode. When Unip/Bip is set, the ADC is configured for unipolar operation and values read from the ADC Data Register are in straight binary format. When Unip/Bip is clear, the ADC is configured for bipolar operation and values. The data values are two's complement and automatically sign extended. |
| 2–0 | Gain<2..0> | Channel Gain Select 2 through 0—These three bits control the gain settings of the input PGIA for the selected analog channel. The gains shown in Table 3-3 can be selected on the PCI E Series boards. |

**Table 3-3.** PGIA Gain Selection

| Gain<2..0> | Actual Gain |
|---|---|
| 000 | 0.5[1,2] |
| 001 | 1 |
| 010 | 2[3] |
| 011 | 5[1, 3] |
| 100 | 10 |
| 101 | 20[1, 3] |
| 110 | 50[1, 3] |
| 111 | 100 |

[1] Not supported on the PCI-MIO-16XE-50
[2] Not supported on the PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E
[3] Not supported on the PCI-6023E, PCI-6024E, and PCI-6025E.

## Configuration Memory High Register

The Configuration Memory High Register works with the Configuration Memory Low Register to control the input channel selection multiplexers, gain, range, and mode settings. This register contains nine of these bits. The contents of this register are cleared by a control register in the DAQ-STC.

**Address:** Base address + 12 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | ChanType2 | ChanType1 | ChanType0 | Reserved | Reserved | Reserved | Reserved |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved/ Bank 1 | Reserved/ Bank 0 | Chan3 | Chan2 | Chan1 | Chan0 |

| Bit | Name | Description |
|---|---|---|
| 15, 11–6 | Reserved | Reserved—Always write 0 to these bits. |
| 14–12 | ChanType<2..0> | Channel Type 2 through 0—These bits indicate which type of resource is active for the current entry in the scan list. The following table lists the valid channel types. |

| Chan Type<2..0> | Resource |
|---|---|
| 000 | Calibration |
| 001 | Differential |
| 010 | NRSE |
| 011 | RSE |
| 100 | X |
| 101 | Aux |
| 110 | X |
| 111 | Ghost |

| | | |
|---|---|---|
| 5-4 | Reserved | Reserved—Always write 0 to these bits (*except for the 6031E, 6033E, and 6071E*). |
| 5–4 | Bank<1..0> | Bank Select 1 through 0—These bits indicate which sets of resources are active for the current entry in the scan list. PCI-6031E, PCI-6033E and PCI-6071E segment the channels into the banks of 16. Not every resource uses all of the bank bits. For example, PCI-6031E uses four banks of channels for the DIFF, NRSE and RSE resources, but only one bank of CAL. |
| 3–0 | Chan<3..0> | Channel Select 3 through 0—These bits indicate which channel is active for the current resource in the scan list. Not every resource uses all 16 channels in a bank. Channel assignments for all PCI E Series or higher boards follow. |

**Table 3-4.** Calibration Channel Assignments

| **Chan Type<2..0> = CAL** | | | |
|---|---|---|---|
| **Chan<3..0>** | **PGIA(+)** | **PGIA(–)** | **Purpose** |
| 0000 | AOGND[1] | AOGND[1] | ADC Offset |
| 0001 | AOGND | AIGND | Ground Differential |
| 0010[2] | DAC0OUT | AOGND | DAC 0 Offset/Linearity |
| 0011[2] | DAC1OUT | AOGND | DAC 1 Offset/Linearity |
| 0100 | REF5V[1] | REF5V[1] | ADC Offset |
| 0101 | REF5V | AI GND | ADC Gain |
| 0110[2] | DAC0OUT | REF5V | DAC 0 Gain |
| 0111[2] | DAC1OUT | REF5V | DAC 1 Gain |
| 1000[2] | DAC1OUT | DAC1OUT | — |
| 1001-1111 | Reserved | Reserved | — |

[1] AIGND on the PCI-MIO-16XE-50, PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E

[2] Not valid for the PCI-6032E and PCI-6033E

**Table 3-5.** Differential Channel Assignments

| Chan Type<2..0> = DIFF | | |
|---|---|---|
| **Chan<3..0>** | **PGIA(+)** | **PGIA(–)** |
| 0000 | ACh0 | ACh8 |
| 0001 | ACh1 | ACh9 |
| 0010 | ACh2 | ACh10 |
| 0011 | ACh3 | ACh11 |
| 0100 | ACh4 | ACh12 |
| 0001 | ACh5 | ACh13 |
| 0110 | ACh6 | ACh14 |
| 0111 | ACh7 | ACh15 |
| 1xxx | Reserved | Reserved |

**Table 3-6.** Nonreferenced Single-Ended Channel Assignments

| Chan Type<2..0> = NRSE | | |
|---|---|---|
| **Chan<3..0>** | **PGIA(+)** | **PGIA(–)** |
| 0000 | ACh0 | AISense |
| 0001 | ACh1 | AISense |
| 0010 | ACh2 | AISense |
| 0011 | ACh3 | AISense |
| 0100 | ACh4 | AISense |
| 0101 | ACh5 | AISense |
| 0110 | ACh6 | AISense |
| 0111 | ACh7 | AISense |
| 1000 | ACh8 | AISense |
| 1001 | ACh9 | AISense |
| 1010 | ACh10 | AISense |
| 1011 | ACh11 | AISense |

**Table 3-6.** Nonreferenced Single-Ended Channel Assignments (Continued)

| Chan Type<2..0> = NRSE | | |
|---|---|---|
| **Chan<3..0>** | **PGIA(+)** | **PGIA(–)** |
| 1100 | ACh12 | AISense |
| 1001 | ACh13 | AISense |
| 1110 | ACh14 | AISense |
| 1111 | ACh15 | AISense |

**Table 3-7.** Referenced Single-Ended Channel Assignments

| Chan Type<2..0> = RSE | | |
|---|---|---|
| **Chan<3..0>** | **PGIA(+)** | **PGIA(–)** |
| 0000 | ACh0 | AIGround |
| 0001 | ACh1 | AIGround |
| 0010 | ACh2 | AIGround |
| 0011 | ACh3 | AIGround |
| 0100 | ACh4 | AIGround |
| 0101 | ACh5 | AIGround |
| 0110 | ACh6 | AIGround |
| 0111 | ACh7 | AIGround |
| 1000 | ACh8 | AIGround |
| 1001 | ACh9 | AIGround |
| 1010 | ACh10 | AIGround |
| 1011 | ACh11 | AIGround |
| 1100 | ACh12 | AIGround |
| 1001 | ACh13 | AIGround |
| 1110 | ACh14 | AIGround |
| 1111 | ACh15 | AIGround |

**Table 3-8.** Auxiliary Channel Assignments

| Chan Type<2..0> = AUX | | | |
|---|---|---|---|
| **Chan<3..0>** | **PGIA (+)** | **PGIA (–)** | **Purpose** |
| 0100 | TEMP | AIGND | Onboard temperature sensor.*<br>+100 mV = -40° C, +1.75 V = +125° C |
| 0000–0011<br>0101–1111 | Reserved | Reserved | — |
| * The temperature sensor is not supported on Revision D and earlier of the PCI-MIO-16XE-50. | | | |

**Table 3-9.** Channel Assignments

| Chan Type<2..0> = GHOST | |
|---|---|
| **Chan<3..0>** | **Purpose** |
| xxxx | Used to preserve timing for multirate sampling. No acquisition is performed for this scan entry. |

## Analog Output Register Group

**Note** ***The analog output register group is not valid for the PCI-6023E, PCI-6032E and PCI-6033E.***

The four registers making up the Analog Output Register Group access the two analog output channels, the analog output FIFO, and the analog output configuration. Data can be transferred to the DACs in one of two ways. Data can be directly sent to the DACs from the host computer, or buffered from the host by the DAC data FIFO.

## AO Configuration Register

The AO Configuration Register contains five bits that control the PCI E Series analog output configuration. The contents of this register are cleared upon power up and after a reset condition.

**Address:** Base address + 16 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | Reserved | DACSel |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Reserved | Reserved | Reserved | Reserved | GroundRef | ExtRef | ReGlitch | BipDac |

| Bit | Name | Description |
|---|---|---|
| 15–9, 7–4 | Reserved | Reserved—Always write 0 to these bits. |
| 8 | DACSel | DAC Select—This bit indicates which DAC is the destination for the configuration bits in this register. DAC0 will be selected when this bit is cleared, and DAC1 will be selected when set. |
| 3 | GroundRef | Ground Reference—This bit connects the reference for both DACs to ground when this bit is set. This is useful for calibration of the DAC linearity. This bit is not currently implemented as a separate selection for the two DACs. Therefore, its state is determined by the last value written to this bit field. This bit is reserved on the PCI-MIO-16XE-50, PCI-MIO-16XE-10, and PCI-6031E. It should be set to 0. |
| 2 | ExtRef | External Reference for DAC—This bit controls the reference selection for the selected DAC. If this bit is set, the reference used for the DAC is the external reference voltage from the I/O connector. If this bit is cleared, the internal +10 $V_{ref}$ is used for the DAC reference. This bit is reserved on the PCI-MIO-16XE-50, PCI-MIO-16XE-10, |

| | | |
|---|---|---|
| | | PCI-6024E, PCI-6025E, and PCI-6031E. It should be set to 0. |
| 1 | ReGlitch | Reglitch DAC—When set, this bit configures the selected DAC to have a more uniform glitch when changing the output at the expense of a higher average glitch energy. This bit is reserved on the PCI-MIO-16E-4, PCI-MIO-16XE-50, PCI-MIO-16XE-10, PCI-6032E, and PCI-6031E. It should be set to 0. |
| 0 | BipDac | Bipolar DAC—This bit configures the voltage range of the selected DAC. If this bit is set, then the DAC is configured for bipolar operation of $-V_{ref}$ to $+V_{ref}$. In this mode, data written to the DAC is interpreted in two's complement format. If this bit is cleared, then the DAC is configured for unipolar operation of 0 V to $+V_{ref}$. In this mode, data written to the DAC is interpreted in straight binary format. This bit is reserved on the PCI-MIO-16XE-50 and should be set to 1. The PCI-MIO-16XE-50, PCI-6024E, and PCI-6025E DACs are always configured in bipolar mode. |

## DAC FIFO Data Register

The DAC FIFO Data Register is used to load the desired data into the DAC data FIFO.

The DAC FIFO is 2 kwords deep on the PCI-MIO-16E-1, PCI-MIO-16XE-10, PCI-6031E, and PCI-6071E. The DAC FIFO is 512 words deep on the PCI-MIO-16E-4. The DAC FIFO is 0 words deep on the PCI-MIO-16XE-50. The empty, half-full, and full flags from the 2 kword and 512 word DAC data FIFO are available in a status register in the DAQ-STC. These flags indicate when the FIFO is empty, half-full, or full, respectively. Only the full flag is available on the boards with the 0 word deep DAC data virtual FIFO. Whenever the FIFO is not full the host is free to write additional data.

**Address:** Base address + 1E (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

| Bit | Name | Description |
|---|---|---|
| 15–0 | D<15..0> | Data bits 15 through 0—The data to be written to the DAC data FIFO. This data is interpreted in straight binary form when DAC is configured for unipolar operation. In unipolar mode, the valid range is 0 to 65,536 for a 16-bit DAC and 0 to 4,096 for a 12-bit DAC. When the DAC is configured for bipolar operation, the data is interpreted in two's complement form. In bipolar mode, the valid range is –32,768 to 32,767 for a 16-bit DAC and –2,048 to 2,047 for a 12-bit DAC. |

## DAC0 Direct Data Register

The DAC0 Direct Data Register loads the desired data directly into DAC0, without using the DAC data FIFO.

**Address:** Base address + 18 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

| Bit | Name | Description |
|---|---|---|
| 15–0 | D<15..0> | Data bits 15 through 0—The data to be written directly to DAC0. This data is interpreted in straight binary form when DAC0 is configured for unipolar operation. In unipolar mode, the valid range is 0 to 65,536 for a 16-bit DAC and 0 to 4,096 for a 12-bit DAC. When DAC0 is configured for bipolar operation, the data is interpreted in two's complement form. In bipolar mode, the valid range is –32,768 to 32,767 for a 16-bit DAC and –2,048 to 2,047 for a 12-bit DAC. |

## DAC1 Direct Data Register

The DAC1 Direct Data Register loads the desired data directly into DAC1, without using the DAC data FIFO.

**Address:** Base address + 1A (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:**

| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
|---|---|---|---|---|---|---|---|
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 |

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |

| Bit | Name | Description |
|---|---|---|
| 15–0 | D<15..0> | Data bits 15 through 0—The data to be written directly to DAC1. This data is interpreted in straight binary form when DAC1 is configured for unipolar operation. In unipolar mode, the valid range is 0 to 65,536 for a 16-bit DAC and 0 to 4,096 for a 12-bit DAC. When DAC1 is configured for bipolar operation, the data is interpreted in two's complement form. In bipolar mode, the valid range is –32,768 to 32,767 for a 16-bit DAC and –2,048 to 2,047 for a 12-bit DAC. |

## DMA Control Register Group

The two registers making up the DMA Control Register Group configure the PCI E Series boards DMA interface. The AI AO Select and G0 G1 Select Registers select the DMA channels for the analog input, analog output, and general purpose counter timer resources.

The PCI E Series boards support four logical channels, which are routed through the three physical DMA channels of the MITE. Only one physical DMA channel is supported on the PCI-6023E, PCI-6024E, and PCI-6025E.

## AI AO Select Register

The AI AO Select Register contains 8 bits that control the logical DMA selection for the analog input and analog output resources. The contents of this register are cleared upon power up and after a reset condition.

**Address:** Base address + 09 (hex)

**Type:** Write-only

**Word Size:** 8-bit

**Bit Map:**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Output D/ Reserved | Output C/ Reserved | Output B/ Reserved | Output A/ Reserved | Input D | Input C | Input B | Input A |

| Bit | Name | Description |
|---|---|---|
| 7-4 | Reserved | Reserved—Always write 0 to these bits (*for PCI-6032E and PCI-6033E only*). |
| 7–4 | Output <D..A> | Analog Output Logical Channel D through A—These four bits select the logical channels of the MITE to be used by the analog output. You can only set one of these bits at a time (except for the PCI-6032E and PCI-6033E). |
| 3–0 | Input <D..A> | Analog Input Logical Channel D through A—These four bits select the logical channels to be used by the analog input. You can only set one of these bits at a time. |

## G0 G1 Select Register

The G0 G1 Select Register contains 8 bits that control the logical DMA selection for the two general purpose counter timer resources. The contents of this register are cleared upon power up and after a reset condition.

**Address:** Base address + 0B (hex)

**Type:** Write-only

**Word Size:** 8-bit

**Bit Map:**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| GPCT1 D | GPCT1 C | GPCT1 B | GPCT1 A | GPCT0 D | GPCT0 C | GPCT0 B | GPCT0 A |

| Bit | Name | Description |
|---|---|---|
| 7–4 | GPCT1 <D..A> | General Purpose Counter Timer 1 Logical Channel C through A—These four bits select the MITE logical channels that the GPCT1 uses. You can only set one of these bits at a time. |
| 3–0 | GPCT0 <D..A> | General Purpose Counter Timer 0 Logical Channel C through A—These four bits select the MITE logical channels that the GPCT1 uses. You can only set one of these bits at a time. |

## DAQ-STC Register Group

The registers making up the DAQ-STC Register Group configure and control the DAQ-STC system timing controller ASIC. These registers are described in the *DAQ-STC Technical Reference Manual*.

## FIFO Strobe Register Group

The three registers making up the FIFO Strobe Register Group are used to clear the three FIFOs on the PCI E Series.

## Configuration Memory Clear Register

Accessing the Configuration Memory Clear Register clears all information in the channel configuration memory and resets the write pointer to the first location in the memory.

**Window Address:** 52 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:** Not applicable; no bits used

## ADC FIFO Clear Register

Accessing the ADC FIFO Clear Register clears all information in the ADC data FIFO.

**Window Address:** 53 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:** Not applicable; no bits used

## DAC FIFO Clear Register

Accessing the DAC FIFO Clear Register clears all information in the DAC data FIFO.

☞ **Note** ***This Register is not valid for the PCI-6023E, PCI-6032E and PCI-6033E.***

**Window Address:** 54 (hex)

**Type:** Write-only

**Word Size:** 16-bit

**Bit Map:** Not applicable; no bits used

# 4

# Programming

This chapter contains programming instructions for operating the circuitry on the PCI E Series boards.

Programming the PCI E Series boards involves writing to and reading from registers on the board. Many of these registers belong to the DAQ-STC, and you will find a listing of these registers in the *DAQ-STC Technical Reference Manual*. For a list of all board registers not on the DAQ-STC, see Chapter 3, *Register Map and Descriptions*, of this manual.

The programming steps for analog input, output, digital I/O, and timing I/O are explained in the *DAQ-STC Technical Reference Manual*. These operations are explained in terms of bitfields. A *bitfield* is defined as a group of contiguous bits that jointly perform a function. Using bitfields has the advantage of establishing an efficient mapping technique to the underlying hardware. Also, the programming style becomes very modular, and a functional description of every programming step is easily understood.

Because the *DAQ-STC Technical Reference Manual* has detailed, step-by-step programming instructions, this register-level programmer manual gives a series of common programming examples with references to the *DAQ-STC Technical Reference Manual*. The program for each example is presented as it is in the *DAQ-STC Technical Reference Manual*; that is, in terms of functions, with bitfields to be written for each function. To implement the function, you need to perform a memory write to or a memory read from the specified registers. The complete examples are provided on the *PCI E Series Register-Level Programmer Manual* Companion Disk.

## PCI Local Bus

The PCI E Series boards are fully compatible with the *PCI Local Bus Specification*, Version 2.1 from the PCI Special Interest Group (SIG). The PCI Local Bus is a high performance, 32-bit bus with multiplexed address and data lines. The PCI system arbitrates and assigns resources through software, freeing you from manually setting switches and jumpers. You

must configure the bus-related resources before attempting to execute a register level program. Both the bus-related and data acquisition-related PCI configurations are described in the *PCI E Series User Manual*.

PCI local bus boards can be used on any PCI bus system, however, this manual only discusses IBM-compatible and Apple Macintosh systems. The *PCI E Series Register-Level Programmer Manual* Companion Disk contains different functions in C code for users to configure the PCI E Series boards.

The following sections describe the outline of PCI initialization on the PC and Macintosh. Detailed code and comments can be found on the Companion Disk.

## PCI Initialization for the IBM Compatible System

The PCI E Series boards use the MITE Application Specific Integrated Circuit (ASIC) chip as the PCI bus interface. National Instruments designed this ASIC specifically for data acquisition. In order for the board to operate properly this chip must be configured. Ordinarily, NI-DAQ performs this function, but if you are not using NI-DAQ, then you must configure the MITE ASIC chip. The following sections explain how to accomplish this.

The initialization is done by the `Setup_Mite()` function found on the companion diskette. `Setup_Mite()` performs the following functions:

1. It detects whether the PCI bus is present using the Is_PCI() function while utilizing PCI BIOS calls[1]. This verifies that the PCI bus is present.
2. It scans the PCI bus for all National Instruments PCI E Series boards using Find_NI_Devices() function. PCI BIOS calls are again used to find PCI boards that contain the National Instruments Vendor ID (0x1093) and a valid PCI E Series device ID (for example the device ID of the PCI-MIO-16XE-50 is 0x0162). If a board is found matching these requirements, its pertinent information is stored in a data structure.
3. It configures the device windows of the MITE and re-maps the board under 1 MB in the memory map. See the companion diskette and the *Re-mapping the PCI E Series Board* section in this chapter for further information

[1] The PCI SIG provides further information on the PCI BIOS calls.

## Re-mapping the PCI E Series Board

The PCI E Series board uses two base address registers (BAR). BAR points to the base address for the MITE registers, while BAR1 points to the base address of the board registers such as the DAQ-STC.

