DAQ

DAQCard E Series Register-Level Programmer Manual

Multifunction I/O Boards for the PCMCIA Bus

NATIONAL INSTRUMENTS™

November 1998 Edition
Part Number 341080A-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

Important Information

Warranty

The DAQCard 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

Trademarks

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

Chapter 1
General Description

Chapter 2
Theory of Operation

Chapter 3
Register Map and Descriptions

Chapter 4
Programming

Chapter 5
Calibration

Appendix A
Customer Communication

Glossary

Index

Figures

Tables

About This Manual

Thank you for buying a National Instruments DAQCard E Series card. The DAQCard E Series cards are multifunction analog, digital, and timing I/O cards for computers equipped with Type II PCMCIA slots. This family of cards features 12-bit and 16-bit ADCs with eight lines of TTL-compatible digital I/O, and two 24-bit counter/timers for timing I/O.

The DAQCard E Series cards 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.

The DAQCard E Series cards 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.

This manual is intended for programming at the register level. Even if you are an experienced register-level programmer, consider using NI-DAQ or other National Instruments software to program your DAQCard E Series board. If NI-DAQ does not support your operating system or you have other reasons to write register-level programs continue reading this manual.

Organization of This Manual

The *DAQCard E Series RLPM* is organized as follows:

- Chapter 1, *General Description*, describes the general characteristics and features of the DAQCard E Series boards.
- Chapter 2, *Theory of Operation*, contains a functional overview of the DAQCard Series boards and explains the operation of each functional unit making up the DAQCard Series boards.
- Chapter 3, *Register Map and Descriptions*, describes in detail the address and function of each of the E Series control and status registers.
- Chapter 4, *Programming*, contains programming instructions for operating the circuitry on the E Series boards.

- Chapter 5, *Calibration*, explains how to calibrate the analog input and output sections of the E Series boards by reading calibration constants from the EEPROM and writing them to the calibration DACs.
- Appendix A, *Customer Communication*, contains forms for you to complete to help you communicate with National Instruments about our products.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including acronyms, abbreviations, metric prefixes, mnemonics, and symbols.
- The *Index* alphabetically lists topics covered in this manual, including the page where you can find the topic.

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, port, or signal name (for example, ACH<0..7> stands for ACH0 through ACH7).
☞	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
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 NI-DAQ 6.*x*.
`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.

Related Documentation

You may find the following National Instruments documents helpful for programming interrupts:

- *Programming Interrupts for Data Acquisition on 80x86-Based Computers, Application Note 010*

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

- *DAQCard E Series User Manual*
- *DAQ-STC Technical Reference 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 and features of the DAQCard E Series boards.

General Characteristics

Thank you for buying a National Instruments DAQCard E Series card. The DAQCard E Series cards are multifunction analog, digital, and timing I/O cards for computers equipped with Type II PCMCIA slots. This family of cards features 12-bit and 16-bit ADCs with eight lines of TTL-compatible digital I/O, and two 24-bit counter/timers for timing I/O.

The DAQCard E Series cards 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.

The DAQCard E Series cards 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.

This manual is intended for programming at the register level. Even if you are an experienced register-level programmer, consider using NI-DAQ or other National Instruments application software to program the DAQCard E Series. If NI-DAQ does not support your operating system or you have other reasons to write your own register-level programs, continue reading this manual.

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

Table 1-1 lists the boards in the DAQCard E Series along with their distinguishing features.

Table 1-1. DAQCard E Series Boards and Features

Board	Features
DAQCard-AI-16E-4	16 single-ended, 16 pseudo differential, or 8 differential 12 bit analog inputs, 250 kS/S (single channel), 250 kS/S (scanning), eight digital I/O, analog and digital triggering
DAQCard-AI-16XE-50	16 single-ended or 8 differential 16 bit analog inputs, 200 kS/s (single channel), 20 kS/S (scanning)