The size of each BAR is 4 KB and both are mapped into memory space. Therefore, you cannot use the C language I/O space read and write functions inp() and outp() to communicate with the PCI E Series board. Both BARs most likely map above 1 MB in the memory map. This means you must know how to perform memory cycles to extended memory, which is very difficult under operating systems such as Windows.

The Setup_Mite() function provides you with the capability to re-map the board under 1 MB making the communication much simpler. If you know how to perform memory calls under Windows or your operating system re-mapping the board is both unnecessary and undesirable.

The pseudo code shown below demonstrates how to re-map the board below 1 MB. If you do not want to re-map the board, you must still perform steps 4 and 5 to enable the device window of the MITE. All values and bit masks are 32 bits and all pseudo code functions are of the format function_name (address, data). This example assumes the new BAR0 address is 0xd000 and BAR1 address is 0xd1000. If you are using other addresses, you must make the appropriate changes.

1. Write the address where you want to re-map the MITE (BAR0) to PCI configuration space offset 0x10.

   Using 0xd0000 for the new BAR0 address

   PCI_Config_Write (0x10, 0xd0000)

2. Write the value 0xaeae to offset 0x340 from the new MITE address.

   Mem_Write (0xd0340, 0xaeae)

3. Write the address where you want to re-map the board registers (BAR1) to configuration space offset (0x14)

   Using 0xd1000 for the new BAR1 address

   PCI_Config_Write (0x14, 0xd1000)

4. Create the window_data_value by masking the board address.

   If you are not re-mapping the board,
   window_data_value = ((0xffffff00 AND BAR1) OR (0x80))

   If you are re-mapping the board,
   window_data_value = ((0xffffff00 AND 0xd1000) OR (0x80))
   window_data_value = 0xd1080 in this example.

5. Write the window data value to offset 0xc0 from the MITE (BAR0) address.

   If you are not re-mapping the board,
   Mem_Write (BAR0 + 0xc0, window_data_value)

   If you are re-mapping the board
   Mem_Write (0xd00c0, window_data_value)

The code provided on the companion diskette to re-map the board under the 1 MB region does not operate successfully on PCs with PCI to PCI bridge chips. These computers include the Dell Optiplex GXpro series. You must re-write these functions for proper operation on these systems.

## PCI Initialization for the Macintosh

The PCI E Series boards use the MITE Application Specific Integrated Circuit (ASIC) chip as the PCI bus interface. National Instruments designed this ASIC specifically for data acquisition. In order for the board to operate properly this chip must be configured. Ordinarily, NI-DAQ performs this function, but if you are not using NI-DAQ, then you must configure the MITE ASIC chip. The following sections explain how to accomplish this.

The initialization is done by the `Setup_Mite()` function. `Setup_Mite()` consists of two parts. First, the PCI bus is scanned for all National instruments PCI E Series boards. An NI-DAQ function[1] finds all PCI boards that contain the National Instruments vendor ID (0x1093) and PCI E Series board ID (for example, the board ID for the PCI-MIO-16XE-50 is 0x0162) in their configuration space. If a board is found, the program stores pertinent information of the board into a data structure.

Second, the MITE board windows must be configured. The NI-DAQ function stores the `Mite_Address` from Base Address Register 0 located at PCI configuration space 0x10 and the board address from Base Address Register 1 located at PCI configuration space 0x14. The window is enabled by performing a memory write to offset 0XC0 from the MITE address (BAR0). The DAQ-STC registers can then be read or written to from the board address (BAR1) + offset.

[1] Information on NI-DAQ function calls can be obtained from the *NI-DAQ Function Reference Manual*.

# Windowing Registers

Since the DAQ-STC contains a large number of registers (180), eight address lines would be required to decode the addresses in direct address mode. In addition to the DAQ-STC registers, the PCI E Series boards have other discrete registers. In order to retain compatibility with the AT E Series boards, which have a limited address space, the PCI E Series boards use the same windowing scheme for the DAQ-STC.

The windowing scheme allows a smaller address space requirement for the DAQ-STC at the expense of requiring more accesses to perform the same task. To write to a register in Windowed mode, write the address offset to the Window_Address_Register. Write the bit pattern (data) to the Window_Data_Write_Register. Similarly, to read from a register in Windowed mode, write the address offset to the Window_Address_Register; read from the Window_Data_Read_Register.

# Programming Examples

The programs presented in this chapter are broken into three sections: Digital I/O, Analog Input, and Analog Output. Each example provides the pseudo-code for the main program. The examples follow the procedure presented in detail in the *DAQ-STC Technical Reference Manual*, and further explanation of the functions are in this manual. Each program is also provided in its entirety on the *PCI E Series Register Level Programmer Manual* Companion Disk.

The Companion Disk also contains the support files necessary to run the examples. The support files for the PC are under the *PC* directory on the disk; the support files for the Macintosh are under the *Mac* directory, and all of the examples are under the *examples* directory. Include all of the files under the *PC* or *Mac* directory and the appropriate example file in your project. In addition, Macintosh users should include the NI-DAQ library for Macintosh as a shared library.

The support files for the PC include `lowio.h`, `pci.h`, `eserrlp.h`, `lowio.c`, `pci.c, pc_rw.c`, and `pc_ini.c. lowio.c` has the low level routines for memory read and write. `essrrlp.h` declares the external function prototypes and the register addresses; `pc_ini.c` includes the `Setup_Mite` function, which searches for the PCI board and reassigns the board address. `pc_rw.c` contains the functions which read and write from Windows and board discrete registers: `DAQ_STC_Windowed_Mode_Write`, `DAQ_STC_Windowed_Mode_Read`, `Board_Write`, `Board_Write_8bit`, and `Board_Read`.

The support files for the Macintosh include `eserrlp.h`, `mac_rw.c` and `mac_ini.c`. NI-DAQ Library has the routines for memory read and write. `essrrlp.h` declares the external function prototypes and the register addresses; `mac_ini.c` includes the `Setup_Mite` function which searches for the PCI-MIO board and saves the board address. `mac_rw.c` contains the functions which read and write from Windows and board discrete registers: `DAQ_STC_Windowed_Mode_Write`, `DAQ_STC_Windowed_Mode_Read`, `Board_Write`, `Board_Write_8bit` and `Board_Read`. In addition, Macintosh users should include the NI-DAQ library since the NI-DAQ library has the low-level routines for reading the configuration.

Before beginning register-level programming of the analog input and analog output sections, you should test the Windowed addressing scheme. To test the Windowed addressing scheme, use a simple example to operate on the DIO lines. When this example works, try to write code for the other sections.

As mentioned in Chapter 2 of the *DAQ-STC Technical Reference Manual*, several write-only registers on the DAQ-STC contain bitfields that control a number of functionally independent parts of the chip. To update bitfields, you must set or clear bits without changing the status of the remaining bits in the register. Since you cannot read these registers to determine which bits are set, you should maintain a software copy of the write-only registers.

☞ **Note** ***For simplicity these examples do not include software copies of the registers. If you wish to write your own examples, or modify these examples, we strongly recommend adding software copies of the write-only registers. Please refer to Chapter 2,*** *Register and Bitfield Programming Considerations****, in the*** *DAQ-STC Technical Reference Manual* ***for further information.***

# Digital I/O

Chapter 7 of the *DAQ-STC Technical Reference Manual* describes the DIO module of the DAQ-STC and illustrates an example (C language) in Windowed mode to toggle the DIO lines. Example 1 verifies that the Windowed addressing scheme works. Example 2 illustrates digital I/O.

## Example 1

This example illustrates the use of Windowed registers by toggling the digital lines.

First configure all the digital lines as outputs. Write 0x0 through 0xFF to the DIO output register and make sure the digital lines toggle. (This example is also presented in Chapter 7 of the *DAQ-STC Technical Reference Manual*.)

1. Set up the PCI board resources. Use the function `Setup_Mite` provided on the Companion Disk.
2. Configure all the digital lines as outputs.

   ```
   DIO_Control_Register = 0xFF;
   ```

3. Output the digital patterns.

   ```
   for (i=0;i<=255;i++)
   {
       DIO_Output_Register = i;
   }
   ```

## Example 2

This example shows how to perform digital I/O.

Configure digital lines 0, 2, 4, and 6 as outputs and the remaining lines as inputs. Wrap back the output lines to the input lines. Write patterns 0b0000 and 0b1111 to the output lines. Read back the input pins to verify the data. Also check some intermediate patterns.

1. Set up the PCI board resources. Use the function `Setup_Mite` provided on the Companion Disk.
2. Configure lines 0, 2, 4, and 6 as outputs and 1, 3, 5, and 7 as inputs.

   ```
   DIO_Control_Register = 0x55;
   ```

3. Write the digital pattern.

   ```
   DIO_Output_Register = 0x00;
   ```

4. Read the digital pattern.

   Pattern = DIO_Parallel_Input Register;

5. Repeat Steps 3 and 4 for subsequent patterns.

## Analog Input

See Chapter 2 in the *DAQ-STC Technical Reference Manual* for information on programming analog input and the relevant registers. This section also describes the programming sequences for the various functions and organizes these functions for a particular operation. Individual bitfield descriptions are also provided.

Programming the PCI E Series boards for analog input can be divided into writing to and reading from two main register groups: discrete board registers and DAQ-STC registers. The following functions configure the board by calling the `Board_Read` and `Board_Write` functions:

```
Setup_Mite;
Configure_Board;
```

Analog input DAQ-STC programming consists of the following functions which call the `DAQ_STC_Windowed_Mode_Read` and `DAQ_STC_Windowed_Mode_Write` functions:

```
AI_Initialize_Configuration_Memory_Ouput,
MSC_Clock_Configure,
Clear_FIFO,
AI_Reset_All,
AI_Board_Personalize,
FIFO_Request,
AI_Hardware_Gating,
AI_Trigger_Signals,
Number_of_Scans,
AI_Scan_Start,
AI_End_of_Scan,
Convert_Signal,
AI_Interrupt_Enable,
AI_Arming,
AI_Start_The_Acquisition,
```

As the analog input examples increase in complexity, more of these functions will be necessary. The functions will be presented in the applicable examples; subsequent examples address only the specific differences from Example 1. This manual provides the structure and

pseudo-code for each example. The *PCI E Series Register Level Programmer Manual* Companion Disk contains the complete programs. The following pseudo-code examples and the programs on the Companion Disk follow the flowchart structure presented in the *DAQ-STC Technical Reference Manual*.

## Example 1

This example acquires one sample from channel 0.

Connect a voltage source to ACH0 on the I/O connector. Configure channel 0 for bipolar RSE with no dithering. No external multiplexers are present. Sample analog input channel 0 at a gain of 1 and read the unscaled result. Compare the unscaled value to the input voltage.

The first two steps set up the E Series board, and the subsequent steps configure the DAQ-STC.

1. Set up the PCI board resources. Use the function `Setup_Mite` provided on the Companion Disk.
2. Configure the analog channel for the given settings.

   The function `Configure_Board` clears the configuration memory, clears the ADC FIFO, and then sets channel 0 to the given settings. (Clearing the configuration memory and ADC FIFO require windowed mode writes as well as board writes.)

   Write_Strobe_0_Register
   Write strobe 0 = 1;

   Write_Strobe_1_Register
   Write strobe 1 = 1;

   Configuraton_Memory_High_Register
   Channel number = 0;
   Channel type = 3;

   Configuration_Memory_Low_Register
   Last channel = 1;
   Gain = 1;
   Polarity = 0;
   Dither enable = 0;
3. The programming of the DAQ-STC begins with the clock configuration.

   The function `MSC_Clock_Configure` selects the timebase for the DAQ-STC.

Clock_and_FOUT_Register
Slow internal timebase = 1;
Divide timebase by two = 1;
Clock to board = 1;
Board divide by two =1;

4. Call the function `Clear_FIFO` to clear the ADC FIFO.

   Write_Strobe_1_Register
   Write strobe 1 = 1;

5. The function `AI_Reset_All` stops any other activities in progress and start the configuration process.

   Joint_Reset_Register
   AI reset = 1;
   AI configuration start = 1;

   Interrupt_A_Ack_Register = 0x3F80;

   AI_Mode_1_Register
   Reserved one = 1;
   AI start stop = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

6. `AI_Board_Personalize` sets the DAQ-STC for the PCI E Series board.

   Joint_Reset_Register
   AI configuration start = 1;

   Clock_and_FOUT_Register
   Output divide by two = 1;

   Set the AI_Personal_Register = 0xA4A0;

   Set the AI_Output_Control Register = 0x032E;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

7. Call the function `AI_Initialize_Configuration_Memory_Output` to output one pulse and access the first value in the configuration FIFO.

   AI_Command_1_Register
   Convert pulse = 1;

8. The function `AI_Board_Environmentalize` configures the board for any external multiplexers.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Mode_2_Register
   External mux present = 0;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

9. Call `AI_Trigger_Signals` to set the triggering options.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Mode_1_Register
   Trigger once = 1;

   AI_Trigger_Select_Register
   Start edge = 1;
   Start sync = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

10. The function `AI_Scan_Start` selects the scan start event.

    Joint_Reset_Register
    AI configuration start = 1;

    AI_Start_Stop_Select_Register
    Start edge = 1;
    Start sync = 1;

    Joint_Reset_Register
    AI configuration start = 0;
    AI configuration end = 1;

11. Call the function `AI_End_of_Scan` to select the end of scan event.

    Joint_Reset_Register
    AI configuration start = 1;

    AI_Start_Stop_Select_Register
    Stop select = 19;
    Stop sync = 1;

    Joint_Reset_Register
    AI configuration start = 0;
    AI configuration end = 1;

12. Call the function `Clear_FIFO` to clear the ADC FIFO.

    Write_Strobe_1_Register
        Write strobe 1 = 1;

13. Now start the acquisition with `AI_Start_The_Acquisition.`

    AI_Command_1_Register
        Convert Pulse = 1;

14. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

    Do
    {
        If (AI FIFO not empty) then
            read FIFO data;
    } while (FIFO not read)

## Example 2

Example 2 illustrates the manner in which to program the STC for scanning.

Acquire 5 scans at a scan interval of 1 ms. The scan list contains channels 5, 4, 1, and 0, respectively. The channels are configured as RSE at a gain of 1. Within each scan, the sample interval should be 100 µs. Dithering should remain off during the operation. No external multiplexers are used. Use polled input to acquire the data.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for each channel in the scan list. Only channel 0 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3-9.
4. Call the function `Number_of_Scans` to load the number of scans.

    Joint_Reset_Register
        AI configuration start = 1;

    AI_SC_Load_A_Registers (24 bits)
        Number of posttrigger scans -1 = 4;

    AI_Command_1_Register
        AI SC Load = 1;

    Joint_Reset_Register
        AI configuration start = 0;
        AI configuration end = 1;

5. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Start_Stop_Select_Register
   Start edge = 1;
   Start sync = 1;

   AI_SI_Load_A_Registers (24 bits)
   AI SI special ticks -1 = 1;

   AI_Command_1_Register
   AI SI load = 1;

   AI_SI_Load_A_Registers (24 bits)
   AI SI ordinary ticks -1 = 19999;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

6. Perform Analog Input Example 1 Step 11.
7. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SI2_Load_A_Register
   AI SI2 special ticks -1 = 1999;

   AI_SI2_Load_B_Register
   AI SI2 ordinary ticks -1 = 1999;

   AI_Mode_2_Register
   AI SI2 reload mode = 1;

   AI_Command_1_Register
   AI SI2 load = 1;

   AI_Mode_2_Register
   AI SI2 initial load source = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

8. Perform Analog Input Example 1 Step 4.

9. Call `AI_Arming` to arm the analog input counter.

   AI_Command_1_Register
   AI SC arm = 1;
   AI SI arm = 1;
   AI SI2 arm = 1;
   AI DIV arm = 1;

10. The function `AI_Start_The_Acquisition` starts the acquisition process.

    AI_Command_2_Register
    AI START1 Pulse = 1;

11. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

    Do
    {
    If (AI FIFO not empty) then
    read FIFO data;
    } while (20 samples have not been read)

## Example 3

Example 3 performs the same acquisition as Example 2, but with interrupts.

Acquire 5 scans at a scan interval of 1 ms. The scan list contains channels 5, 4, 1, and 0 respectively. The channels are configured as RSE at a gain of 1. Within each scan, the sample interval should be 100 µs. Dithering should remain off during the operation. No external multiplexers are used. Use interrupts to acquire the data.

Only minor modifications to Analog Input Example 2 are needed for this example. Instead of installing the interrupt service routine (ISR) as an interrupt, this example emulates the operation of the interrupt by polling the status register in the DAQ-STC. When the status register indicates an interrupt, the main loop transfers control to the ISR. To use the example ISR as an actual interrupt, learn how to install software interrupts on your system. Generally, the procedure is as follows:

1. Determine the software interrupt number corresponding to the IRQ line you are using.
2. Use the OS specific functions such as `getvect()` and `setvect()` for DOS to replace the default interrupt handler with your ISR. You should disable interrupts during this step.
3. Reset the interrupt controller hardware.

4. Perform your analog input.
5. After the analog input operation completes, re-install the default interrupt handler.

## Example Program

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for each channel in the scan list. Only channel 0 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3-9.
4. Call the function `Number_of_Scans` to load the number of scans.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SC_Load_A_Registers (24 bits)
   Number of posttrigger scans -1 = 4;

   AI_Command_1_Register
   AI SC Load = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
5. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Start_Stop_Select_Register
   Start edge = 1;
   Start sync = 1;

   AI_SI_Load_A_Registers (24 bits)
   AI SI special ticks -1 = 1;

   AI_Command_1_Register
   AI SI load = 1;

   AI_SI_Load_A_Registers (24 bits)
   AI SI ordinary ticks -1 = 19999;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
6. Perform Analog Input Example 1 Step 11.

7. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SI2_Load_A_Register
   AI SI2 special ticks -1 = 1999;

   AI_SI2_Load_B_Register
   AI SI2 ordinary ticks -1 = 1999;

   AI_Mode_2_Register
   AI SI2 reload mode = 1;

   AI_Command_1_Register
   AI SI2 load = 1;

   AI_Mode_2_Register
   AI SI2 initial load source = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

8. Perform Analog Input Example 1 Step 4.

9. The function `AI_Interrupt_Enable` enables interrupts for the acquisition.

   Interrupt_A_Enable_Register
   AI FIFO interrupt enable = 1;
   AI error interrupt enable = 1;

   Interrupt_Control_Register
   MSC IRQ pin = 0;
   MSC IRQ enable = 1;

10. Call `AI_Arming` to arm the analog input counter.

    AI_Command_1_Register
    AI SC arm = 1;
    AI SI arm = 1;
    AI SI2 arm = 1;
    AI DIV arm = 1;

11. Install the interrupt service routine to handle the interrupt.

    Interrupt_Service_Routine()
    read FIFO data;
    increment sample counter;

12. The function `AI_Start_The_Acquisition` starts the acquisition process.

    AI_Command_2_Register
        AI START1 Pulse = 1;

13. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and call the ISR.

    Do
    {
        If (AI FIFO not empty) then
            call `Interrupt_Service_Routine`;
    } while (20 samples have not been read)

## Example 4

Example 4 performs the same acquisition as Example 2, but with DMA.

This example scans four channels 10 times using DMA to transfer data from the AI FIFO. Acquire 10 scans with a scan interval of 1 ms. The scan list contains channels 5, 4, 1, and 0, respectively, each at a gain of 1 and in RSE mode. Within each scan, the sample interval should be 100 µs. Dithering should remain off during the acquisition. Use DMA to transfer data from the FIFO. The DMA controller puts the data into two different buffers, each has a size of 40 bytes. This example constructs a linked list to contain the information about the buffers, such as their physical address, logical address and total transfer bytes. You need to pass the linked list head node to `MITE_DMAProgram` to program the MITE.

Since the physical address calculation is only valid for real mode on a PC, this example only runs under DOS. If you want to program DMA under other operating systems, you must learn about the memory's physical address calculation and the usage of virtual memory. For example, if you want to program DMA under Windows or Macintosh OS, make sure that you lock the memory locations for DMA, so the virtual memory manager cannot move the data from current physical memory location. This example provides a basic outline for you to program the MITE for DMA transfers.

`dma.c` contains the functions `MITE_DMAProgram`, `MITE_DMAarm`, and `MITE_DMAdisarm`. `miteregb.h` contains the bit field assignment for the MITE.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for each channel in the scan list. Only channel 0 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3-9.

4. Call the function `Number_of_Scans` to load the number of scans.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SC_Load_A_Registers (24 bits)
   Number of posttrigger scans -1 = 4;

   AI_Command_1_Register
   AI SC Load = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

5. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register

   AI configuration start = 1;

   AI_Start_Stop_Select_Register
   Start edge = 1;
   Start sync = 1;

   AI_SI_Load_A_Registers (24 bits)
   AI SI special ticks -1 = 1;

   AI_Command_1_Register

   AI SI load = 1;

   AI_SI_Load_A_Registers (24 bits)

   AI SI ordinary ticks -1 = 19999;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

6. Perform Analog Input Example 1 Step 11.

7. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SI2_Load_A_Register
   AI SI2 special ticks -1 = 1999;

   AI_SI2_Load_B_Register
   AI SI2 ordinary ticks -1 = 1999;

   AI_Mode_2_Register
   AI SI2 reload mode = 1;

   AI_Command_1_Register
   AI SI2 load = 1;

AI_Mode_2_Register
AI SI2 initial load source = 1;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

8. Perform Analog Input Example 1 Step 4.
9. AI_Arming to arm the analog input counter

   AI_Command_1_Register
   AI SC arm = 1;
   AI SI arm = 1;
   AI SI2 arm = 1;
   AI DIV arm = 1;

10. Construct the buffer information linked list. Each node contains the information about one buffer: Every node stores each buffer's physical address and logical address, and total transfer bytes for that buffer.
11. Set up the DRQ channel

    AI_AO_Select_Register
    DMA Channel A enable=1;

12. Call the function `MITE_DMAProgram` to set up the MITE for DMA transfer

    You need to pass the following parameters to the function:

| | |
|---|---|
| *BufferinfoNodeHeadPtr:* | Point to the buffer information linked list head node. |
| *NumberOfBuffer:* | Number of buffers you use for DMA transfers |
| *dmaChannel:* | DMA channel you want to use, there are four DMA channels on each PCI board (0 to 3). |
| *direction:* | Your DMA transfer direction (INPUT or OUTPUT) |
| *continuous:* | Perform continuous DMA transfer (FALSE or TRUE) |
| *drqnum:* | DRQ channel you want to use. Make sure it is the same in Step 11 (0 to 3). |

13. The function `AI_Start_The_Acquisition` starts the acquisition process.

    AI_Command_2_Register
        AI START1 Pulse = 1;

14. Call the function `MITE_DMAarm` to start the DMA transfer.
15. Loop to check the status about how many bytes still remain in process. This loop stops when the number of transfers remaining exceeds the number of buffers. When DMA finishes transferring data to each buffer, transfers remaining becomes 0. Thus, there are at least *n* transfers remaining for *n* buffer transfers.

    do {
        Call the `MITE_DMAgettransfersRemaining` to check the number of bytes remaining
        if (transfersRemaining==0) increase transfersendcount;
        } while (transfersendount<=NumberOfBuffer);

16. Print out the data in the buffers
17. Call the function `MITE_DMAdisarm` to disarm the MITE for DMA.
18. Release the memory storage used by the buffer information linked list.

## Programming the MITE for Different DMA Transfers

To use DMA transfer for Analog Output, store the output data in the buffers and construct the output buffer information linked list. Then, perform all the steps above but pass OUTPUT for the *direction* parameter to the function `MITE_DMAProgram`.

To perform two or more DMA transfers at the same time, use different DMA and DRQ channels for different DMA transfers. For example, you want to have DMA transfers for analog input and analog output at the same time. Use DRQ channel 0 and DMA channel 0 for analog input, and DRQ channel 1 and DMA channel 1 for analog output. In other words, use only one DMA channel and one DRQ channel to perform either analog input or analog output. When using two or more channels for DMA transfers, call the functions `MITE_DMAProgram`, `MITE_DMAarm` two or more times. To operate analog input and analog output at the same time, call the function `MITE_DMAProgram` and pass the input buffer information linked list head node pointer, number of buffers, 0 for *drqnum*, INPUT for *direction*, FALSE for continuous and 0 for *dmachannel*. Call the function `MITE_DMAProgram` again and pass the output buffer information linked list head node pointer, number of buffer, 1 for *drqnum*, OUTPUT for *direction*, FALSE for continuous and 1 for *dmachannel*. Then call `MITE_DMAarm` twice for DMA channel 0 and channel 1. After the DMA

transfers finish, call `MITE_DMAdisarm` twice for DMA channel 0 and channel 1.

## Example 5

Example 5 performs the same acquisition as Example 2, but with the start trigger and scan start pulses applied externally.

Acquire 5 scans. The scan list contains channels 5, 4, 1, and 0, respectively, each at a gain of 1 and in RSE mode. The acquisition should begin on a start trigger applied to PFI0. The sample rate should be set to 100 µs, and the start pulses should be connected to PFI1 to trigger each scan. Use polled input to read the AI FIFO data.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for each channel in the scan list. Only channel 0 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3 through 8.
4. Call `AI_Trigger_Signals` to set the triggering options.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Mode_1_Register
   AI trigger once = 1;

   AI_Trigger_Select_Register = 0x8061;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
5. Call the function `Number_of_Scans` to load the number of scans.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SC_Load_A_Registers (24 bits)
   Number of posttrigger scans -1 = 4;

   AI_Command_1_Register
   AI SC Load = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
6. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register
   AI configuration start = 1;

AI_START_STOP_Select_Register = 0x0062;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

7. Perform Analog Input Example 1 Step 11.
8. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Mode_3_Register
   AI SI2 source = 1;

   AI_SI2_Load_A_Register
   AI SI2 special ticks -1 = 1999;

   AI_SI2_Load_B_Register
   AI SI2 ordinary ticks -1 = 1999;

   AI_Mode_2_Register
   AI SI2 reload mode = 1;

   AI_Command_1_Register
   AI SI2 load = 1;

   AI_Mode_2_Register
   AI SI2 initial load source = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
9. Perform Analog Input Example 1 Step 4.
10. Call `AI_Arming` to arm the analog input counter.

    AI_Command_1_Register
    AI SC arm = 1;
    AI SI arm = 0;
    AI SI2 arm = 1;
    AI DIV arm = 1;
11. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

    Do
    {
    If (AI FIFO not empty) then
    read FIFO data;
    } while (20 samples have not been read)

## Example 6

Example 6 performs 20 scans as in Example 2. Additionally, an external start and stop trigger control the acquisition.

Acquire 20 scans. The scan list contains channels 5, 4, 1, and 0, respectively, each at a gain of 1 and in RSE mode. The acquisition should begin on a start trigger applied to PFI0. Set the sample rate to 100 µs. Connect the stop trigger to PFI1. Acquire 10 scans after the stop trigger, leaving 10 scans before the stop trigger. Use polled input to read the AI FIFO data.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for each channel in the scan list. Only channel 0 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3 through 8.
4. Call `AI_Trigger_Signals` to set the triggering options.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Mode_1_Register
   AI trigger once = 1;

   AI_Trigger_Select_Register = 0xB161;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
5. Call the function `Number_of_Scans` to load the number of scans.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_Mode_2_Register
   AI pretrigger = 1;

   AI_SC_Load_B_Registers (24 bits)
   Number of pretrigger scans -1 = 9;

   AI_Mode_2_Register
   AI SC initial load source = 1;
   AI SC reload mode = 1;

   AI_SC_Load_A_Registers (24 bits)
   Number of posttrigger scans -1 = 9;

   AI_Command_1_Register
   AI SC load = 1;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

6. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_START_STOP_Select_Register = 0x0060;

   AI_SI_Load_A_Registers (24 bits)
   AI SI special ticks -1 = 1;

   AI_Command_1_Register
   AI SI load = 1;

   AI_SI_Load_A_Registers (24 bits)
   AI SI ordinary ticks -1 = 19999;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

7. Perform Analog Input Example 1 Step 11.

8. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SI2_Load_A_Register
   AI SI2 special ticks -1 = 1999;

   AI_SI2_Load_B_Register
   AI SI2 ordinary ticks -1 = 1999;

   AI_Mode_2_Register
   AI SI2 reload mode = 1;

   AI_Command_1_Register
   AI SI2 load = 1;

   AI_Mode_2_Register
   AI SI2 initial load source = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

9. Perform Analog Input Example 1 Step 4.

10. Call `AI_Arming` to arm the analog input counter.

    AI_Command_1_Register
        AI SC arm = 1;
        AI SI arm = 1;
        AI SI2 arm = 1;
        AI DIV arm = 1;

11. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

    Do
    {
        If (AI FIFO not empty) then
            read FIFO data;
            if (count = 80)
                reset count to 0;
    } while (SC_TC flag in the AI_Status_1_Register is not set or the AIFIFO not empty)

## Example 7

Example 7 performs the same scanning as Example 2, but as a single wire acquisition.

Acquire 5 scans. The scan list contains channels 5, 4, 1, and 0, respectively, each at a gain of 1 and in RSE mode. The START and CONVERT signals should be applied to PFI0. Use polled input to read the AI FIFO data.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for each channel in the scan list. Only channel 0 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3 through 9.
4. Call the function `Number_of_Scans` to load the number of scans.

    Joint_Reset_Register
        AI configuration start = 1;

    AI_SC_Load_A_Registers (24 bits)
        Number of posttrigger scans -1 = 4;

    AI_Command_1_Register
        AI SC Load = 1;

    Joint_Reset_Register
        AI configuration start = 0;
        AI configuration end = 1;

5. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_START_STOP_Select_Register = 0x0021;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

6. Perform Analog Input Example 1 Step 11.
7. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register

   AI configuration start = 1;

   AI_Mode_2_Register
   AI SC gate enable = 1;
   AI START STOP gate enable = 1;

   AI_Mode_1_Register
   AI CONVERT source select = 1;
   AI CONVERT source polarity = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

8. Perform Analog Input Example 1 Step 4.
9. Call `AI_Arming` to arm the analog input counter.

   AI_Command_1_Register
   AI SC arm = 1;
   AI SI arm = 0;
   AI SI2 arm = 0;
   AI DIV arm = 1;

10. The function `AI_Start_The_Acquisition` starts the acquisition process.

    AI_Command_2_Register
    AI START1 pulse = 1;

11. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

    Do
    {
    If (AI FIFO not empty) then
    read FIFO data;
    } while (20 samples have not been read)

## Example 8

This example samples one channel on an AMUX-64T.

Use the factory default settings on the AMUX-64T, internal power, single-board configuration, no temperature setting, and the shield unconnected. Sample channel 3 on the AMUX-64T to acquire 100 samples at a sample interval of 10 µs. Connect a voltage source to channel 3, and compare the unscaled results to the applied voltage. Read the samples using polled input.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for channel 3.
3. Perform Analog Input Example 1 Steps 3 through 6.
4. Call the function `AI_Initialize_Configuration_Memory_Output` to output one pulse and access the first value in the configuration FIFO. This function also configures the DIO circuitry for the AMUX-64T.

   AI_Command_1_Register
   AI convert pulse = 1;

   DIO_Control_Register = 0x0003;
   DIO_Output_Register = 0x0003;
   DIO_Control_Register = 0x0803;
   DIO_Control_Register = 0x0003;
5. Perform Analog Input Example 1 Steps 8 through 9.
6. Call the function `Number_of_Scans` to load the number of scans.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SC_Load_A_Registers (24 bits)
   Number of posttrigger scans -1 = 99;

   AI_Command_1_Register
   AI SC Load = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;
7. The function `AI_Scan_Start` selects the scan start event.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_START_STOP_Select_Register = 0x0060;

AI_SI_Load_A_Registers (24 bits)
AI SI special ticks -1 = 1

AI_Command_1_Register
AI SI load = 1;

AI_SI_Load_A_Registers (24 bits)
AI AI ordinary ticks -1 = 199;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

8. Perform Analog Input Example 1 Step 11.
9. `Convert_Signal` selects the convert signal for the acquisition.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SI2_Load_A_Register
   AI SI2 special ticks -1 = 1;

   AI_SI2_Load_B_Register
   AI SI2 ordinary ticks -1 = 1;

   AI_Mode_2_Register
   AI SI2 reload mode = 1;

   AI_Command_1_Register
   AI SI2 load = 1;

   AI_Mode_2_Register
   AI SI2 initial load source = 1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end = 1;

10. Perform Analog Input Example 1 Step 4.
11. Call `AI_Arming` to arm the analog input counter.

    AI_Command_1_Register
    AI SC arm = 1;
    AI SI arm = 1;
    AI SI2 arm = 1;
    AI DIV arm = 1;

12. Now start the acquisition with `AI_Start_The_Acquisition.`

    AI_Command_2_Register
    AI START1 pulse = 1;

13. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

```
Do
{
    If (AI FIFO not empty) then
        read FIFO data;
} while (100 samples have not been read)
```

## Example 9

This example scans 8 channels on an AMUX-64T.

Use the default settings on the AMUX-64T, internal power, single-board configuration, no temperature setting, and the shield unconnected. Scan channels 0 through 7 on the AMUX-64T. Acquire 10 scans at a scan interval of 200 µs and a sample interval of 20 µs. Connect a voltage source to channels 0 through 3 and ground channels 4 through 7. Compare the unscaled results to the applied voltage. Read the samples using polled input.

1. Perform Analog Input Example 1 Step 1.
2. Perform Analog Input Example 1 Step 2 for channels 0 and 1. Only channel 1 has Last channel set to 1.
3. Perform Analog Input Example 1 Steps 3 through 6.
4. Call the function `AI_Initialize_Configuration_Memory_Output` to output one pulse and access the first value in the configuration FIFO. This function also configures the DIO circuitry for the AMUX-64T.

```
AI_Command_1_Register
    AI convert one pulse = 1;

AI_Mode_2_Register
    AI external mux present = 1;

DIO_Control_Register = 0x0800;
DIO_Control_Register = 0x0000;
DIO_Output_Register = 0x0000;
DIO_Control_Register = 0x0003;
DIO_Control_Register = 0x0803;
DIO_Control_Register = 0x0003;
```

5. The function `AI_Board_Environmentalize` configures the board for any external multiplexers.

```
Joint_Reset_Register
    AI configuration start = 1;
```

AI_Mode_2_Register
External mux present = 1;

AI_Output_Control_Register
AI external mux clock select = 3;

AI_DIV_Load_A_Register
AI number of channels ratio -1 = 3;

AI_Command_1_Register
AI DIV load = 1;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

6. Perform Analog Input Example 1 Step 9.
7. Call the function `Number_of_Scans` to load the number of scans.

Joint_Reset_Register
AI configuration start = 1;

AI_SC_Load_A_Registers (24 bits)
Number of posttrigger scans - 1 = 99;

AI_Command_1_Register
AI SC Load = 1;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

8. The function `AI_Scan_Start` selects the scan start event.

Joint_Reset_Register
AI configuration start = 1;

AI_START_STOP_Select_Register = 0x0060;

AI_SI_Load_A_Registers (24 bits)
AI SI special ticks - 1 = 1

AI_Command_1_Register
AI SI load = 1;

AI_SI_Load_A_Registers (24 bits)
AI AI ordinary ticks - 1 = 3999;

Joint_Reset_Register
AI configuration start = 0;
AI configuration end = 1;

9. Perform Analog Input Example 1 Step 11.

10. `Convert_Signal` selects the convert signal for the acquisition.

    Joint_Reset_Register
        AI configuration start = 1;

    AI_SI2_Load_A_Register
        AI SI2 special ticks - 1 = 399;

    AI_SI2_Load_B_Register
        AI SI2 ordinary ticks -1 = 399;

    AI_Mode_2_Register
        AI SI2 reload mode = 1;

    AI_Command_1_Register
        AI SI2 load = 1;

    AI_Mode_2_Register
        AI SI2 initial load source = 1;

    Joint_Reset_Register
        AI configuration start = 0;
        AI configuration end = 1;

11. Perform Analog Input Example 1 Step 4.
12. Call `AI_Arming` to arm the analog input counter.

    AI_Command_1_Register
        AI SC arm = 1;
        AI SI arm = 1;
        AI SI2 arm = 1;
        AI DIV arm = 1;

13. Now start the acquisition with `AI_Start_The_Acquisition.`

    AI_Command_2_Register
        AI START1 pulse = 1;

14. Poll the AI FIFO not empty flag in the AI_Status_1_Register until not empty and read the ADC FIFO data in the ADC_FIFO_Data_Register.

    Do
    {
        If (AI FIFO not empty) then
            read FIFO data;
    } while (80 samples have not been read)

# Analog Output

Chapter 3 of the *DAQ-STC Technical Reference Manual* contains all the information on the analog output timing/control module of the DAQ-STC,

with specific programming steps in the *Programming Information* section. The instructions are arranged in a sequence of functions, which are used in the examples below to program the DAQ-STC. Because the PCI-6032E and PCI-6033E do not have analog output circuitry, they cannot run analog output examples.

Analog output DAQ-STC programming consists of the following functions:

```
AO_Reset_All;
MSC_Clock_Configure;
AO_Board_Personalize;
AO_Triggering;
AO_Counting;
AO_Updating;
AO_Channels;
AO_LDAC_Source_And_Update_Mode;
AO_Errors_To_Stop_On;
AO_FIFO;
AO_Arming;
AO_Start_The_Acquisition;
```

As the analog output examples increase in complexity, more of these functions will be necessary. The functions will be presented in the applicable examples; subsequent examples address only the specific differences from Example 1. This manual provides the structure and pseudo-code for each example. The *PCI E Series Register Level Programmer Manual* Companion Disk contains the complete programs. The following pseudo-code examples and the programs on the Companion Disk follow the flowchart structure presented in the *DAQ-STC Technical Reference Manual*.

## Example 1

This is an example of CPU write to the DAC.

Write 5 V to DAC1 in bipolar mode. Use internal reference and disable reglitching. The data FIFO is not used. Use the immediate update mode to update the output on DAC1.

1. Set up the PCI board resources. Use the function `Setup_Mite` provided on the Companion Disk.
2. Configure the board by setting the following board level bits.

AO_Configuration_Register
DACSel<3..0> = 1;
BipDac = 1;
ExtRef = 0;
ReGlitch = 0;
GroundRef = 0;

3. Call `AO_Reset_All` to reset the DAQ-STC.

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Command_1_Register
   AO disarm = 1;

   Interrupt_B_Enable_Register = 0x0000;

   AO_Personal_Register
   AO BC source select = 1;

   Interrupt_B_Ack_Register = 0x3F98;

   Joint_Reset_Register
   AO configuration start = 0;
   AO configuration end = 1;

4. Call `AO_Board_Personalize` to configure the DAQ-STC.

   Joint_Reset_Register
   AO_Configuration_Start = 1;

   If (Board Type = PCI-MIO-16E-1, PCI-6071E or PCI-MIO-16E-4)
   AO_Personal_Register = 0x1410;

   Else

   AO_Personal_Register = 0x1430;

   Clock_and_FOUT_Register = 0x1B20;

   AO_Output_Control_Register = 0x0000;

   AO_START_Select_Register = 0x 0000;

   Joint_Reset_Register
   AO configuration start = 0;
   AO configuration end = 1;

5. Call `AO_LDAC_Source_And_Update_Mode` to set the update mode to immediate update mode.

   Joint_Reset_Register
   AO configuration start = 1;

AO_Command_1_Register
AO LDAC0 source select = 0;
AO DAC0 update mode = 0;
AO LDAC1 source select = 0;
AO DAC1 update mode = 0;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

6. Write to the DAC1 Data Register. In bipolar mode, +10 V corresponds to 2,047 (0x07FF) and –10 V corresponds to –2,048 (0xF800). Hence, +5 V corresponds to 1,024 (0x0400). Writing this to the DAC1 data register results in +5 V appearing at the DAC1OUT line.