2

Theory of Operation

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

Functional Overview

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

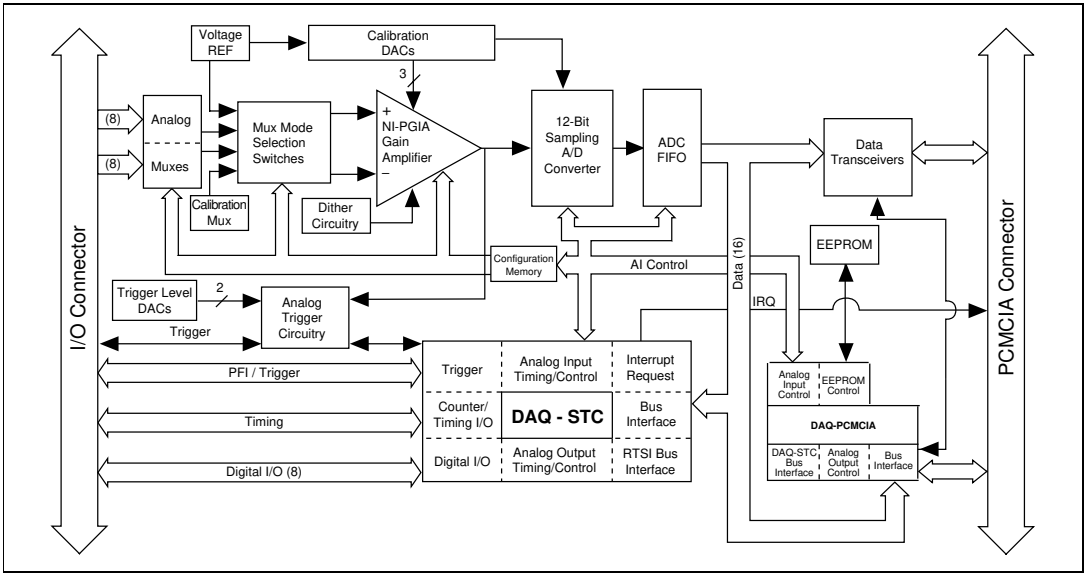

Figure 2-1. DAQCard-AI-16E-4 Block Diagram

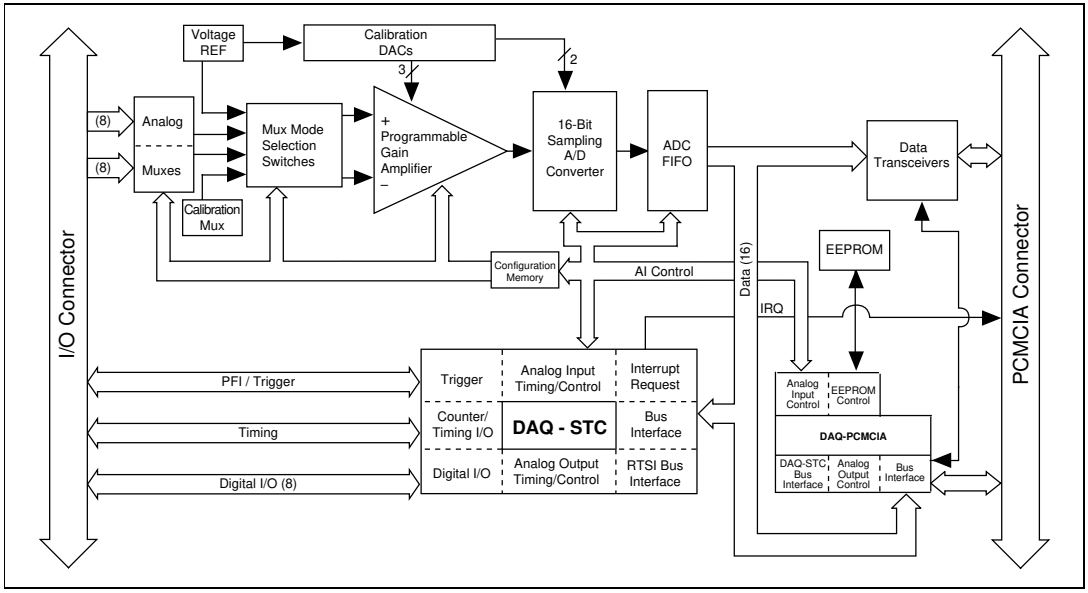

Figure 2-2. DAQCard-AI-16XE-50 Block Diagram

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

- PCMCIA bus interface circuitry
- Analog input circuitry
- Analog trigger circuitry
- Digital I/O circuitry
- Timing I/O circuitry (DAQ-STC)

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*.

PCMCIA Bus Interface Circuitry

The DAQCard E Series boards are all 16-bit PCMCIA interface cards. The PCMCIA bus features a 16-bit address bus, a 16-bit data bus, one interrupt line, and several control and support signals. Figure 2-3 shows the functional blocks making up the DAQCard E Series PCMCIA bus interface circuitry.