## Example 2

This example generates a waveform using polled writes to the data FIFO.

Initialize the buffer with 3000 points. Use polled writes to write each point to the data FIFO. Updates occur every 2 ms. Output the buffer five times. Confirm operation with an oscilloscope.

1. Perform Analog Output Example 1 Step 1.
2. Load an array with the voltages of the waveform to be output. In bipolar mode, +10 V corresponds to 2,047 (0x07FF) and –10 V corresponds to –2,048 (0xF800). To convert the voltage you want to output to the binary value, use the following formula:

   binary value = (voltage/10) * 2048
3. Configure the board by setting the following board level bits.

   AO_Configuration_Register:
   DACSel<3..0> = 1;
   BipDac = 1;
   ExtRef = 0;
   ReGlitch = 0;
   GroundRef = 0;
4. Reset the data FIFO.
5. Pre-load the data FIFO with the voltages from the array.

   while (FIFO is not full and there is more data to write)
   {
   AO_DAC_FIFO_Data = data;
   }

6. Call `MSC_Clock_Configure` to program the timebase options.

   Clock_and_FOUT_Register = 0x1B00;

7. Call `AO_Reset_All` to program the timebase options.

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Command_1_Register
   AO disarm = 1;

   Interrupt_B_Enable_Register = 0x0000;

   AO_Personal_Register
   AO BC source select = 1;

   Interrupt_B_Ack_Register = 0x3F98;

   Joint_Reset_Register
   AO configuration start = 0;
   AO configuration end = 1;

8. Call `AO_Board_Personalize` to configure the DAQ-STC for the MIO board. Use the following bitfield settings:

   Joint_Reset_Register
   AO_Configuration_Start = 1;

   If (Board Type = PCI-MIO-16E-1, PCI-6071E or PCI-MIO-16E-4)
   AO_Personal_Register = 0x1410;

   Else

   AO_Personal_Register = 0x1430;

   Clock_and_FOUT_Register = 0x1B20;

   AO_Output_Control_Register = 0x0000;

   AO_START_Select_Register = 0x 0000;

   Joint_Reset_Register
   AO configuration start = 0;
   AO configuration end = 1;

9. Call `AO_Triggering` to program the trigger signal. Configure the DAQ-STC to trigger once and use a software START1 trigger.

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Mode_1_Register
   AO trigger once = 1;

AO_Trigger_Select_Register
AO START1 select = 0;
AO START1 polarity = 0;
AO START1 edge = 1;
AO START1 sync = 1;

AO_Mode_3_Register
AO trigger length = 0;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

10. Call `AO_Counting` to program the buffer size and the number of buffers. Configure the DAQ-STC for non-continuous operation (AO will stop on BC_TC). Load the BC counter with 4 (output the buffer 5 times). Load the UC counter with 3000 (the first buffer contains 3000 points). Write 2999 to UC Load Register A (each subsequent buffer contains 3000 points).

    Joint_Reset_Register
    AO configuration start = 1;

    AO_Mode_1_Register
    AO continuous = 0;

    AO_Mode_2_Register
    AO BC initial load source = 0;

    AO_BC_Load_A_Registers (24 bits)
    Number of buffers -1 = 4;

    AO_Command_1_Register
    AO BC load =1;

    AO_Mode_2_Register
    AO UC initial load source = 0;

    AO_UC_Load_A_Registers (24 bits)
    Points per buffer = 3000;

    AO_Command_1_Register
    AO UC load = 1;

    AO_UC_Load_A_Registers (24 bits)
    Points per buffer - 1 = 2999;

    Joint_Reset_Register
    AO configuration start = 0;
    AO configuration end = 1;

11. Call `AO_Updating` to program the update interval. Use the internal UPDATE mode. Set the UI source to AO_IN_TIMEBASE1. Load the

UI counter with 1 (minimum delay from the START1 to the first UPDATE). Write 0x9C40 to UI Load Register A (2 ms update interval).

Joint_Reset_Register
AO configuration start = 1;

AO_Command_2_Register
AO BC gate enable = 0;

AO_Mode_1_Register
AO UPDATE source select = 0;
AO UPDATE source polarity = 0;

AO_Mode_1_Register
AO UI source select = 0;
AO UI source polarity = 0;

AO_Mode_2_Register
AO UI initial load source = 0;
AO UI reload mode = 0;

AO_UI_Load_A_Registers (24 bits)
AO UI special ticks -1 = 1;

AO_Command_1_Register
AO UI load = 1;

AO_UI_Load_A_Registers (24 bits)
AO UI ordinary ticks - 1 = 9C40;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

12. Call `AO_Channels` to select single channel output.

Joint_Reset_Register
AO configuration start = 1;

AO_Mode_1_Register
AO multiple channels = 0;

AO_Output_Control_Register
AO number of channels = 1;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

13. Call `AO_LDAC_Source_And_Update_Mode` to configure LDAC1 to output UPDATE in the timed update mode.

    Joint_Reset_Register
        AO configuration start = 1;

    AO_Command_1_Register
        AO LDAC0 source select = 0;
        AO DAC0 update mode = 1;
        AO LDAC1 source select = 0;
        AO DAC1 update mode = 1;

    Joint_Reset_Register
        AO configuration start = 0;
        AO configuration end = 1;

14. Call `AO_Errors_To_Stop_On` to configure the analog output to stop on overrun error.

    Joint_Reset_Register
        AO configuration start = 1;

    AO_Mode_3_Register = 0x0020;

    Joint_Reset_Register
        AO configuration start = 0;
        AO configuration end = 1;

15. Call `AO_FIFO` to disable the FIFO retransmit.

    Joint_Reset_Register
        AO configuration start = 1;

    AO_Mode_2_Register
        AO FIFO retransmit enable = 0;

    Joint_Reset_Register
        AO configuration start = 0;
        AO configuration end = 1;

16. Call `Kick_Start_FIFO` to initialize the virtual FIFO boards.

    if (VirtualFIFO)
        AO_DAC_FIFO_Data = 0;

17. Call `AO_Arming` to arm the counters and preload the DAC with the first analog output value.

    AO_Mode_3_Register
        AO not an UPDATE = 1;

    AO_Mode_3_Register
        AO not an UPDATE = 0;

*If (!VirtualFIFO)*
Wait until DACs have been preloaded.

AO_Command_1_Register = 0x0554;

18. Call `AO_Start_The_Acquisition` to pulse the software START1 trigger.

    AO_Command_2_Register
    AO START1 pulse = 1;

19. Poll the AO_FIFO_Full_St bit and write data from the array into the data FIFO whenever the data FIFO is not full.

```
while (there is more data to write)
{
    while (FIFO is not full)
    {
        AO_DAC_FIFO_Data = data;
    }
}
```

## Example 3

This example generates a waveform using local buffer mode. This example does not support the PCI-MIO-16XE-50 boards, because it has virtual analog output FIFOs.

Initialize the data FIFO with a 100 point buffer. Output the buffer 50 times. The update interval is 100 µs. Confirm operation with an oscilloscope.

1. Perform Analog Output Example 2 Steps 1 through 9.
2. Call `AO_Counting` to program the buffer size and the number of buffers. Configure the DAQ-STC for non-continuous operation (AO will stop on BC_TC). Load the BC counter with 49 (output the buffer 50 times). Load the UC counter with 100 (the first buffer contains 100 points). Write 99 to UC Load Register A (each subsequent buffer contains 100 points).

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Mode_1_Register
   AO continuous = 0;

   AO_Mode_2_Register
   AO BC initial load source = 0;

   AO_BC_Load_A_Registers (24 bits)
   Number of buffers -1 = 49;

AO_Command_1_Register
AO BC load =1;

AO_Mode_2_Register
AO UC initial load source = 0;

AO_UC_Load_A_Registers (24 bits)
Points per buffer = 100;

AO_Command_1_Register
AO UC load = 1;

AO_UC_Load_A_Registers (24 bits)
Points per buffer - 1 = 99;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

3. Call `AO_Updating` to program the update interval. Use the internal UPDATE mode. Set the UI source to AO_IN_TIMEBASE1. Load the UI counter with 1 (minimum delay from the START1 to the first UPDATE). Write 1999 to UI Load Register A (100 µs update interval).

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Command_2_Register
   AO BC gate enable = 0;

   AO_Mode_1_Register
   AO UPDATE source select = 0;
   AO UPDATE source polarity = 0;

   AO_Mode_1_Register
   AO UI source select = 0;
   AO UI source polarity = 0;

   AO_Mode_2_Register
   AO UI initial load source = 0;
   AO UI reload mode = 0;

   AO_UI_Load_A_Registers (24 bits)
   AO UI special ticks -1 = 1;

   AO_Command_1_Register
   AO UI load = 1;

   AO_UI_Load_A_Registers (24 bits)
   AO UI ordinary ticks - 1 = 1999;

   Joint_Reset_Register
   AO configuration start = 0;
   AO configuration end = 1;

4. Perform Analog Output Example 2 Steps 12 through 14.
5. Call `AO_FIFO` to enable the FIFO retransmit.

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Mode_2_Register
   AO FIFO retransmit enable = 1;

   Joint_Reset_Register
   AO configuration start = 0;
   AO configuration end = 1;
6. Call `AO_Arming` to arm the counters and preload the DAC with the first analog output value.

   AO_Mode_3_Register
   AO not an UPDATE = 1;

   AO_Mode_3_Register
   AO not an UPDATE = 0;

   *Wait until DACs have been updated.*

   AO_Command_1_Register = 0x0554;
7. Call `AO_Start_The_Acquisition` to pulse the software START1 trigger.

   AO_Command_2_Register
   AO START1 pulse = 1;

## Example 4

This example generates a waveform using local buffer mode with an external UPDATE and external trigger. This example does not support those boards with a virtual FIFO due to the use of the local buffer mode, but the programming of the external UPDATE and START1 applies to all boards.

Initialize the data FIFO with a 100 point buffer. Output the buffer 50 times. The update interval is determined externally. Confirm operation with an oscilloscope.

1. Perform Analog Output Example 2 Steps 1 through 8.
2. Call `AO_Triggering` to program the trigger signal. Configure the DAQ-STC to trigger once. Set the START1 select to PFI6.

   Joint_Reset_Register
   AO configuration start = 1;

AO_Mode_1_Register
AO trigger once = 1;

AO_Trigger_Select_Register
AO START1 select = 7;
AO START1 polarity = 0;
AO START1 edge = 1;
AO START1 sync = 1;

AO_Mode_3_Register

AO trigger length = 0;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

3. Call `AO_Counting` to program the buffer size and the number of buffers. Configure the DAQ-STC for non-continuous operation (AO will stop on BC_TC). Load the BC counter with 49 (output the buffer 50 times). Load the UC counter with 100 (the first buffer contains 100 points). Write 99 to UC Load Register A (each subsequent buffer contains 100 points).

Joint_Reset_Register
AO configuration start = 1;

AO_Mode_1_Register
AO continuous = 0;

AO_Mode_2_Register
AO BC initial load source = 0;

AO_BC_Load_A_Registers (24 bits)
Number of buffers - 1 = 49;

AO_Commmand_1_Register
AO BC load = 0;

AO_Mode_2_Register
AO UC initial load source = 0;

AO_UC_Load_A_Registers (24 bits)
Points per buffer = 100;

AO_Command_1_Register
AO UC load = 1;

AO_UC_Load_A_Registers (24 bits)
Points per buffer - 1 = 99;

Joint_Reset_Register
AO configuration start = 0;
AO configuration end = 1;

4. Call `AO_Updating` to program the update interval.

   Use the external UPDATE mode.
   Set the UPDATE source to PFI5.

5. Perform Analog Output Example 3 Steps 4 through 7.

## Example 5

This example generates a waveform using interrupts to write data to the data FIFO.

Initialize the buffer with 3000 points. Use polled writes to write each point to the data FIFO. Updates occur every 2 ms. Output the buffer 5 times. Confirm operation with an oscilloscope.

Instead of installing the interrupt service routine (ISR) as an interrupt, this example emulates the operation of the interrupt by polling the status register in the DAQ-STC. When the status register indicates an interrupt, the main loop transfers control to the ISR. To use the example ISR as an actual interrupt, you need to learn how to install software interrupts on your system. Generally, the procedure is as follows:

1. Determine the software interrupt number corresponding to the IRQ line you are using.
2. Use the OS specific functions, such as `getvect()` and `setvect()` for DOS, to replace the default interrupt handler with your ISR. You should disable interrupts during this step.
3. Reset the interrupt controller hardware.
4. Perform your analog output.
5. After the analog output operation completes, you should re-install the default interrupt handler.

## Example Program

1. Perform Analog Output Example 2 Steps 1 through 14.
2. Call `AO_FIFO` to disable the FIFO retransmit and set the FIFO mode.

   Joint_Reset_Register
   AO configuration start = 1;

   AO_Mode_2_Register
   AO FIFO Mode = 1;
   AO FIFO retransmit enable = 0;

Joint_Reset_Register
    AO configuration start = 0;
    AO configuration end = 1;

3. Call `Kick_Start_FIFO` to initialize the virtual FIFO boards.

   if (VirtualFIFO)
       AO_DAC_FIFO_Data = 0;

4. Call `AO_Arming` to arm the counters and preload the DAC with the first analog output value.

   AO_Mode_3_Register
       AO not an UPDATE = 1;

   AO_Mode_3_Register
       AO not an UPDATE = 0;

   if (!VirtualFIFO)
       Wait until DACs have been preloaded.

   AO_Command_1_Register = 0x554;

5. Program the DAQ-STC to generate interrupts on the FIFO condition.

   Interrupt_B_Enable_Register
       AO FIFO interrupt enable = 1;

   Interrupt_Control_Register
       Interrupt B output select = IRQ number;
       Interrupt B enable = 1;

6. Install the interrupt service routine to handle the interrupt.

   service_interrupt()

   ```
   Do
   {
       If (AO FIFO not full) {
           AO_DAC_FIFO_Data = data;
           increment data index counter;

       }
   } while (AO FIFO not full && total point have not been written)
   ```

7. Call `AO_Start_The_Acquisition` to pulse the software START1 trigger.

   AO_Command_2_Register
       AO START1 pulse = 1;

8. Poll the AO FIFO half full flag in the AO_Status_1_Register until half full and call the ISR.

   If (AO FIFO half full) then
       call service_interrupt;

# General-Purpose Counter/Timer

There are two 24-bit up/down counters available for performing event counting, pulse-width measurement, pulse and pulse train generation, etc. Associated with each counter are load and save registers. You can use interrupts with these counters to do buffered measurements. Each time the counter generates an interrupt, a new set of values can be loaded into the counters.

You can select the input signals to the counters from any of the 17 (10 PFI and 7 RTSI) external timing I/O pins. Signals can also be output on certain dedicated lines. For example, the pin name PFI8/CTRSRC0 means that the clock source input for Counter 0 can come from any of the PFI/RTSI lines but can be output only on the PFI8 line.

Chapter 4 of the *DAQ-STC Technical Reference Manual* contains all the information on the DAQ-STC general-purpose counter and timer module, with specific programming steps in the *Programming Information* section. Example 1 shows simple gated event counting. Example 2 shows buffered pulsewidth measurement, and Example 3 provides the framework for continuous pulse generation.

## Example 1

This is the example for gated event counting.

G0 counter counts the number of pulses (rising edge) that occur on the G_Source after software arm, using PFI3 as G_Source and PFI4 as G_Gate. The signal from G_Gate starts and stops the counter, allowing the counter to count only when G_Gate is high. After the software arm, read from the save register and display the counter contents. Counting and reading stop when the counter reaches 10,000.

1. Set up the PCI board. Use the `Setup_Mite()` function provided on the Companion Disk.
2. Call `MSC_IO_Pin_Configure()` to set all the PFI pins for input.

   ```
   IO_Bidirection_Pin_Register
       BD_i_Pin_Dir <= 0;
   ```

3. Call `G0_Reset_All()` to reset all the necessary registers in DAQ-STC.

   Joint Reset Register
   G0_Reset=1;

G0_Mode_Register=0x0000;

G0_Command_Register=0x0000;

G0_Input_Select_Register=0x0000;

G0_Autoincrement_Register=0x0000;

Interrupt_A_Enable_Register=0x0000;

G0_Command_Register
    G0_Synchronized_Gate =1;

Interrupt_A_Ack_Register=0xc060;

G0_Autoincrement_Register
    G0_Autoincrement=0;

4. Call `Simple_Gated_Count()` to set up DAC-STC for simple counting.

   Go_Mode_Register
       G0_Load_Source=0;

   G0_Load_A_Registers (24 bits)
       G0_Load_A=0x0000; //initial counter value

   G0_Command_Register
       G0_Load=1;

   G0_Input_Select_Register
       G0_Source_Select=4; (PFI3)
       G0_Source_Polarity=0; (rising edges)
       G0_Gate_Select=5; (PFI4)
       G0_OR_Gate=0;
       G0_Output_Polarity=0; (active high)
       G0_Gate_Select_Load_Source=0;

   G0_Mode_register
       G0_Output_Mode=1; (one clock cycle output)
       G0_Gate_Polarity = 1; (enable inversion)
       G0_Loading_On_Gate = 0;
       G0_Loading_On_TC = 0;
       GO_Gating_Mode = 1;
       G0_Gate_On_Both_Edges = 0;
       GO_Trigger_Mode_For_Edge_Gate = 2;
       G0_Stop_Mode = 0;
       G0_Counting_Once = 0;

   G0_Command_Register
       G0_Up_Down = 1; (up counting)
       G0_Bank_Switch_Enable = 0;
       G0_Bank_Switch_Mode = 0;

Interrupt_A_Enable_Register
G0_TC_Interrupt_Enable = 0;
G0_Gate_Interrupt_Enable = 0;

5. Call `G0_Arm()` to begin the operation.

   G0_Command_Register
   G0_Arm=1;

6. Call `G0_Watch()` to read the save registers.

   do {

   G0_Command_Register
   G0_Save_Trace=0;

   G0_Command_Register
   G0_Save_Trace=1;

   /* Compare the counter content; if they are not the same, read again*/

   save_1=G0_Save_Registers (24 bits);

   save_2=G0_Save_Registers (24 bits);

   if (save_1!= save_2)
   save_1=G0_Save_Registers (24 bits);

   } while (save_1<=10000); // Count until it exceeds 10000

## Example 2

This is the example for buffered pulsewidth measurement.

The counter uses G_In_TimeBase as G_Source to measure the signal's pulsewidth on PFI4 (G_Gate), counting the number of edges that occur on G_Source. At the completion of each pulsewidth interval for G_Gate, software reads the counter values from the HW_Save_Registers. An interrupt occurs after each measurement. Readings are done after each generated interrupt.

For more information about how to install the software interrupt, please see Example 5 in the *Analog Output* section, or Example 3 in the *Analog Input* section.

1. Perform General Purpose Counter and Timer Example 1 Step 1 through 3.
2. Call `Buffered_Pulse_Width_Measurement()` to set up the DAQ-STC for buffered pulse width measurement.

   Go_Mode_Register
   G0_Load_Source=0;

G0_Load_A_Registers (24 bits)
G0_Load_A=0x0000; //initial counter value
G0_Command_Register
G0_Load=1;

G0_Input_Select_Register
G0_Source_Select=0; (G_In_TimeBase)
G0_Source_Polarity=0; (rising edges)
G0_Gate_Select=5; (PFI4)
G0_OR_Gate=0;
G0_Output_Polarity=0; (active high)
G0_Gate_Select_Load_Source=0;

G0_Mode_register
G0_Output_Mode=1; (one clock cycle output)
G0_Gate_Polarity = 1; (enable inversion)
G0_Loading_On_Gate = 1;
G0_Loading_On_TC = 0;
GO_Gating_Mode = 1;
G0_Gate_On_Both_Edges = 0;
GO_Trigger_Mode_For_Edge_Gate = 3;
G0_Stop_Mode = 0;
G0_Counting_Once = 0;
G0_Command_Register
G0_Up_Down = 1; (up counting)
G0_Bank_Switch_Enable = 0;
G0_Bank_Switch_Mode = 0;

Interrupt_A_Enable_Register
G0_TC_Interrupt_Enable = 0;
G0_Gate_Interrupt_Enable = 1;

3. Call `G0_Arm()` to begin the operation.

   G0_Command_Register
   G0_Arm=1;

4. `Pulse_Width_Measurement_ISR()` performs the reading from HW_Save Register.

   save_1=G0_HW_Registers (24 bits);

   if (G0_Stale_Data_St==1) then
   save_1=0;

```
if (buffer is not done and buffer is not full) then
        {
    current buffer value =save_1;
    increase buffer pointer;
}
if (all the points have been written into the buffer) then
{
    G0_Command_Register
        G0_Disarm=1;
        indicate buffer done;
}
    Interrupt_A_Ack_Register
        G0_Gate_Interrupt_Ack <=1;
/*read G_Status_Register and check for the G0_Gate_Error_St bit if
the bit is set means the hardware saves are too fast */
    if (G0_Gate_Error_St==1)
        {
            Interrupt_A_Ack_Register
            G0_Gate_Error_Confirm <=1;
            }
    if (G0_TC_St==1){
/*rollover error - counter value is not correct */
        confirm user rollover has occurred.
        Interrupt_A_Ack_Register
            G0_TC_Interrupt_Ack<=1;
        }
```

5. Call `ISR` in a do-while loop.

```
do
{
    call Buffered_Pulse_Width_Measurement_ISR();
    } while (the buffer is not done);
print out the buffer values.
```

## Example 3

This is the example for continuous pulse train generation.

It generates continuous pulses on the `G_Out` pin with a three clock delay from the trigger, pulse interval of four clocks, and pulsewidth of three clocks. `G_in_timebase` (20 MHz) is `G_source`. The waveform

generation begins by a trigger signal from G_Gate on PFI4. Confirm this operation with an oscilloscope.

1. Perform General Purpose Counter and Timer Example 1 Step 1 through 3.
2. Call MSC_Clock _Configure() to set up the G_In_timebase2.

   Clock_and_FOUT_Register
   Slow_Internal_Timebase = 1
   Slow_Internal_Time_Divide_By_2 = 1
   Clock_To_Board = 1
   Clock_To_Board_Divide_By_2 = 1
3. Call Cont_Pulse_Train_Generation() to set up the DAQ-STC for continuous pulse train generation.

   Go_Mode_Register
   G0_Load_Source=0;

   G0_Load_A_Registers (24 bits)
   G0_Load_A=0x0002; //delay from the trigger -1
   G0_Command_Register
   G0_Load=1;
   G0_Load_A_Registers (24 bits)
   G0_Load_A=0x0003; //pulse interval -1

   G0_Load_B_Registers (24 bits)
   G0_Load_B=0x0002; //pulse width -1

   G0_Mode_Register
   G0_Load_Source_Select=1;

   G0_Input_Select_Register
   G0_Source_Select=0; (G_In_TimeBase)
   G0_Source_Polarity=0; (rising edges)
   G0_Gate_Select=5; (PFI4)
   G0_OR_Gate=0;
   G0_Output_Polarity=0; (active high)
   G0_Gate_Select_Load_Source=0;

   G0_Mode_register

G0_Output_Mode=2; (toggle on TC)
G0_Gate_Polarity = 1; (enable inversion)
G0_Reload_source_switching=1;
G0_Loading_On_Gate = 0;
G0_Loading_On_TC = 1;
GO_Gating_Mode = 2;
G0_Gate_On_Both_Edges = 0;
GO_Trigger_Mode_For_Edge_Gate = 2;
G0_Stop_Mode = 0;
G0_Counting_Once = 0;

G0_Command_Register
G0_Up_Down = 0; (down counting)
G0_Bank_Switch_Enable = 0;

or

G0_Bank_Switch_Enable = 1
if you want to change rate. Then call
G0_Seamless_Pulse_Train_Change()
G0_Bank_Switch_Mode = 0;

Interrupt_A_Enable_Register
G0_TC_Interrupt_Enable = 0;
G0_Gate_Interrupt_Enable = 0;

4. Call `G0_Out_Enable()` to enable `GPCTR0_Out` pin.

   Analog_Trigger_Etc_Register
   GPFO_0_Output_Enable =1;
   GPFO_0_Output_Select=0;

5. Call `G0_Arm()` to begin the operation.

   G0_Command_Register
   G0_Arm=1;

6. Call `G0_Seamless_Pulse_Train()` to change the pulse rate during the operation.

```
/*Check if you can legally change the rate. You cannot
change the rate twice in a row before
generation of at least one cycle of intermediate
frequency*/
```

if ((G0_Bank_St==1)==g_bank_to_be_used) then
{
G0_Load_A_Register (24 bits)
G0_Load_A <= pulse interval -1 (3)

G0_Load_B_Register
G0_Load_B <= pulsewidth -1 (3)

```
G0_Command_Register
      G0_Bank_Switch_Start<=1
if (g_bank_to_be_used == 0)
      g_bank_to_be_used =1;
else
      g_bank_to_be_used =0;
}
else{
inform the user the wave rate cannot be changed
}
```

# RTSI Trigger Lines Programming Considerations

For detailed descriptions of RTSI and RTSI programming, see Chapter 6, *RTSI Trigger*, of the *DAQ-STC Technical Reference Manual*. Also, see the same section of the DAQ-STC manual for a description of the signal lines referred to in this chapter.

There are seven 12 to 1 muxes that drive the seven RTSI lines. Any of the RTSI lines can be driven with any of eight internally generated timing signals or with any of the four RTSI board signals. Similarly, there are four 8 to 1 muxes that can drive the four RTSI board signals with any of the seven RTSI trigger signals and the AISTART or AISTOP signal.

See the DAQ-STC manual for information on programming the RTSI interface. The function `MSC_RTSI_Pin_Configure` is very straightforward and easy to use.

Notice that of the four RTSI board signals, only RTSI_BRD0 is connected to GENTRIG0. The remaining three are unused. The GENTRIG0 signal is a logical AND of the CONVERT pulse and the DOTRIG0 bit in the configuration memory. When the DOTRIG0 is set, the GENTRIG line is driven by the CONVERT signal. This can be driven out to the RTSI lines and used by some other board for synchronization purposes.

# Analog Triggering

All of the PCI E Series boards except the PCI-MIO-16XE-50, PCI-6023E, PCI-6024E, and PCI-6025E contain true analog triggering hardware, which provides fast slope and level detection, as well as window detection. The mode selection circuitry is in the DAQ-STC, while the analog comparison is performed by onboard circuitry.

Refer to the *Analog Trigger* section of Chapter 10, *Miscellaneous Functions*, in the *DAQ-STC Technical Reference Manual* for information about the analog trigger functionality of the DAQ-STC. The various modes, Low Window, High Window, Middle Window, and high and low hysteresis are discussed. The High Value and Low Value signals are controlled by two DACs. For the PCI-6052E, PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-6071E, these correspond to DACs 11[1] and 12 in the MB88341, which contain 8-bit DACs. For the PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E, these correspond to DACs 0 and 1 in the AD8522, which contains 12-bit DACs. For more information about the MB88341, and AD8522, see Chapter 5, *Calibration*, in this manual.

One of two sources may be used as the trigger source, the PFI0/TRIG1 input on the I/O connector or an analog input passing through the PGIA. The PGIA allows you to apply gain to an external signal for more flexible triggering conditions. The PFI0/TRIG1 pin has an input voltage range of ±10 V. The Int/Ext Trig bit in the Misc Command Register controls which input is used. The PGIA output is selected when this bit is set, and the PFI0/TRIG1 input is selected when this bit is cleared. When the PFI0/TRIG1 input is being used for an analog signal, it must be disconnected from the DAQ-STC PFI input. You can do this by clearing the Analog_Trigger_Drive bit in the DAQ-STC. When the Analog_Trigger_Drive bit is set, the PFI0/TRIG1 pin is connected to PFI0 on the DAQ-STC.

The high and low thresholds are set by the DACs to be within plus/minus full scale. For the 8-bit DACs, writing 0x00 sets it for + full scale, while writing 0xFF sets it to – full scale. For the 12-bit DACs, writing 0x000 sets it for + full scale, while writing 0xFFF sets it to – full scale. The selected input is compared against each of these thresholds by a comparator. The outputs of each comparator go high when the input voltage is greater than the threshold. The outputs of these two comparators are connected to the DAQ-STC analog trigger inputs 0 and 1. The DAQ-STC circuitry uses these digital signals to generate the different slope and level detection modes. See the *DAQ-STC Technical Reference Manual* for their appropriate use. Figure 4-1 shows the analog trigger structure.

[1] When writing to CALDAC 11, also write the same value to CALDAC 0.

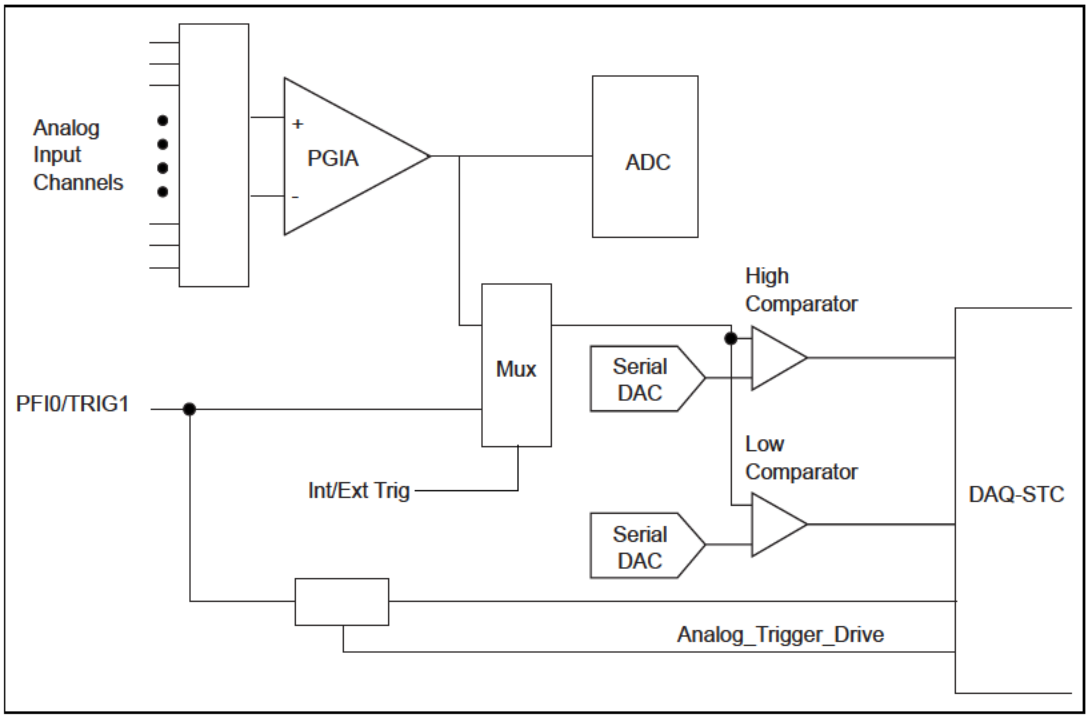



**Figure 4-1.** Analog Trigger Structure

To set the low and high analog thresholds, the DACs must be written with appropriate values. The protocol for writing to these DACs is described in Chapter 5, *Calibration*.

The function `Analog_Trigger_Control` enables analog triggering to set a mode of operation. When analog trigger is enabled, the analog trigger signal takes over the PFI0/TRIG1 slot, and this pin can no longer be used as an input. For more information about this function, see the *Programming Analog Trigger* section of Chapter 10 in the *DAQ-STC Technical Reference Manual*.

The following example is written for the PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-6071 boards, which use 8-bit CALDAC for the analog trigger; the PCI-6052E, PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E use an AD8522, which contains two 12-bit DACs. The PCI-MIO-16XE-50 does not support analog triggering. The write cycle for the 8-bit and 12-bit DACs is illustrated in Chapter 5 of this manual in the *Calibration DACs* section. The example uses low-hysteresis mode and the PGIA as the triggering source. The low value is set to be 0 V,

and the high value is set to be relative 1.25. The software scans the analog channel 0 100 times. Each scan is at a gain of 1 and in RSE mode. Within each scan, the sample interval should be 100 µs. Dithering should remain off during the acquisition. Use polled input to obtain the data.

1. Perform Step 1 of the *Analog Input* Example 1.
2. Call `Configure_Board()` to clear all of the necessary registers bits.

   Write_Strobe_0_Register
   Write strobe 0 =1;

   Write_Strobe_1_Register
   Write strobe 1=1;

   Configuration_Memory_High_Register
   Channel Number=0;
   Channel type =3;

   Configuration_Memory_Low_Register
   Last Channel =1;
   Gain =1;
   Polarity =0;
   Dither enable =0;
3. Perform Steps 3 through 9 of *Analog Input* Example 1.
4. Call the function `Number_of Scans()` to load the number of scans.

   Joint_Reset_Register
   AI configuration start = 1;

   AI_SC_Load_A_Register(24 bits)
   Number of postrigger scan -1 = 99;

   AI_Command_1_Register
   AI SC Load=1;

   Joint_Reset_Register
   AI configuration start = 0;
   AI configuration end=1;
5. Perform Steps 5 through 8 in *Analog Input* Example 2.
6. Call the `Analog Trigger Control`.

   Analog_Trigger_Etc_Register
   Analog_Trigger_Mode =6 (low hysteresis);
   Analog_Trigger_Drive=0;
   Analog_Trigger_Enable=1(Enable);

   Mis_Command_Register (8 bits)
   Int/Ext Trigger =1 (PGIA2)

Writing to Serial CALDAC11
CALDAC11=0x80;

Writing to Serial CALDAC0
CALDAC0 = 0x80;

Writing to Serial CALDAC12
CALDAC12=0xao;

7. Perform Steps 9 and 10 in *Analog Input* Example 2.
8. Poll the AIFIFO *not empty* flag in the AI_Status_Register until *not empty*, then *read the data* in the ADC_FIFO_Data_Register.

Do{

If (AIFIFO not empty) then
read FIFO data;

}while (100 samples have not been read)

# Interrupt Programming

Chapter 8, *Interrupt Control*, in the *DAQ-STC Technical Reference Manual*, discusses the interrupt programming aspect of the PCI E Series boards.

There are two groups—Interrupt Group A and Interrupt Group B. Group A handles the analog input interrupts, general-purpose Counter 0 interrupts, and one pass-through interrupt. The Group A pass-through interrupt is not used. Group B handles the analog output interrupts, general purpose counter 1 interrupts, and one pass-through interrupt. The Group B pass-through interrupt is not used.

The `MSC_IRQ_Configure` function, found in the *DAQ-STC Technical Reference Manual*, should select the IRQ_OUT0 line for Group A and for Group B. The MITE maps these interrupts to the INTA line on the PCI bus. The *Interrupt Control* section also describes two interrupt programs, one for Group A and one for Group B, which are skeletons of the actual interrupt service routines. These programs do not address the programming of the interrupt controller.

# Interrupt Sharing

It is possible for multiple PCI boards to share the same interrupt. In this case the interrupt service routine (ISR) must be daisy-chained such that

they determine if the pending interrupt belongs to their device. If it does not, the ISR must pass control to the next ISR in the chain.

In order to determine if a PCI E Series board has a pending interrupt do the following:

1. Perform a 32 bit memory read from BAR0 + 0x14.
2. Check the status of bit 31 (highest order bit in the register). If this bit is high the PCI E Series board is currently asserting an interrupt.

# DMA Programming

You can program your PCI E Series board so that the analog input, analog output, or general purpose counter/timers can generate DMA requests under appropriate circumstances. There are four logical DMA channels—A, B, C, and D.

Each logical channel, in turn, can service either analog input, analog output, or the general-purpose counter/timers. You must program the AO AI Select Register (address 0x09) and the G0G1 Select Register (address 0x0B) to assign particular logical channels to either AI, AO, or GPCTs. Figure 4-2 shows the three-stage DMA structure.

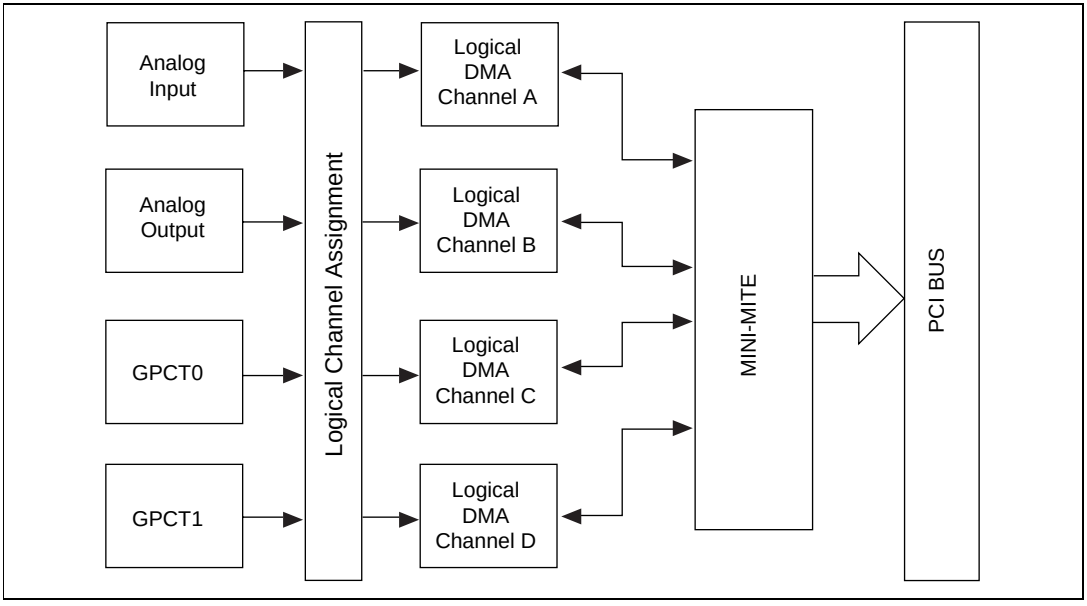



**Figure 4-2.** DMA Structure

Make sure that the same logical channel is not assigned to more than one resource.

DMA requests are generated when all of the above mentioned initializations are done and when the source is programmed appropriately. For analog input, the AIFREQ signal from the DAQ-STC is used as the DMA source. For more information, see Chapter 2 of the *DAQ-STC Technical Reference Manual*. By using the `FIFO_Request_Selection` function, you can generate DMA requests based on any of the following conditions:

- FIFO not empty
- FIFO half-full
- FIFO full
- FIFO half-full until FIFO is empty

For analog output, the AOFREQ signal from the DAQ-STC is used as the DMA source. By using the `AO_FIFO` function you can generate DMA requests based on any of the following conditions:

- FIFO empty
- FIFO less than half-full
- FIFO not full
- Assert on FIFO half-full and deassert on FIFO full

For general-purpose counter/timers, an interrupt is produced in buffered modes, such as the buffered event counting, the buffered period measurement, and so on. The secondary bank of interrupts in the DAQ-STC is used for generating DMA requests for the general-purpose counter/timers.

## The Link Chaining Mode for DMA Transfer

The MITE contains DMA channels that may be used to transfer data to the I/O and CPU port. Each channel supports several modes of operation. One of the modes of operation is the Link Chaining Mode. It enables the user to do DMA transfers, limited only by the size of the memory space. This mode of operation is used in analog input Example 4. The Link Chaining Mode requires a linked list structure to store the information for each buffer. The linked list structure enables DMA transfers on different memory segments and makes seamless data transfer possible.