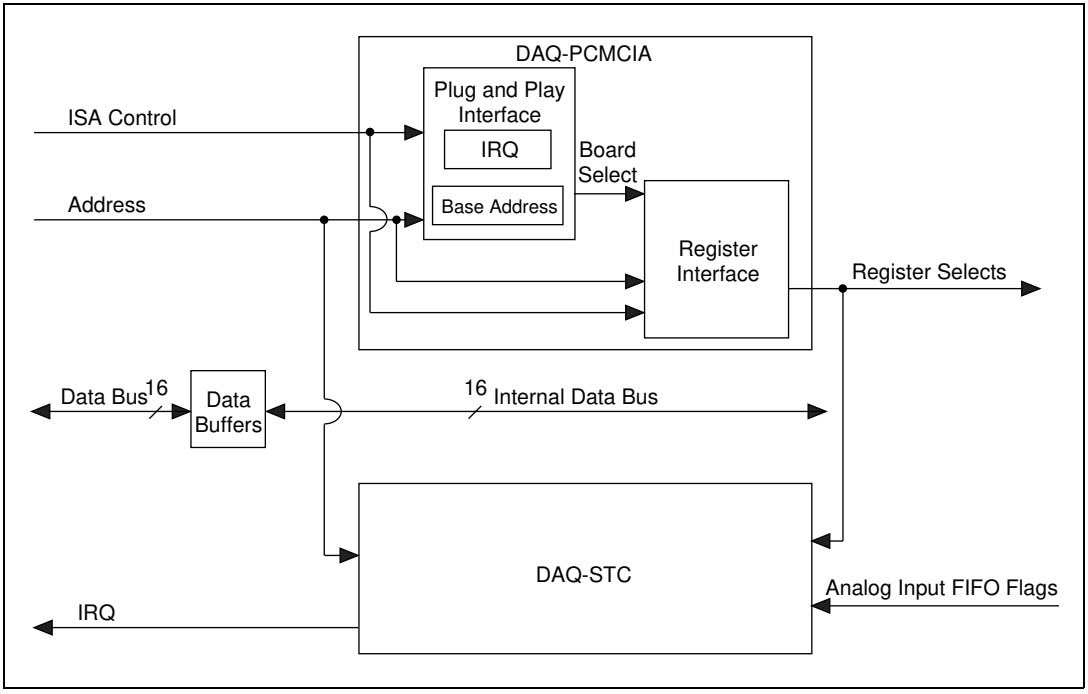

Figure 2-3. PCMCIA Bus Interface Circuitry Block Diagram

The DAQ-STC can generate interrupts from over 20 sources and can route these interrupts to one interrupt line on the PCMCIA bus interface. The DAQ STC will always route the interrupt over the IRQOUT0 line. The actual interrupt level that is used depends on the PC Card host adapter on the computer and which IRQ level it has selected.

Analog Input and Timing Circuitry

The DAQCard E Series boards have 16 analog input channels and a timing core within the DAQ-STC that is dedicated to analog input operation. Figure 2-4 shows a general block diagram for the analog input circuitry.

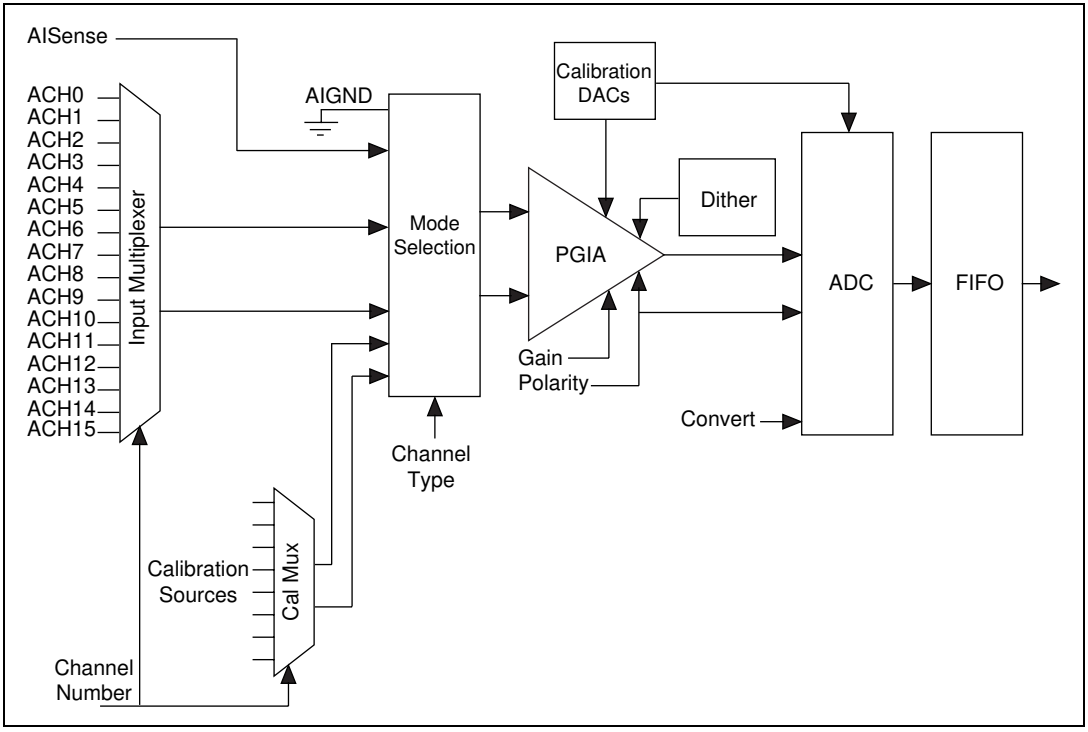

Figure 2-4. Analog Input and Data Acquisition Circuitry Block Diagram

Analog Input Circuitry

The general model for analog input on the DAQCard 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 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 DAQCard 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 of 0.5, 1, 2, 5, 10, 20, 50, and 100, for the DAQCard-AI-16E-4. On the DAQCard-AI-16XE-50, the PGIA supports gains of 1, 2, 10, and 100.

The dither circuitry adds approximately 0.5 LSB rms of white Gaussian noise to the signal to be 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 DAQCard-AI-16XE-50.

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 DAQCard E Series boards use sampling, successive approximation ADCs with 12 or 16 bits of resolution with maximum conversion rates between 5 µs and 2 µs. 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 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 1k words deep on the DAQCard-AI-16E-4 and DAQCard-AI-16XE-50. 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 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 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. You should never write to the entire lower half of the EEPROM because it is reserved.

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 low 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 E Series register set. Any one of these operations will generate the timing shown in Figure 2-5.

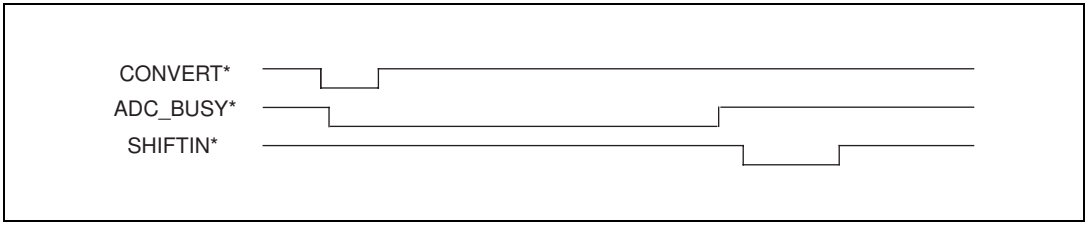

Figure 2-5. ADC Timing

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 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 E Series, 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 terminal count 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 (50 scans). Figure 2-6 shows the timing for each scan in Example 1.

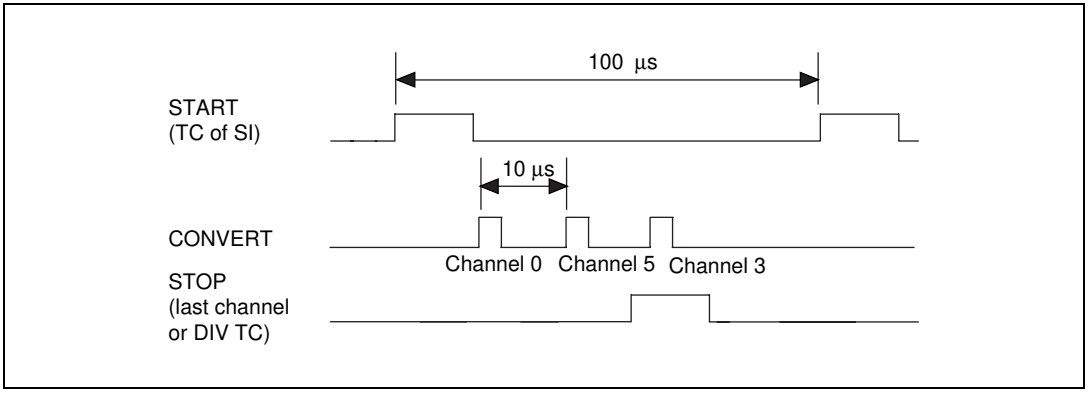

Figure 2-6. Timing of Scan in Example 1

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 2-7 shows the timing sequence for two scans.

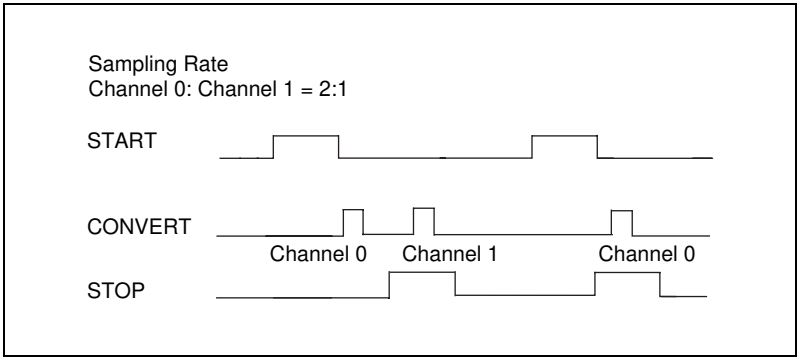

Figure 2-7. Multirate Scanning of Two Channels

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 2-8, 2-9, and 2-10 show timing sequences for different ratios. In these figures, the numbers above the CONVERT pulses indicate the channels sampled in that conversion.