Inside the linked list structure, each node contains values for TCR, MAR, DAR, and LKAR. TCR (Transfer Count Register) stores the total number

of transfer bytes for each buffer; MAR (Memory Address Register) stores the buffer's physical address; DAR (Device Address Register) stores default values for simple operation; LKAR (Link Address Register) stores the physical address of the next node.

Before beginning the DMA transfers, LKAR must be loaded with the physical address of the first node since the MITE needs to have an entry point to access the link chain. After arming the DMA transfer, the MITE goes through the link chain and loads the buffer's physical address into MAR. Then the MITE transfers data from the FIFO to the buffers or from the buffer to the output port. This process continues until the MITE reaches the empty node, the node which contains all zeros.

Figure 4-3 illustrates the basic operation of the Link Chain Mode.

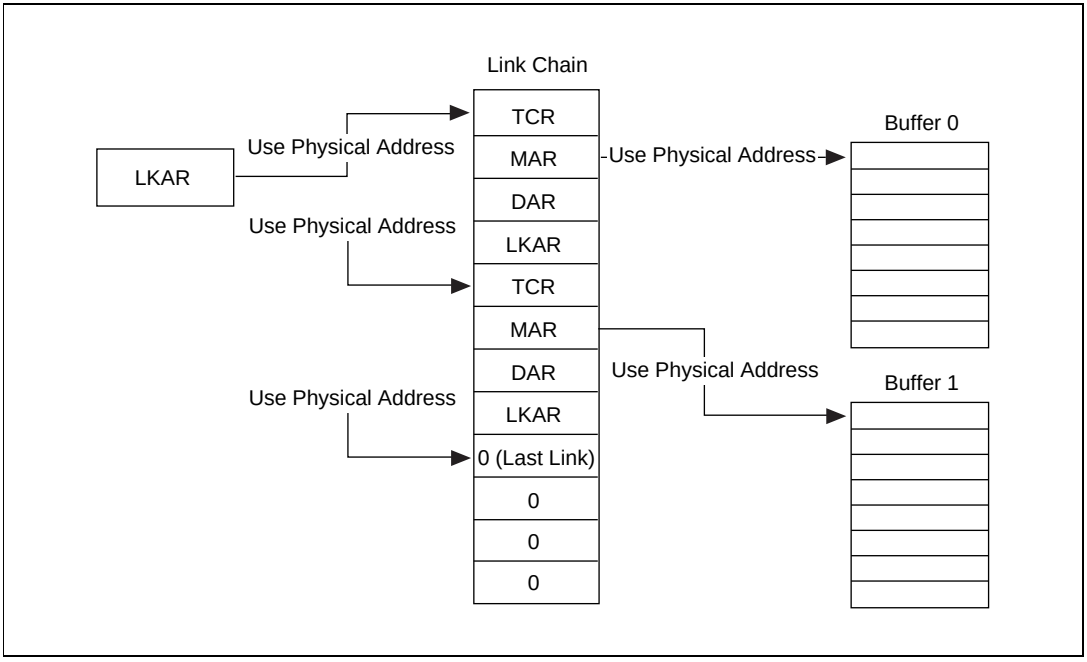



**Figure 4-3.** DMA Link Chaining Mode Structure

# 5

# Calibration

This chapter explains how to calibrate the analog input and output sections of the PCI E Series boards by reading calibration constants from the EEPROM and writing them to the calibration DACs. This chapter also explains how to generate the calibration constants using NI-DAQ.

All PCI E Series boards are factory calibrated before shipment, and the resulting calibration constants are stored in the EEPROM. Because the calibration DACs have no memory capability, they do not retain calibration information when the computer is turned off. Therefore, they must be reloaded every time the computer is turned on, and the most straightforward method is to copy these values from the EEPROM. In addition to the factory calibration constants, new calibration constants can be generated in the field through the use of the NI-DAQ calibration function call. Generating new constants results in a more accurate calibration for the actual environment in which the board is used.

## About the EEPROM

The EEPROM is used to store all non-volatile information about the board, including the factory and user calibration constants. The PCI E Series boards use a XICOR X25040 EEPROM, which is 512 by 8 bits in size and has a serial interface. The signals used to interface to the EEPROM are clock, data in, data out, and chip select.

The Serial Command Register has three bits—SerClk (bit 0), SerData (bit 1), and EEPROMCs (bit 2)—that are connected to the EEPROM clock, data in, and chip select pins, respectively. PROMOut (bit 0) of the Status Register is connected to the EEPROM data out pin.

The format for reading from the EEPROM is very straightforward. The basic read cycle consists of shifting a 7-bit instruction and 9-bit address into the EEPROM, then shifting an 8-bit data out of the EEPROM. The timing diagram for the read cycle is shown in Figure 5-1.

The EEPROM interface is relatively slow compared to the PCI bus. In order to meet the timing specifications of the EEPROM, you must double write

the clock bit. In other words, when setting the clock bit low, you must write 0 twice, and when setting it high, you must write 1 twice.

**Note** ***Review the timing diagram and specifications very carefully before attempting to write code. Do not attempt to write to the EEPROM. If the factory area of the EEPROM (the upper 128 bytes) is lost, the board can be rendered inoperable. In this situation, you will have to send the board back to National Instruments to be reprogrammed. National Instruments is NOT liable for such mistakes, and you will have to bear the full expense of the RMA.***

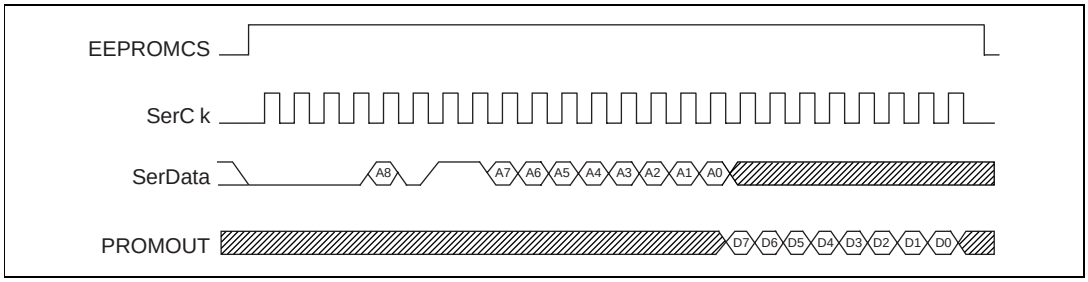



**Figure 5-1.** EEPROM Read Timing

Notice that because the CalDACs and the EEPROM share the same clock and data in lines, it might seem that writing to the CalDACs could result in accidental writes to the EEPROM. However, this is not true. A write cycle to the EEPROM needs the chip select bit asserted. While writing to the CalDACs, make sure that this bit is cleared. Clearing this bit ensures that no writes to the EEPROM occur. It might also seem as though an access to the EEPROM could result in an access to the CalDACs, but this is also not true. The CalDACs update only when the LdCalDAC<2..0> bit is pulsed.

Tables 5-1 through 5-6 show a selected portion of the EEPROM map for each of the boards. The upper 256 bytes are divided into two sections of 128 locations each. The uppermost 128 locations contain factory data. The lower section of 128 locations contains the user calibration constants. There are five user calibration constants sections that use a format identical to the factory calibration section. These user areas start at location 371 in the EEPROM.

**Table 5-1.** PCI-MIO-16E-1, PCI-MIO-16E-4, PCI-6071E EEPROM Map

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 511 | NI-DAQ Board Code[1] | — | — | — | — |
| 510 | Revision | — | — | — | — |
| 509 | Sub-revision | — | — | — | — |
| 508 | Year of last factory fabrication | — | — | — | — |
| 507 | Month of last factory fabrication | — | — | — | — |
| 506 | Day of last factory fabrication | — | — | — | — |
| 426 | Factory reference MSB | — | — | — | — |
| 425 | Factory reference LSB | — | — | — | — |
| 424 | CALDAC factory constant | AI | MB88341 (8-bit) | 4 | 0 |
| 423 | CALDAC factory constant | AI | MB88341 (8-bit) | 1 | 0 |
| 422 | CALDAC factory constant | AI | MB88341 (8-bit) | 3 and 14[2] | 0 |
| 421 | CALDAC factory constant | AI | MB88341 (8-bit) | 2 | 0 |
| 420 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 5 | 0 |
| 419 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 7 and 13[2] | 0 |
| 418 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 6 | 0 |
| 417 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 8 | 0 |
| 416 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 10 | 0 |

**Table 5-1.** PCI-MIO-16E-1, PCI-MIO-16E-4, PCI-6071E EEPROM Map (Continued)

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 415 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 9 | 0 |
| 414 | CALDAC factory constant | Unipolar AO | MB88341 (8-bit) | 5 | 0 |
| 413 | CALDAC factory constant | Unipolar AO | MB88341 (8-bit) | 7 | 0 |
| 412 | CALDAC factory constant | Unipolar AO | MB88341 (8-bit) | 6 | 0 |
| 411 | CALDAC factory constant | Unipolar AO | MB88341 (8-bit) | 8 | 0 |
| 410 | CALDAC factory constant | Unipolar AO | MB88341 (8-bit) | 10 | 0 |
| 409 | CALDAC factory constant | Unipolar AO | MB88341 (8-bit) | 9 | 0 |
| 408 | Factory calibration temperature | — | — | — | — |
| 371 | Start of the five user calibration sections | — | — | — | — |

[1]Board Codes:

205—PCI-MIO-16E-1
206—PCI-MIO-16E-4
207—PCI-6071E

[2] Write both CALDAC addresses for these constants

**Table 5-2.** PCI-MIO-16XE-50 EEPROM Map

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 511 | NI-DAQ Board Code: 202 | — | — | — | — |
| 510 | Revision | — | — | — | — |
| 509 | Sub-revision | — | — | — | — |
| 508 | Year of last factory fabrication | — | — | — | — |
| 507 | Month of last factory fabrication | — | — | — | — |
| 506 | Day of last factory fabrication | — | — | — | — |
| 438 | Factory reference MSB | — | — | — | — |
| 437 | Factory reference LSB | — | — | — | — |
| 436 | CALDAC factory constant MSB | Bipolar AI | DAC8043 (12-bit) | — | 1 |
| 435 | CALDAC factory constant LSB | Bipolar AI | DAC8043 (12-bit) | — | 1 |
| 434 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 2 | 0 |
| 433 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 0 | 0 |
| 432 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 1 | 0 |
| 431 | CALDAC factory constant MSB | Unipolar AI | DAC8043 (12-bit) | — | 1 |
| 430 | CALDAC factory constant LSB | Unipolar AI | DAC8043 (12-bit) | — | 1 |
| 429 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 2 | 0 |
| 428 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 0 | 0 |

**Table 5-2.** PCI-MIO-16XE-50 EEPROM Map (Continued)

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 427 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 1 | 0 |
| 426 | CALDAC factory constant | AO | 8800 (8-bit) | 6 | 0 |
| 425 | CALDAC factory constant | AO | 8800 (8-bit) | 4 | 0 |
| 424 | CALDAC factory constant | AO | 8800 (8-bit) | 7 | 0 |
| 423 | CALDAC factory constant | AO | 8800 (8-bit) | 5 | 0 |
| 422 | Factory calibration temperature | — | — | — | — |
| 371 | Start of the five user calibration sections | — | — | — | — |

**Table 5-3.** PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E EEPROM Map

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 511 | NI-DAQ Board Code[1] | — | — | — | — |
| 510 | Revision | — | — | — | — |
| 509 | Sub-revision | — | — | — | — |
| 508 | Year of last factory fabrication | — | — | — | — |
| 507 | Month of last factory fabrication | — | — | — | — |
| 506 | Day of last factory fabrication | — | — | — | — |
| 431 | Factory reference MSB | — | — | — | — |
| 430 | Factory reference LSB | — | — | — | — |
| 429 | CALDAC factory constant MSB | Bipolar AI | DAC8043 (12-bit) | — | 1 |
| 428 | CALDAC factory constant LSB | Bipolar AI | DAC8043 (12-bit) | — | 1 |
| 427 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 2 | 0 |
| 426 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 3 | 0 |
| 425 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 0 | 0 |
| 424 | CALDAC factory constant | Bipolar AI | 8800 (8-bit) | 1 | 0 |
| 423 | CALDAC factory constant MSB | Unipolar AI | DAC8043 (12-bit) | — | 1 |
| 422 | CALDAC factory constant LSB | Unipolar AI | DAC8043 (12-bit) | — | 1 |
| 421 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 2 | 0 |

**Table 5-3.** PCI-MIO-16XE-10, PCI-6031E, PCI-6032E, and PCI-6033E EEPROM Map (Continued)

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 420 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 3 | 0 |
| 419 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 0 | 0 |
| 418 | CALDAC factory constant | Unipolar AI | 8800 (8-bit) | 1 | 0 |
| 417 | CALDAC factory constant | Bipolar AO | 8800 (8-bit) | 6 | 0 |
| 416 | CALDAC factory constant | Bipolar AO | 8800 (8-bit) | 4 | 0 |
| 415 | CALDAC factory constant | Bipolar AO | 8800 (8-bit) | 7 | 0 |
| 414 | CALDAC factory constant | Bipolar AO | 8800 (8-bit) | 5 | 0 |
| 413 | CALDAC factory constant | Unipolar AO | 8800 (8-bit) | 6 | 0 |
| 412 | CALDAC factory constant | Unipolar AO | 8800 (8-bit) | 4 | 0 |
| 411 | CALDAC factory constant | Unipolar AO | 8800 (8-bit) | 7 | 0 |
| 410 | CALDAC factory constant | Unipolar AO | 8800 (8-bit) | 5 | 0 |
| 409 | Factory calibration temperature | — | — | — | — |
| 371 | Start of the five user calibration sections | — | — | — | — |

[1]Board Codes:

204—PCI-MIO-16XE-10
220—PCI-6031E
221—PCI-6032E
222—PCI-6033E

**Table 5-4.** PCI-6023E EEPROM Map

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 511 | NI-DAQ Board Code LSB:11[1] | — | — | — | — |
| 474 | NI-DAQ Board Code MSB:1[1] | — | — | — | — |
| 510 | Revision | — | — | — | — |
| 509 | Sub-revision | — | — | — | — |
| 508 | Year of last factory fabrication | — | — | — | — |
| 507 | Month of last factory fabrication | — | — | — | — |
| 506 | Day of last factory fabrication | — | — | — | — |
| 444 | Factory reference MSB | — | — | — | — |
| 443 | Factory reference LSB | — | — | — | — |
| 442 | CALDAC factory constant | AI | MB88341 (8-bit) | 4 | 0 |
| 441 | CALDAC factory constant | AI | MB88341 (8-bit) | 11 | 0 |
| 440 | CALDAC factory constant | AI | MB88341 (8-bit) | 1 | 0 |
| 439 | CALDAC factory constant | AI | MB88341 (8-bit) | 2 | 0 |
| 438 | Factory calibration temperature | — | — | — | — |
| 371 | Start of the five user calibration sections | — | — | — | — |
| [1]Board Codes | (MSB × 256) + LSB = Board Code<br>(1 × 256) + 11 = 267 | | | | |

**Table 5-5.** PCI-6024E and PCI-6025E EEPROM Map

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 511 | NI-DAQ Board Code LSB[1] | — | — | — | — |
| 474 | NI-DAQ Board Code MSB[2] | — | — | — | — |
| 510 | Revision | — | — | — | — |
| 509 | Sub-revision | — | — | — | — |
| 508 | Year of last factory fabrication | — | — | — | — |
| 507 | Month of last factory fabrication | — | — | — | — |
| 506 | Day of last factory fabrication | — | — | — | — |
| 432 | Factory reference MSB | — | — | — | — |
| 431 | Factory reference LSB | — | — | — | — |
| 430 | CALDAC factory constant | Bipolar AI | MB88341 (8-bit) | 4 | 0 |
| 429 | CALDAC factory constant | Bipolar AI | MB88341 (8-bit) | 11 | 0 |
| 428 | CALDAC factory constant | Bipolar AI | MB88341 (8-bit) | 1 | 0 |
| 427 | CALDAC factory constant | Bipolar AI | MB88341 (8-bit) | 2 | 0 |
| 426 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 5 | 0 |
| 425 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 7 | 0 |
| 424 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 6 | 0 |
| 423 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 8 | 0 |

**Table 5-5.** PCI-6024E and PCI-6025E EEPROM Map (Continued)

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 422 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 10 | 0 |
| 421 | CALDAC factory constant | Bipolar AO | MB88341 (8-bit) | 9 | 0 |
| 420 | Factory calibration temperature | — | — | — | — |
| 371 | Start of the five user calibration sections | — | — | — | — |

[1]Board Codes LSB (MSB × 256) + LSB = Board Code

13—PCI-6024E (1 × 256) + 13 = 269
15—PCI-6025E (1 × 256) + 15 = 271

[2] Board Codes MSB:

01—PCI-6024E
01—PCI-6025E

**Table 5-6.** PCI-6052E EEPROM Map

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 511 | NI-DAQ Board Code LSB:17[1] | — | — | — | — |
| 474 | NI-DAQ Board Code MSB:1[1] | — | — | — | — |
| 510 | Revision | — | — | — | — |
| 509 | Sub-revision | — | — | — | — |
| 508 | Year of last factory calibration | — | — | — | — |
| 507 | Month of last factory calibration | — | — | — | — |
| 506 | Day of last factory calibration | — | — | — | — |
| 416 | Factory reference MSB | — | — | — | — |
| 415 | Factory reference LSB | — | — | — | — |
| 414 | CALDAC factory constant | AI pregain offset coarse | MB88341 | 0 | 0 |
| 413 | CALDAC factory constant | AI pregain offset fine | MB88341 | 8 | 0 |
| 412 | CALDAC factory constant | AI postgain offset coarse | MB88341 | 4 | 0 |
| 411 | CALDAC factory constant | AI postgain offset fine | MB88341 | 12 | 0 |
| 410 | CALDAC factory constant | AI gain coarse | MB88341 | 2 | 0 |
| 409 | CALDAC factory constant | AI gain fine | MB88341 | 10 | 0 |
| 408 | CALDAC factory constant | AI unipolar offset coarse | MB88341 | 14 | 0 |
| 407 | CALDAC factory constant | AI unipolar offset fine | MB88341 | 7 | 0 |

**Table 5-6.** PCI-6052E EEPROM Map (Continued)

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 406 | CALDAC factory constant | AO 0 bipolar linearity | MB88341 | 0 | 1 |
| 405 | CALDAC factory constant | AO 0 bipolar gain offset coarse | MB88341 | 8 | 1 |
| 404 | CALDAC factory constant | AO 0 bipolar gain fine | MB88341 | 4 | 1 |
| 403 | CALDAC factory constant | AO 0 bipolar offset | MB88341 | 12 | 1 |
| 402 | CALDAC factory constant | AO 1 bipolar linearity | MB88341 | 2 | 1 |
| 401 | CALDAC factory constant | AO 1 bipolar gain coarse | MB88341 | 10 | 1 |
| 400 | CALDAC factory constant | AO 1 bipolar gain fine | MB88341 | 6 | 1 |
| 399 | CALDAC factory constant | AO 1 bipolar offset | MB88341 | 14 | 1 |
| 398 | CALDAC factory constant | AO 0 unipolar linearity | MB88341 | 0 | 1 |
| 397 | CALDAC factory constant | AO 0 unipolar gain coarse | MB88341 | 8 | 1 |
| 396 | CALDAC factory constant | AO 0 unipolar gain fine | MB88341 | 4 | 1 |
| 395 | CALDAC factory constant | AO 0 unipolar offset | MB88341 | 12 | 1 |

**Table 5-6.** PCI-6052E EEPROM Map (Continued)

| EEPROM Address | Data Stored at EEPROM Address | Data Type | CALDAC Type | CALDAC Address | Ser DacLd Line |
|---|---|---|---|---|---|
| 394 | CALDAC factory constant | AO 1 unipolar linearity | MB88341 | 2 | 1 |
| 393 | CALDAC factory constant | AO 1 unipolar gain coarse | MB88341 | 10 | 1 |
| 392 | CALDAC factory constant | AO 1 unipolar gain fine | MB88341 | 6 | 1 |
| 391 | CALDAC factory constant | AO 1 unipolar offset | MB88341 | 14 | 1 |
| 390 | Factory calibration temp | — | — | — | 1 |
| 371 | Start of the four user calibration sections | — | — | — | 1 |
| [1]Board Codes | (MSB × 256) + LSB = Board Code<br>(1 × 256) + 17 = 267 | | | | |

## Calibration DACs

The calibration DACs are used to adjust the errors in the analog signal paths for errors such as offset and gain. Table 5-7 shows which type of DAC(s) each board uses for calibration.