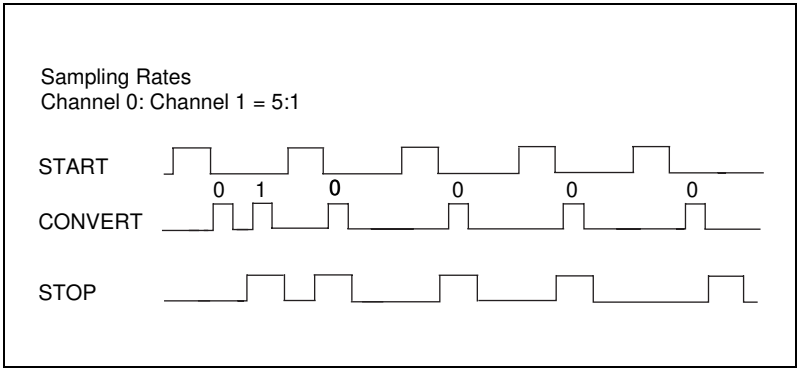

Figure 2-8. Multirate Scanning of Two Channels with 1:*x* Sampling Rate

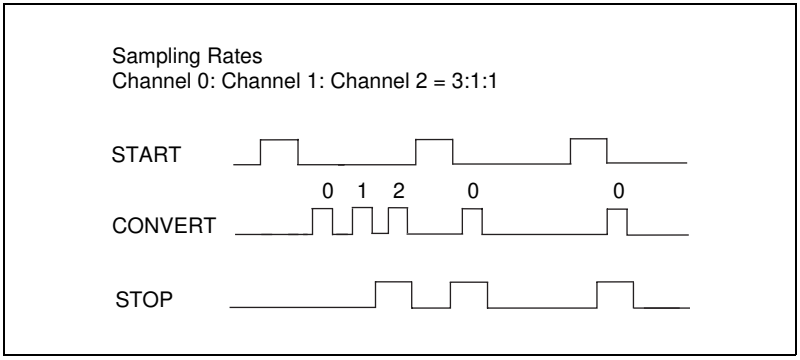

Figure 2-9. Multirate Scanning of Two Channels with 3:1:1 Sampling Rate

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

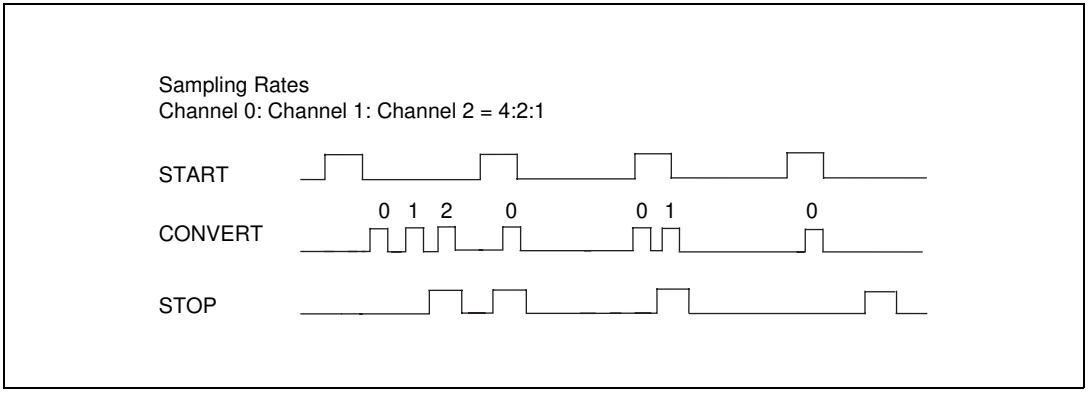

Figure 2-10. Multirate Scanning of Three Channels with 4:2:1 Sampling Rate

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 2-11 and 2-13 illustrate the advantages of using the ghost feature. Figure 2-9 shows example 3 timing, and Figure 2-13 shows the same example using ghost.

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

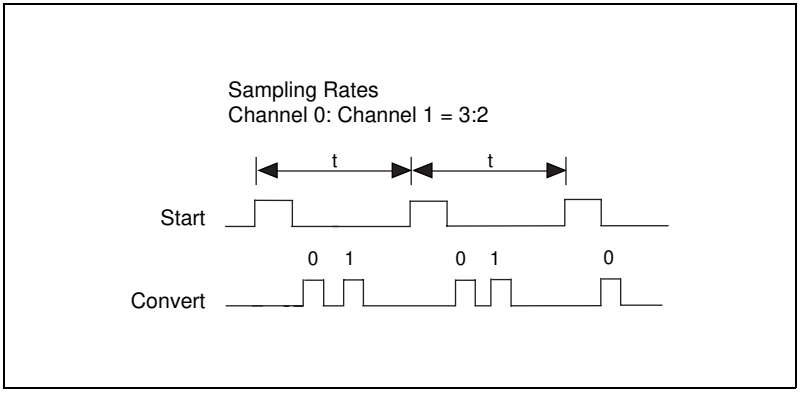

Figure 2-11. Multirate Scanning without Ghost

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 2-12 shows the relative occurrences of convert pulses in Figure 2-11.

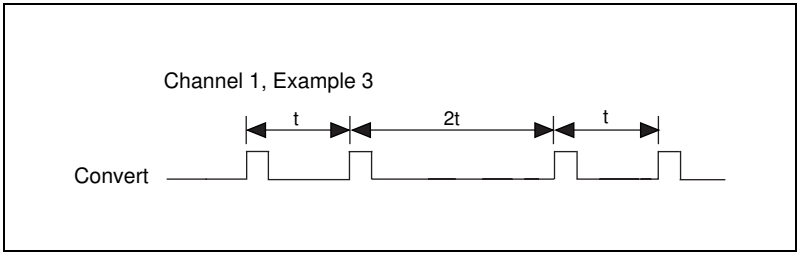

Figure 2-12. Occurrences of Conversion on Channel 1 in Example 3.

To rectify the problem, use ghost as illustrated in Figure 2-13.

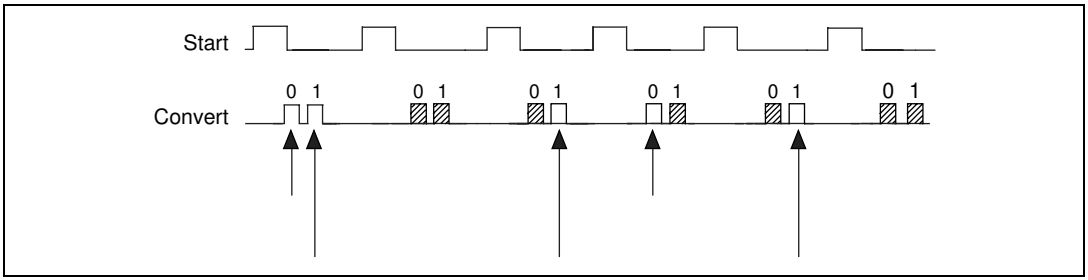

Figure 2-13. Successive Scans Using Ghost

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 2-1 shows what the configuration memory would look like.

Table 2-1. 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

The symbol — indicates that ghost or last channel is absent.

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

The DAQCard-AI-16E-4 has 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. An 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. These thresholds are within ± 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*.

Digital I/O Circuitry

The DAQCard 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 DAQCard 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 2-14.

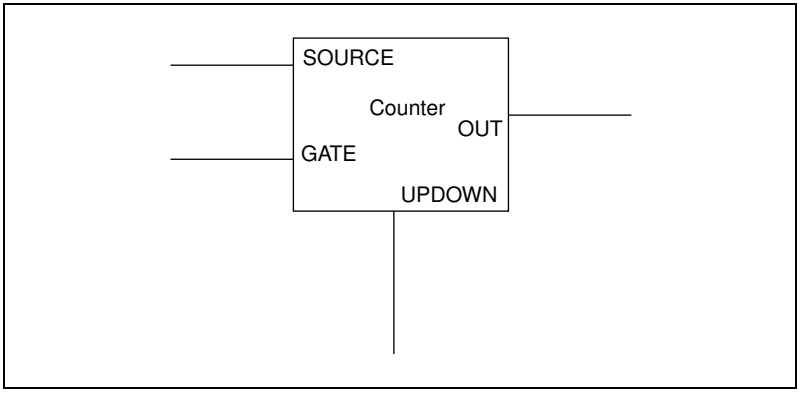

Figure 2-14. DAQ-STC Counter Diagram

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 E Series 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 or 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*.

3

Register Map and Descriptions

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

If you plan to use a programming software package such as NI-DAQ for DOS/Windows or Lab Windows with your E Series board, you need not read this chapter. However, you will gain added insight into your E Series board by reading this chapter.

Register Map

Table 3-1 shows the register map for the DAQCard 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 I/O base address of the DAQCard 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. You can access remaining registers in the DAQ-STC 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. 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
DAQ-STC Register Group			Read-and-write	16-bit
Window Address	0	0	Read-and-write	16-bit
Window Data	2	2	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. E Series Windowed Register Map

Register Name	Word Offset		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

Register Sizes

Two different transfer sizes for read-and-write operations are available on the PC—byte (8-bit) and word (16-bit). Table 3-1 shows the size of each 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. Pay particular attention to the register sizes; they are very important.

Register Descriptions

This section discusses each of the DAQCard 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 a register is written, these bits should be set 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 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	Reserved	SerDacLd1	SerDacLd0	EEPromCS	SerData	SerClk

Bit	Name	Description
7–5	Reserved	Reserved—Always write 0 to these bits.
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 (*DAQCard-AI-16XE-50 only*).
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 device. In order to write to these devices, this bit should be set first to 0 and then to 1. The data in the SerData bit will be written to the devices on the low-to-high transition of the serial clock.

Misc Command Register

The Misc Command Register contains one bit that controls the 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 PGIA 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—Always write 0 to these 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 DAQCard E Series ADC data FIFO to the PC. 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 will 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 will 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	Reserved	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.
14–12, 11–10, 7–3	Reserved	Reserved—Always write 0 to these bits.
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 DAQCard-AI-16XE-50 and should be set to 1. Dither cannot be disabled on the DAQCard-AI-16XE-50.
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 following gains can be selected on the E Series:

Table 3-3. PGIA Gain Selection

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

[1] Not supported on the DAQCard-16XE-50.

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	Reserved	Chan3	Chan2	Chan1	Chan0

Bit	Name	Description
15, 11–4	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.

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

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 DAQCard E Series boards follow.