**Table 5-7.** Type of CALDAC Used on Board

| Board | MB88341 (8-bit) | DAC8800 (8-bit) | DAC8043 (12-bit)[1] | AD8522 (12-bit)[2] |
|---|---|---|---|---|
| PCI-MIO-16E-1 | ✓ | — | — | — |
| PCI-MIO-16E-4 | ✓ | — | — | — |
| PCI-6071E | ✓ | — | — | — |
| PCI-MIO-16XE-50 | — | ✓ | ✓ | — |
| PCI-MIO-16XE-10 | — | ✓ | ✓ | ✓ |

**Table 5-7.** Type of CALDAC Used on Board (Continued)

| | | | | |
|---|---|---|---|---|
| PCI-6023E | ✓ | — | — | — |
| PCI-6024E | ✓ | — | — | — |
| PCI-6025E | ✓ | — | — | — |
| PCI-6031E | — | ✓ | ✓ | ✓ |
| PCI-6032E | — | ✓ | ✓ | ✓ |
| PCI-6033E | — | ✓ | ✓ | ✓ |
| PCI-6052E[3] | ✓ | — | — | ✓ |

[1] When writing to this CALDAC an address is not necessary.
[2] Used for analog triggering.
[3] This board uses two MB88341 CALDACs.

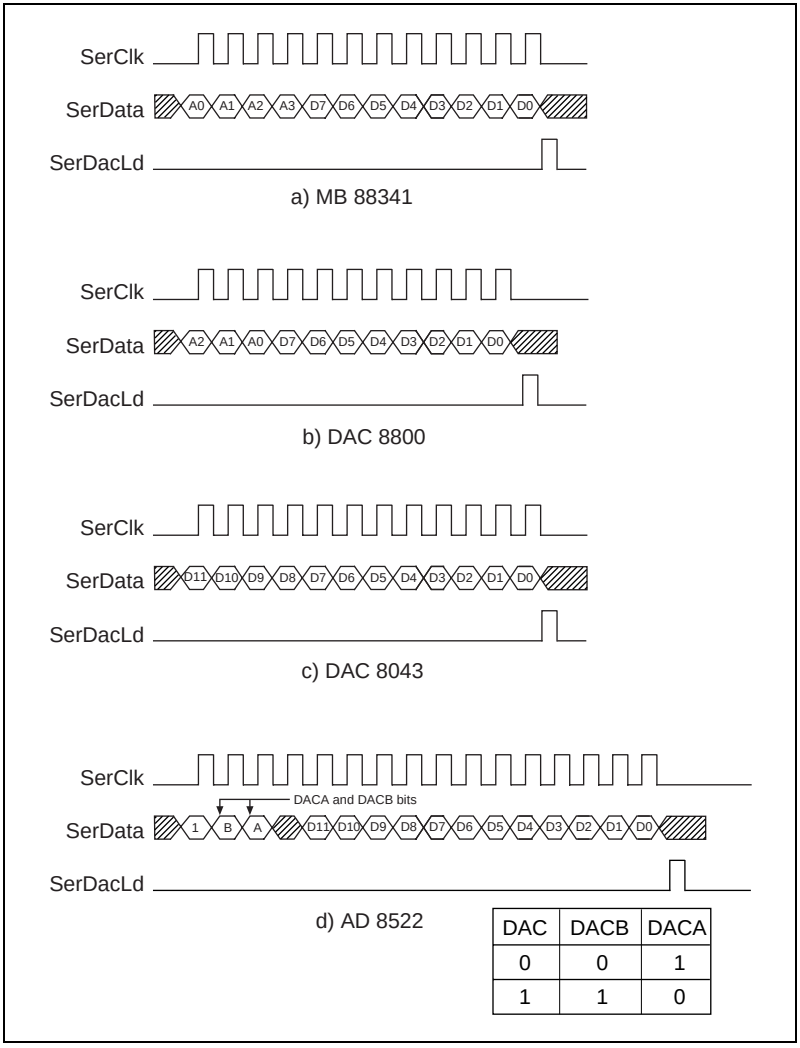
When calibrating an E-series board for either unipolar or bipolar analog output or input, make sure that all calibration constants for that particular mode are loaded. For example, if you want to calibrate the PCI-6031E for bipolar analog output, be sure to load the constants stored in EEPROM memory location 414, 415, 416 and 417 into the MB88341 CALDAC.

The Serial Command Register has five bits—SerClk (bit 0), SerData (bit 1), SerDacLd0 (bit 3), SerDacLd1 (bit 4), and SerDacLd2 (bit 5)—that are connected to the serial DAC clock, data in, load for the CALDAC.

The format for writing to all of the serial DACs is similar to the EEPROM. The basic write cycle consists of shifting an address/data pair into the DAC, then pulsing the appropriate SerDacLd pin. The timing diagram for the write cycle for each DAC is shown in Figure 5-2 (a, b, c, d).

The serial DAC interface is relatively slow compared to the PCI bus. In order to meet the timing specifications of the serial DAC, you must double write the clock bit. In other words, when setting the clock bit low, you must write 0 twice, and when setting it high, you must write 1 twice.

☞ **Note** ***Review the timing diagram and specifications very carefully before attempting to write code.***




| DAC | DACB | DACA |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

**Figure 5-2.** Calibration AC Write Timing

☞ **Note** ***Refer to the EEPROM Map table of your board for use with Figure 5-2.***

# NI-DAQ Calibration Function

The NI-DAQ function called `Calibrate_E_Series` can calibrate the analog input, analog output, and internal reference on the PCI E Series boards. Due to the complexity of the actual calibration algorithm, use `Calibrate_E_Series` to calibrate each section and store the results in the EEPROM. You can write a separate application using `Calibrate_E_Series`, which is run only when the board needs new calibration constants. Writing such an application allows the normal application to simply copy the calibration constants from the EEPROM and write them to the calibration DACs upon board initialization.

# A

# Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

## Electronic Services

### Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

### FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

## Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

## E-Mail Support (Currently USA Only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

# Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

| Country | Telephone | Fax |
|---|---|---|
| Australia | 03 9879 5166 | 03 9879 6277 |
| Austria | 0662 45 79 90 0 | 0662 45 79 90 19 |
| Belgium | 02 757 00 20 | 02 757 03 11 |
| Brazil | 011 288 3336 | 011 288 8528 |
| Canada (Ontario) | 905 785 0085 | 905 785 0086 |
| Canada (Québec) | 514 694 8521 | 514 694 4399 |
| Denmark | 45 76 26 00 | 45 76 26 02 |
| Finland | 09 725 725 11 | 09 725 725 55 |
| France | 01 48 14 24 24 | 01 48 14 24 14 |
| Germany | 089 741 31 30 | 089 714 60 35 |
| Hong Kong | 2645 3186 | 2686 8505 |
| Israel | 03 6120092 | 03 6120095 |
| Italy | 02 413091 | 02 41309215 |
| Japan | 03 5472 2970 | 03 5472 2977 |
| Korea | 02 596 7456 | 02 596 7455 |
| Mexico | 5 520 2635 | 5 520 3282 |
| Netherlands | 0348 433466 | 0348 430673 |
| Norway | 32 84 84 00 | 32 84 86 00 |
| Singapore | 2265886 | 2265887 |
| Spain | 91 640 0085 | 91 640 0533 |
| Sweden | 08 730 49 70 | 08 730 43 70 |
| Switzerland | 056 200 51 51 | 056 200 51 55 |
| Taiwan | 02 377 1200 | 02 737 4644 |
| United Kingdom | 01635 523545 | 01635 523154 |
| United States | 512 795 8248 | 512 794 5678 |

# Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax ( ___ ) _____Phone ( ___ ) _____

Computer brand_____ Model _____Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed_____

Hard disk capacity _____MB Brand_____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem: _____

_____

_____

_____

_____

# PCI E Series Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

## National Instruments Products

PCI-MIO E Series hardware revision _____

Interrupt level of PCI-MIO E Series device _____

Base memory address of PCI-MIO E Series device _____

Other National Instruments boards in system _____

Base memory address of other National Instruments boards _____

Interrupt level of other National Instruments boards _____

## Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Programming language _____

Programming language version _____

Other boards in system _____

Base memory address of other boards _____

Interrupt level of other boards _____

# Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

**Title:** *PCI E Series Register-Level Programmer Manual*

**Edition Date:** November 1998

**Part Number:** 341079B-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

E-Mail Address _____

Phone ( ___ ) _____ Fax ( ___ ) _____

**Mail to:** Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

**Fax to:** Technical Publications
National Instruments Corporation
512 794 5678

# Glossary

| Prefix | Meanings | Value |
|---|---|---|
| p- | pico | $10^{-12}$ |
| n- | nano- | $10^{-9}$ |
| µ- | micro- | $10^{-6}$ |
| m- | milli- | $10^{-3}$ |
| k- | kilo- | $10^{3}$ |
| M- | mega- | $10^{6}$ |
| G- | giga- | $10^{9}$ |

## Symbols

| | |
|---|---|
| * | inverted bit (negative logic) if after a bit name |
| Ω | ohms |

## A

| | |
|---|---|
| A | amperes |
| A/D | analog-to-digital |
| AC | alternating current |
| ADC | A/D converter |
| AIGND | analog input ground signal |
| AOGND | analog output ground signal |
| ASIC | application-specific integrated circuit |

## B

| | |
|---|---|
| Bank | bank select bit |
| BC | buffer counter |
| BipDac | bipolar DAC bit |

## C

| | |
|---|---|
| CALDAC | calibration DAC |
| Chan | channel select bit |
| ChanEnable | DMA channel enable bit |
| Channel | physical channel select bit |
| ChanType | channel type bit |
| CONVERT | convert signal |

## D

| | |
|---|---|
| D | data bit |
| D/A | digital-to-analog |
| DAC | D/A converter |
| DAC0OUT | analog channel 0 output signal |
| DAC1OUT | analog channel 1 output signal |
| DACSe | DAC select bit |
| DAQ | data acquisition |
| DAR | Device Address Register |
| DC | direct current |
| DitherEn | dither enable bit |

| | |
|---|---|
| DIV | divide by signal |
| DMA | direct memory access |
| DMATCA | DMA terminal count A bit |
| DmaTcAClr | DMA terminal count A clear bit |
| DMATCB | DMA terminal count B bit |
| DmaTcBClr | DMA terminal count B clear bit |
| DMATCC | DMA terminal count C bit |
| DmaTcCClr | DMA terminal count C clear bit |

## E

| | |
|---|---|
| EEPROM | electrically erasable programmable read-only memory |
| EEPromCS | EEPROM chip select bit |
| EXTREF | external reference signal |
| ExtRef | external reference for DAC bit |
| EXTSTROBE* | external strobe signal |
| EXTTRIG | external trigger signal |

## F

| | |
|---|---|
| FIFO | first-in-first-out |

## G

| | |
|---|---|
| Gain | channel gain select bit |
| GenTrig | general trigger bit |
| ghost | a conversion that is performed but the data is thrown away |
| GPCT0 | general-purpose counter timer 0 bit |

| | |
|---|---|
| GPCT1 | general-purpose counter timer 1 bit |
| GroundRef | ground reference bit |

## H

| | |
|---|---|
| hex | hexadecimal |
| Hz | hertz |

## I

| | |
|---|---|
| I/O | input/output |
| Input | analog input bit |
| Int/Ext Trig | internal/external analog trigger |
| IRQ | interrupt request signal |
| ISA | Industry Standard Architecture |
| ISR | interrupt service routine |

## L

| | |
|---|---|
| LASTCHANNEL | last channel bit |
| LKAR | Link Address Register |
| LSB | least significant bit |

## M

| | |
|---|---|
| m | meters |
| MAR | Memory Address Register |
| MB | megabytes of memory |

| | |
|---|---|
| MSB | most significant bit |
| MUX | multiplexer |

## N

| | |
|---|---|
| NRSE | nonreferenced single-ended input |

## O

| | |
|---|---|
| op-amp | operational amplifiers |
| OS | operating system |
| Output | analog output bit |

## P

| | |
|---|---|
| PFI0/Trig1 | PFI 0/Trigger 1 signal |
| PFI1/Trig2 | PFI 1/Trigger 2 signal |
| PGIA | Programmable Gain Instrumentation Amplifier |
| ppm | parts per million |
| PRETRIG | pretrigger signal |
| PROMOUT | EEPROM output data bit |

## R

| | |
|---|---|
| ReGlitch | reglitch DAC bit |
| RTD | resistance-temperature detector |
| RTSI | Real-Time System Integration bus |
| RTSI_BRD0 | RTSI board |

## S

| | |
|---|---|
| S | samples |
| s | second |
| SC | scan counter |
| SerClk | serial clock bit |
| SerDacLd | serial DAC load bit |
| SerData | serial data bit |
| SHIFTIN | shift in signal |
| SI | scan interval counter |
| SI2 | sample interval |
| START | start signal |
| STOP | stop signal |

## T

| | |
|---|---|
| TC | terminal count |
| TCIntEnable | DMATC interrupt enable bit |
| TCR | Transfer Count Register |
| Transfer | transfer type bit |
| TTL | transistor-transistor logic |

## U

| | |
|---|---|
| UC | update counter |
| UI | update interval |

| | |
|---|---|
| UI2 | update interval 2 |
| Unip/Bip | channel unipolar/bipolar bit |

## V

| | |
|---|---|
| V | volts |
| $V_{ref}$ | input voltage reference |

## X

| | |
|---|---|
| X | don’t care bits |

# Index

## A


ADC FIFO Clear Register
  description, 3-24
  register map, 3-3
ADC FIFO Data Register
  description, 3-8
  register map, 3-2
ADCs
  single-read timing, 2-11 to 2-12
  theory of operation, 2-10
AI AO Select Register
  description, 3-22
  register map, 3-2
AI_Arming function
  AMUX-64T examples
    sampling one channel, 4-28
    scanning eight channels, 4-31
  STC scanning examples, 4-14
    with DMA, 4-19
    with external start and stop trigger, 4-25
    with external start trigger and scan start pulses, 4-22
    with interrupts, 4-16
    single wire acquisition example, 4-26
AI_Board_Environmentalize function
  acquiring one sample from channel 0, 4-11
  AMUX-64T example, 4-29
AI_Board_Personalize function, 4-10
AI_End_of_Scan function, 4-11
AI_FIFO_Empty_St bit, 2-12
AIFREQ signal, 4-58
AI_Initialize_Configuration_Memory_Output function
  acquiring one sample from channel 0, 4-10
  AMUX-64T examples
    sampling one channel, 4-27
    scanning eight channels, 4-29
AI_Interrupt_Enable function, 4-16
AI_Reset_All function, 4-10
AI_Scan_Start function
  AMUX-64T examples
    sampling one channel, 4-27
    scanning eight channels, 4-30
  sampling from channel 0, 4-11
  STC scanning examples, 4-13
    with DMA, 4-18
    with external start and stop trigger control, 4-24
    with external start trigger and scan start, 4-21
    with interrupts, 4-15
    single wire acquisition, 4-26
AI_Start_The_Acquisition function
  acquiring one sample from channel 0, 4-12
  AMUX-64T examples
    sampling one channel, 4-28
    scanning eight channels, 4-31
  STC scanning examples, 4-14
    with DMA, 4-20
    with interrupts, 4-17
    with single wire acquisition, 4-26
AI_Trigger_Signals function
  acquiring one sample from channel 0, 4-11
  STC scanning
    with external start and stop trigger, 4-23
    with external start trigger and scan start, 4-21
AMUX-64T programming examples
  sampling one channel, 4-27 to 4-29
  scanning eight channels, 4-29 to 4-31
analog input circuitry
  block diagram, 2-8
  theory of operation, 2-8 to 2-11

analog input programming examples, 4-8 to 4-31
acquiring one sample from channel 0, 4-9 to 4-12
AMUX-64T
sampling one channel, 4-27 to 4-29
scanning eight channels, 4-29 to 4-31
functions for programming, 4-8
single wire acquisition, 4-25 to 4-26
STC scanning, 4-12 to 4-14
acquiring 20 scans with external start and stop trigger, 4-23 to 4-25
with DMA, 4-17 to 4-20
with external start trigger and scan start pulses, 4-21 to 4-22
with interrupts, 4-14 to 4-17
Analog Input Register Group
ADC FIFO Data Register, 3-8
Configuration Memory High Register, 3-11 to 3-15
Configuration Memory Low Register, 3-9 to 3-10
overview, 3-7
register map, 3-2
analog output circuitry
block diagram, 2-20
theory of operation, 2-21 to 2-22
analog output programming examples, 4-31 to 4-44
CPU write to DAC, 4-32 to 4-34
functions for programming, 4-32
waveform generation
with external UPDATE and external trigger, 4-41 to 4-43
with polled writes, 4-34 to 4-39
using interrupts, 4-43 to 4-44
using local buffer mode, 4-41 to 4-43
Analog Output Register Group
AO Configuration Register, 3-16 to 3-17
DAC FIFO Data Register, 3-18
DAC0 Direct Data Register, 3-19
DAC1 Direct Data Register, 3-20
overview, 3-15
register map, 3-2
analog output timing circuitry, 2-22 to 2-24
single-point output, 2-22 to 2-23
waveform generation, 2-23 to 2-24
analog triggering
programming considerations, 4-52 to 4-56
example, 4-54 to 4-56
trigger structure (figure), 4-54
theory of operation, 2-19 to 2-20
Analog_Trigger_Control function
analog triggering example, 4-55
purpose and use, 4-54
Analog_Trigger_Drive bit, 4-53
AO Configuration Register
description, 3-16 to 3-17
register map, 3-2
AO_Arming function
waveform generation examples
using interrupts, 4-44
using local buffer mode, 4-41
using polled writes, 4-38
AO_Board_Personalize function
CPU write to DAC, 4-33
waveform generation, 4-35
AO_Channels function, 4-37
AO_Counting function
waveform generation examples
using external UPDATE and trigger, 4-42
using local buffer mode, 4-39
using polled writes, 4-36
AO_Errors_To_Stop_On function, 4-38
AO_FIFO function
generating DMA requests, 4-58
waveform generation examples
using interrupts, 4-43
using local buffer mode, 4-41
using polled writes, 4-38

AOFREQ signal, 4-58
AO_LDAC_Source_And_Update_Mode function
  CPU write to DAC, 4-33
  waveform generation, 4-38
AO_Reset_All function
  CPU write to DAC, 4-33
  waveform generation, 4-35
AO_Start_The_Acquisition function
  waveform generation examples
    using interrupts, 4-44
    using local buffer mode, 4-41
    using polled writes, 4-39
AO_Triggering function
  waveform generation examples
    using local buffer mode, 4-41
    using polled writes, 4-35
AO_Updating function
  waveform generation examples
    using external UPDATE and trigger, 4-43
    using local buffer mode, 4-40
    using polled writes, 4-36