Table 3-4. Calibration Channel Assignments

Type<2..0> = CAL			
Chan<3..0>	**PGIA(+)**	**PGIA(–)**	**Purpose**
0000	AIGND	AIGND	ADC Offset
0101	REF5V	AI GND	ADC Gain
0001-0100, 0110-1111	Reserved	Reserved	—

Table 3-5. Differential Channel Assignments

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

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
1100	ACh12	AISense
1001	ACh13	AISense
1110	ACh14	AISense
1111	ACh15	AISense

Table 3-7. Referenced Single-Ended Channel Assignments

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. Channel Assignments

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

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 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

4

Programming

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

Programming the 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 board discrete registers, not on the DAQ-STC, see Chapter 3, *Register Map and Descriptions*, of this manual.

The DAQ-STC has a set of Board Environment Registers that must be initialized once at the beginning of configuration. These registers must be initialized based on the properties of the hardware surrounding the DAQ-STC.

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 will need to perform a write to or a read from the specified registers. The complete examples are provided on the *E Series Register-Level Programmer Manual* Companion Disk.

PCMCIA Initialization

Before you can access the DAQ circuitry on the DAQCard E Series, the card must be activated using Card Services. PCMCIA I/O cards are kept inactive until a program has requested that Card Services activate the card by assigning an interrupt level and an address space for the card I/O registers. The DAQCard E Series requires a 32-byte I/O address window and one interrupt level.

There are at least two different ways of activating the card:

- If you are using the DAQCard E Series with National Instruments software such as NI-DAQ, LabVIEW, and LabWindows/CVI, the NI-DAQ configuration programs request the activation of the card. For more information about this procedure, see the section on installation and configuration in your NI-DAQ manual.
- If the option above is not feasible for your application, you can develop your own program to activate the card. However, this is fairly complicated, and it requires significantly more programming. If you develop your own program, you should consult the documents, Card Services Specifications and Socket Services Specifications (PCMCIA) which explain how to activate a card using system-level calls. You will need to request an I/O window, an interrupt level, and a configuration. In the configuration, the configuration index should be set to 01 hex for normal operation. For more information about these operations, see the PCMCIA documents, Card Services Specifications and Socket Services Specifications.

After you activate the card, you are ready to configure the DAQCard E Series for the DAQ setup.

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 DAQCard E Series boards have other discrete registers. This enormous I/O address space makes it impossible to use other peripheral cards with their own registers.

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 five sections: Digital I/O, Analog Input, Analog Output, General Purpose Counters, and Analog Trigger. 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 can be found in this manual. Each program is also provided in its entirety on the *E Series Register Level Programmer Manual* Companion Disk.

The Companion Disk also contains the support files necessary to run the examples. The support files include `ESERRLP.h` and `ESERFNCT.c` which should all be included with the project for each example. `ESERRLP.h` declares the external function prototypes and the register addresses; `ESERFNCT.c` contains functions which write to and read from Windowed and board discrete registers: `DAQ_STC_Windowed_Mode_Write`, `DAQ_STC_Windowed_Mode_Read`, `Board_Write`, and `Board_Read`.

Before beginning register-level programming of the analog input and analog output modules, 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 modules.

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 PCMCIA resources and if necessary change the base address in your program with the base address that you get from setting up the PCMCIA resources.
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 as well.

1. Set up the PCMCIA resources and if necessary change the base address in your program with the base address that you get from setting up the PCMCIA resources.
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 given.

Programming the DAQCard 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 function configures the board by calling the `Board_Read` and `Board_Write` functions:

```
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 *DAQCard 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. Channel 0 should be configured 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 PCMCIA resources and if necessary change the base address in your program with the base address that you get from setting up the PCMCIA resources.
2. The second step configures 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` will stop 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 DAQCard 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. Same as Step 4.
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 1ms. 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 f 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 1ms. 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 f 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 `getvect()` and `setvect()` functions 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, you should 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;
   MSC IRQ polarity = 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 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 3-8.
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 5

Example 5 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 3-8.
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 6

Example 6 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 3-9.
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 7

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 3-6.
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 8-9.
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 8

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 0-3 and ground channels 4-7. 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 3-6.
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)
```

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 10 PFI 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 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 general-purpose counter and timer module of DAQ-STC, with specific programming steps in the programming information section. Example 1 illustrates simple gated event counting. Example 2 shows buffered pulse width 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 and G_Gate trigger. Using PFI3 as G_Source and PFI4 as G_Gate. Trigger signal from G_Gate starts and stop the counter. After the G_Gate trigger, read from the save register and display the content. Counting and reading will stop when counter hits 10000.

1. Setup the PCMCIA resource and if necessary change the base address in your program with the base address that you get from setting up the PCMCIA resources.
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 setup DAC-STC for simple counting.

G0_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;
G0_Gating_Mode = 1;
G0_Gate_On_Both_Edges = 0;
G0_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 perform a reading on 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 do the reading 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 pulse width measurement. The counter uses G_In_TimeBase as G_Source to measure the signal's pulse width on PFI4 (G_Gate), counting the number of the edges that occur on G_Source. At the completion of each pulse width interval for G_Gate, the HW save register latches the counter value for software read. An interrupt informs the CPU after each measurement. Readings are done after each generated interrupt.

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

1. Perform General Purpose Counter and Timer Example 1 Steps 1-3.
2. Call Buffered_Pulse_Width_Measurement() to setup DAC-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;
   G0_Gating_Mode = 1;
   G0_Gate_On_Both_Edges = 0;
   G0_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 three delay from the trigger, pulse interval of four and pulse width of three. G_in_timebase (20 MHz) is G_source. The wave form generation is started by trigger signal from G_Gate on PFI4. Confirm operation with an oscilloscope.

1. Perform General Purpose Counter and Timer Example 1 Steps 1-3.
2. If you want to use G_In_timebase2
   Call MSC_Clock _Configure() to setup the G_In_timebase2

   Clock_and_FOUT_Register
   Slow_Internal_Timebase <= msc_slow_int_tb_enable (1)
   Slow_Internal_Time_Divide_By_2 <=msc_slow_int_tb_divide_by_2 (1)
   Clock_To_Board <= p->msc_clock_to_board_enable (1)
   Clock_To_Board_Divide_By_2 <= msc_clock_to_board_divide_by_2(1)
3. Call Cont_Pulse_Train_Generation() to setup DAC-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;
        G0_Gating_Mode = 2;
        G0_Gate_On_Both_Edges = 0;
        G0_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. And
    need 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.

```
    /*see if you 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 <= pulse width -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
}
```

Analog Triggering

The DAQCard-AI-16E-4 contains 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, high and low hysteresis are discussed. The signals connected to the High Value and Low Value are analog signals generated by CALDAC11[1] and CALDAC12. These are the remaining two CALDACs on the MB88341 or AD8804 chip. The first 10 CALDACs were used up by the calibration circuitry. For more information about the MB88341, see Chapter 5, *Calibration*, later in this manual.

One of two sources may be used as the trigger source, the PFI0/TRIG1 input on the I/O connector or the analog inputs 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 s to the PGIA path at a gain of 1. The Analog_Trigger_Drive bit in the Analog_Trigger_Etc_Register controls which input is used. The PGIA output is selected when this bit is cleared, and the PFI0/TRIG1 input is selected when this bit is set. 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 an 8-bit serial DAC to be within scale. Writing 0x00 to the DAC sets it for + full scale, while writing 0xFF sets it to – full scale. The selected input is compared against each of these thresholds by a comparator. The outputs of each comparator goes high

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

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.

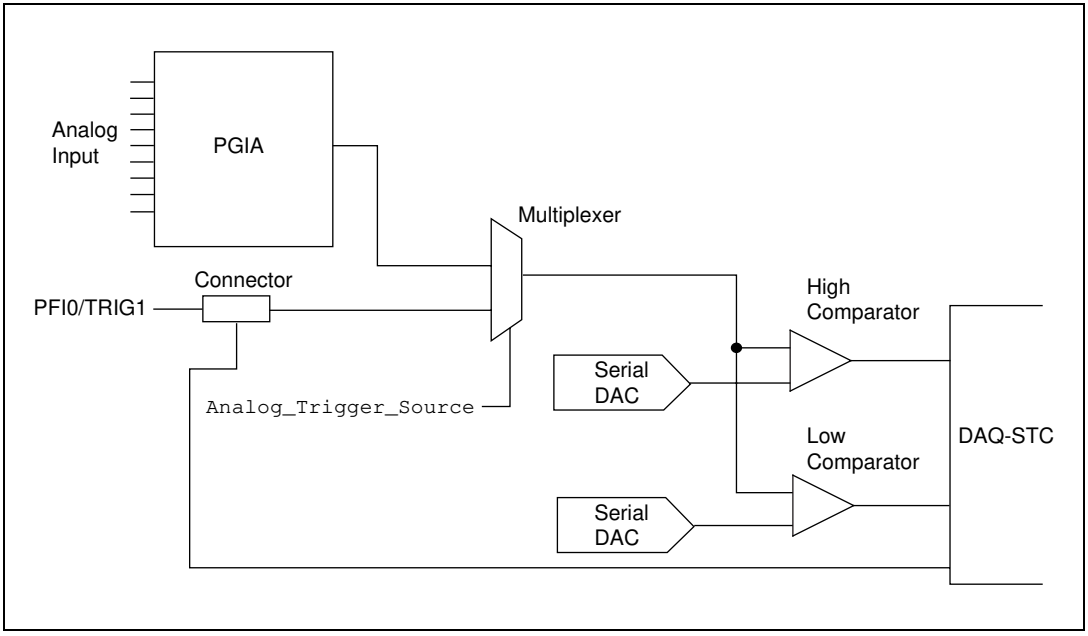

Figure 4-1. Analog Trigger Structure

To set the low and high analog thresholds, CALDACs 11 and 12 must be written with appropriate values. The protocol for writing to these DACs is the same as that for all the CALDACs mentioned in Chapter 5, *Calibration*, in this manual.

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 input. For more information about this function, see the *Programming Analog Trigger* section of Chapter 10 in the *DAQ-STC Technical