## B

Bank<1..0> bits, 3-12
BipDac bit, 3-17
bipolar output, 2-21
bitfields, 4-1
bits
  AI_FIFO_Empty_St, 2-12
  Analog_Trigger_Drive, 4-53
  Bank<1..0>, 3-12
  BipDac, 3-17
  Chan<3..0>, 3-12 to 3-15
  ChanType<2..0>, 3-11
  D<15..0>, 3-8, 3-18, 3-19, 3-20
  DACSel, 3-16
  DitherEn, 3-9 to 3-10
  DOTRIG0, 4-52
  EEPromCS, 3-4, 5-1
  ExtRef, 3-16 to 3-17
  Gain<2..0>, 3-10
  GenTrig, 3-9
  GPCT0<D..A>, 3-23
  GPCT1<D..A>, 3-23
  GroundRef, 3-16
  Input<D..A>, 3-22
  Int/Ext Trig, 3-5, 4-53
  LastChan, 3-9
  LASTCHANNEL, 2-10, 2-13
  Output<D..A>, 3-22
  PROMOUT, 3-6, 5-1
  ReGlitch, 3-17
  SerClk, 3-4, 5-1, 5-15
  SerDacLd0, 3-4, 5-15
  SerDacLd1, 3-4, 5-15
  SerDacLd2, 3-4, 5-15
  SerData, 3-4, 5-1, 5-15
  Unip/Bip, 3-10
buffered pulsewidth measurement example, 4-47 to 4-49
Buffered_Pulse_Width_Measurement function, 4-47
bulletin board support, A-1

## C

CALDACs. *See* calibration DACs.
Calibrate_E_Series function, 5-17
calibration
  circuitry, 2-11
  EEPROM
    calibration constant storage, 2-11, 5-1
    PCI-6023E map (table), 5-9
    PCI-6024E and PCI-6025E map (table), 5-10 to 5-11
    PCI-6052E map (table), 5-12 to 5-14

PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-MIO-6071E map (table), 5-3 to 5-4
PCI-MIO-16XE-10 and PCI-6031E, PCI-6032E, and PCI-6033E map, 5-7 to 5-8
PCI-MIO-16XE-50 map, 5-5 to 5-6
reading from, 5-1 to 5-2
writing to accidentally (note), 5-2
NI-DAQ calibration function, 5-17
calibration DACs
purpose and use, 5-14 to 5-15
type of DAC(s) used by each board (table), 5-14 to 5-15
write timing diagram, 5-16
writing to, 5-15
Chan<3..0> bits
calibration channel assignments (table), 3-12
channel assignments (table), 3-15
description, 3-12
differential channel assignments (table), 3-13
nonreferenced single-ended channel assignments (table), 3-13 to 3-14
referenced single-ended channel assignments (table), 3-14
channel number, analog input, 2-9
channel type field, analog input, 2-9
channels
calibration channel assignments (table), 3-12
channel assignments (table), 3-15
differential channel assignments (table), 3-13
nonreferenced single-ended channel assignments (table), 3-13 to 3-14
referenced single-ended channel assignments (table), 3-14
valid channel types (table), 3-11
ChanType<2..0> bit, 3-11
Clear_FIFO function, 4-10, 4-12
configuration memory
definition, 2-8
multirate scanning with ghost (table), 2-18
Configuration Memory Clear Register
description, 3-24
register map, 3-3
Configuration Memory High Register
description, 3-11 to 3-15
register map, 3-2
Configuration Memory Low Register, 3-9 to 3-10
description, 3-9 to 3-10
register map, 3-2
Configure_Board function
acquiring one sample from channel 0, 4-9
analog triggering example, 4-55
continuous pulse train generation example, 4-49 to 4-52
Cont_Pulse_Train_Generation function, 4-50
CONVERT* signal
data acquisition sequence timing, 2-12 to 2-14
initiating conversions, 2-11
RTSI trigger lines programming considerations, 4-52
Convert_Signal function
AMUX-64T examples
sampling one channel, 4-28
scanning eight channels, 4-31
STC scanning examples, 4-13
with DMA, 4-18
with external start and stop trigger, 4-24
with external start trigger and scan start, 4-22
with interrupts, 4-16
single wire acquisition, 4-26

counter/timer, programming. *See* general-purpose counter/timer.
customer communication, *xv,* A-1 to A-2

## D

D<15..0> bits, 3-8
DAC FIFO Data Register, 3-18
DAC0 Direct Data Register, 3-19
DAC1 Direct Data Register, 3-20
DAC FIFO Clear Register
description, 3-24
register map, 3-3
DAC FIFO Data Register
description, 3-18
register map, 3-2
DAC0 Direct Data Register, 3-19
description, 3-19
register map, 3-2
DAC1 Direct Data Register
description, 3-20
register map, 3-2
DACs
analog output circuitry, 2-21 to 2-22
analog triggering, 4-53
DACSel bit, 3-16
DAQ-STC programming examples. *See* programming examples.
DAQ-STC Register Group
overview, 3-24
register map, 3-2
DAQ-STC system timing controller, *xiii*
counter diagram, 2-24
programming. *See* programming examples.
data acquisition timing circuitry, 2-11 to 2-18
ADC timing (figure), 2-12
block diagram, 2-8
data acquisition sequence timing, 2-12 to 2-18
multirate scanning with ghost, 2-16 to 2-18
advantages (figure), 2-17
analog input configuration memory (table), 2-18
occurrences of conversion on channel 1 (figure), 2-17
successive scans (figure), 2-17
multirate scanning without ghost, 2-14 to 2-16
scanning three channels with 4:2:1 sampling rate (figure), 2-16
scanning two channels (figure), 2-15
1:*x* sampling rate, 2-15
3:1:1 sampling rate, 2-16
single-read timing, 2-11 to 2-12
timing of scan (figure), 2-14
differential channel assignments (table), 3-13
digital I/O circuitry, 2-24
digital I/O programming examples
performing digital I/O, 4-7 to 4-8
windowed registers, 4-7
dither circuitry, 2-10
DitherEn bit, 3-9 to 3-10
DIV counter, 2-12
DMA Control Register Group
AI AO Select Register, 3-22
G0 G1 Select Register, 3-23
overview, 3-21
register map, 3-2
DMA programming, 4-57 to 4-59
analog input examples, 4-17 to 4-20
STC scanning with DMA, 4-17 to 4-20
generating DMA requests, 4-57 to 4-58
Link Chaining Mode for DMA transfer, 4-58 to 4-59
programming MITE for different DMA transfers, 4-20 to 4-21
structure (figure), 4-57

documentation
about this manual, *xiii*
conventions used in manual, *xiv-xv7*
organization of manual, *xiv*
related documentation, *xv*
DOTRIG0 bit, 4-52

## E

EEPROM
calibration constant storage, 2-11, 5-1
PCI-6023E map (table), 5-9
PCI-6024E and PCI-6025E map (table), 5-10 to 5-11
PCI-6052E map (table), 5-12 to 5-14
PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-MIO-6071E map (table), 5-3 to 5-4
PCI-MIO-16XE-10 and PCI-6031E, PCI-6032E, and PCI-6033E map, 5-7 to 5-8
PCI-MIO-16XE-50 map, 5-5 to 5-6
reading from, 5-1 to 5-2
writing to accidentally (note), 5-2
EEPromCS bit, 3-4, 5-1
electronic support services, A-1 to A-2
e-mail support, A-2
ExtRef bit, 3-16 to 3-17
EXTSTROBE* signal, digital I/O circuitry, 2-24

## F

fax and telephone support numbers, A-2
Fax-on-Demand support, A-2
FIFO
analog output, 2-22
configuration memory, 2-8
overflow, 2-11
theory of operation, 2-10 to 2-11
waveform generation, 2-23 to 2-24
FIFO Strobe Register Group
ADC FIFO Clear Register, 3-24
Configuration Memory Clear Register, 3-24
DAC FIFO Clear Register, 3-24
register map, 3-3
FIFO_Request_Selection function, 4-58
files for example programs, 4-6
FTP support, A-1

## G

G0 G1 Select Register
description, 3-23
register map, 3-2
G0_Arm function
buffered pulsewidth measurement, 4-48
continuous pulse train generation example, 4-51
gated event counting example, 4-47
G0_Out_Enable function, 4-51
G0_Reset_All function, 4-45
G0_Seamless_Pulse_Train function, 4-51
G0_Watch function, 4-47
Gain<2..0> bits, 3-10
GATE signal, timing I/O circuitry, 2-25
gated event counting example, 4-45 to 4-47
general-purpose counter/timer
programming considerations, 4-45
programming examples, 4-45 to 4-52
buffered pulsewidth measurement, 4-47 to 4-49
continuous pulse train generation, 4-49 to 4-52
gated event counting, 4-45 to 4-47
GenTrig bit, 3-9
GENTRIG0 signal, 4-52
getvect() function, 4-14
ghost channel
definition, 2-9

multirate scanning
with ghost, 2-16 to 2-18
without ghost, 2-14 to 2-16
glitching, 2-22
GPCT0<D..A> bits, 3-23
GPCT1<D..A> bits, 3-23
GroundRef bit, 3-16

## I

initializing PCI
for IBM compatible systems, 4-2
for Macintosh computers, 4-4
Input<D..A> bits, 3-22
interrupt programming, 4-56
interrupt sharing, 4-56 to 4-57
Interrupt_Service_Routine function, 4-16
Int/Ext Trig bit, 3-5, 4-53

## K

Kick_Start_FIFO function
waveform generation examples
using interrupts, 4-44
using polled writes, 4-38

## L

LastChan bit, 3-9
LASTCHANNEL bit, 2-10, 2-13
Link Chaining Mode for DMA transfer, 4-58 to 4-59

## M

manual. *See* documentation.
Misc Command Register
description, 3-5
register map, 3-2
Misc Register Group
Misc Command Register, 3-5
overview, 3-3
register map, 3-2
Serial Command Register, 3-4
Status Register, 3-6
MITE ASIC
initializing PCI, 4-2
Link Chaining Mode for DMA transfer, 4-58 to 4-59
programming for different DMA transfers, 4-20 to 4-21
re-mapping PCI E Series board, 4-3
MITE_DMAarm function, 4-17, 4-20
MITE_DMAdisarm function, 4-17, 4-20
MITE_DMAgettransfsRemaining function, 4-20
MITE_DMAProgram function, 4-17, 4-19
MSC_Clock_Configure function
acquiring one sample from channel 0, 4-9
continuous pulse train generation, 4-50
waveform generation, 4-35
MSC_IO_Pin_Configure function, 4-45
MSC_IRQ_Configure function, 4-56
MSC_RTSI_Pin_Configure function, 4-52
multirate scanning with ghost, 2-16 to 2-18
advantages (figure), 2-17
analog input configuration memory (table), 2-18
occurrences of conversion on channel 1 (figure), 2-17
successive scans (figure), 2-17
multirate scanning without ghost, 2-14 to 2-16
scanning three channels with 4:2:1 sampling rate (figure), 2-16
scanning two channels (figure), 2-15
1:*x* sampling rate, 2-15
3:1:1 sampling rate, 2-16

## N

nonreferenced single-ended channel assignments (table), 3-13 to 3-14

Number_of_Scans function
AMUX-64T examples
sampling one channel, 4-27
scanning eight channels, 4-30
analog triggering example, 4-55
STC scanning examples, 4-12
with DMA, 4-18
with external start and stop trigger, 4-23
with external start trigger and scan start, 4-21
with interrupts, 4-15
single wire acquisition, 4-25

## O

operation of PCI E Series boards. *See* theory of operation.
OUT signal, timing I/O circuitry, 2-25
Output<D..A> bits, 3-22

## P

PCI E Series boards. *See also* theory of operation.
block diagrams
PCI-6023E, PCI-6024E, and PCI-6025E, 2-3
PCI-6032E and PCI-6033E, 2-4
PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-6071E, 2-1
PCI-MIO-16XE-10, PCI-MIO-6052E, and PCI-6031E, 2-2
PCI-MIO-16XE-50, 2-5
characteristics, 1-1 to 1-2
list of boards, *xiii*
PCI interface circuitry, 2-6 to 2-7
block diagram, 2-7
description, 2-6
PCI local bus programming considerations, 4-1 to 4-4
PCI initialization
for IBM compatible systems, 4-2
for Macintosh computers, 4-4
re-mapping PCI E Series board, 4-3 to 4-4
PFIO/TRIG1 signal, 4-53
PGIA (programmable gain instrumentation amplifier)
analog trigger programming considerations, 4-53
gain selection with Gain<2..0> bits, 3-10
gain set *versus* board (table), 2-9 to 2-10
theory of operation, 2-9
posttrigger acquisition, 2-18 to 2-19
pretrigger acquisition, 2-18 to 2-19
programmable gain instrumentation amplifier (PGIA). *See* PGIA (programmable gain instrumentation amplifier).
programming
analog triggering, 4-52 to 4-56
DMA programming, 4-57 to 4-59
DMA structure (figure), 4-57
Link Chaining Mode for DMA transfer, 4-58 to 4-59
examples. *See* programming examples.
general-purpose counter/timer, 4-45 to 4-52
interrupt programming, 4-56
interrupt sharing, 4-56 to 4-57
PCI local bus, 4-1 to 4-4
PCI initialization
for IBM compatible systems, 4-2
for Macintosh computers, 4-4
re-mapping PCI E Series board, 4-3 to 4-4
RTSI trigger lines considerations, 4-52
windowing registers, 4-5

programming examples
analog input, 4-8 to 4-31
acquiring one sample from channel 0, 4-9 to 4-12
functions for programming, 4-8
sampling one channel on AMUX-64T, 4-27 to 4-29
scanning eight channels on AMUX-64T, 4-29 to 4-31
single wire acquisition, 4-25 to 4-26
STC scanning, 4-12 to 4-14
with DMA, 4-17 to 4-20
with external start and stop trigger, 4-23 to 4-25
with external start trigger and scan start pulses, 4-21 to 4-22
with interrupts, 4-14 to 4-17
analog output, 4-31 to 4-44
CPU write to DAC, 4-32 to 4-34
functions for programming, 4-32
waveform generation
with external UPDATE and external trigger, 4-41 to 4-43
using interrupts, 4-43 to 4-44
using local buffer mode, 4-41 to 4-43
digital I/O, 4-7 to 4-8
files on companion disk, 4-5 to 4-6
general purpose counter/timer, 4-45 to 4-52
buffered pulsewidth measurement, 4-47 to 4-49
continuous pulse train generation, 4-49 to 4-52
gated event counting, 4-45 to 4-47
PROMOUT bit, 3-6, 5-1
pulse train generation example, 4-49 to 4-52
pulsewidth measurement example, 4-47 to 4-49
Pulse_Width_Measurement_ISR function, 4-48

## R

referenced single-ended channel assignments (table), 3-14
registers
Analog Input Register Group
ADC FIFO Data Register, 3-8
Configuration Memory High Register, 3-11 to 3-15
Configuration Memory Low Register, 3-9 to 3-10
overview, 3-7
Analog Output Register Group
AO Configuration Register, 3-16 to 3-17
DAC FIFO Data Register, 3-18
DAC0 Direct Data Register, 3-19
DAC1 Direct Data Register, 3-20
overview, 3-15
DAQ-STC Register Group, 3-24
DMA Control Register Group
AI AO Select Register, 3-22
G0 G1 Select Register, 3-23
overview, 3-21
FIFO Strobe Register Group
ADC FIFO Clear Register, 3-24
Configuration Memory Clear Register, 3-24
DAC FIFO Clear Register, 3-24
Misc Register Group
Misc Command Register, 3-5
overview, 3-3
Serial Command Register, 3-4
Status Register, 3-6
register maps, 3-2
sizes, 3-3
windowed registers
programming considerations, 4-5
register map, 3-2
ReGlitch bit, 3-17
reglitch circuitry, 2-22

re-mapping PCI E Series board, 4-3 to 4-4
RTSI bus interface circuitry
    block diagram, 2-26
    theory of operation, 2-25 to 2-26
RTSI trigger lines, programming considerations, 4-52
RTSI_BRD0 signal, 4-52

## S

sample interval (SI2), 2-12
scan counter (SC), 2-12 to 2-13
scan interval (SI), 2-12
SCAN sequence
    definition, 2-12
    starting, 2-13
scanning, multirate. *See* multirate scanning.
SerClk bit
    connection to EEPROM clock, 5-1
    connection to serial DAC clock, 5-15
    description, 3-4
SerDacLd0 bit
    connection to serial DAC clock, 5-15
    description, 3-4
SerDacLd1 bit
    connection to serial DAC clock, 5-15
    description, 3-4
SerDacLd2 bit
    connection to serial DAC clock, 5-15
    description, 3-4
SerData bit
    connection to EEPROM clock, 5-1
    connection to serial DAC clock, 5-15
    description, 3-4
Serial Command Register
    description, 3-4
    register map, 3-2
Setup_Mite function
    analog input examples
        acquiring one sample from channel 0, 4-9
    digital I/O examples, 4-7
    general-purpose counter/timer examples
        gated event counting, 4-45
    initializing PCI
        IBM compatible systems, 4-2
        Macintosh computers, 4-4
    re-mapping PCI E Series board, 4-3
setvect() function, 4-14
SHIFTIN* signal, ADC timing, 2-12
Simple_Gated_Count function, 4-45
single-point output, analog output timing circuitry, 2-22 to 2-23
single-read timing, data acquisition timing circuitry, 2-11 to 2-12
SOURCE signal, timing I/O circuitry, 2-25
START signal, 2-13
START1 signal, 2-18
START2 signal, 2-18
Status Register
    description, 3-6
    register map, 3-2
STOP signal, 2-13
support files for example programs, 4-6

## T

technical support, A-1 to A-2
telephone and fax support numbers, A-2
theory of operation
    analog input circuitry, 2-8 to 2-11
    analog output circuitry, 2-21 to 2-22
    analog output timing circuitry, 2-22 to 2-24
        single-point output, 2-22 to 2-23
        waveform generation, 2-23 to 2-24
    analog triggering, 2-19 to 2-20
    block diagrams
        PCI-6023E, PCI-6024E, and PCI-6025E, 2-3
        PCI-6032E and PCI-6033E, 2-4

PCI-MIO-16E-1, PCI-MIO-16E-4, and PCI-6071E, 2-1
PCI-MIO-16XE-10, PCI-MIO-6052E, and PCI-6031E, 2-2
PCI-MIO-16XE-50, 2-5
components of PCI E Series boards, 2-5
data acquisition timing circuitry, 2-11 to 2-18
ADC timing (figure), 2-12
block diagram, 2-8
data acquisition sequence timing, 2-12 to 2-18
multirate scanning with ghost, 2-16 to 2-18
multirate scanning without ghost, 2-14 to 2-16
single-read timing, 2-11 to 2-12
timing of scan (figure), 2-14
digital I/O circuitry, 2-24
functional overview, 2-1 to 2-5
PCI interface circuitry, 2-6 to 2-7
posttrigger and pretrigger acquisition, 2-18 to 2-19
RTSI bus interface circuitry, 2-25 to 2-26
timing I/O circuitry, 2-24 to 2-25
timing circuitry
analog output, 2-22 to 2-24
data acquisition, 2-11 to 2-18
ADC timing (figure), 2-12
block diagram, 2-8
data acquisition sequence timing, 2-12 to 2-18
multirate scanning with ghost, 2-16 to 2-18
multirate scanning without ghost, 2-14 to 2-16
single-read timing, 2-11 to 2-12
timing of scan (figure), 2-14
timing I/O circuitry
DAQ-STC counter diagram, 2-24
theory of operation, 2-24 to 2-25
trigger lines, RTSI, 4-52
triggering
analog triggering
programming considerations, 4-52 to 4-56
theory of operation, 2-19 to 2-20
posttrigger and pretrigger acquisition, 2-18 to 2-19

## U

Unip/Bip bit, 3-10
unipolar output, 2-21
update interval counter (UI), 2-23
UPDOWN signal, timing I/O circuitry, 2-25

## W

waveform generation
programming examples
with external UPDATE and external trigger, 4-41 to 4-43
with polled writes, 4-34 to 4-39
using interrupts, 4-43 to 4-44
using local buffer mode, 4-41 to 4-43
theory of, 2-23 to 2-24
windowed registers
digital I/O programming example, 4-7
programming considerations, 4-5
register map, 3-2

**Artisan Technology Group** is an independent supplier of quality pre-owned equipment

## Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

## We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

## Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP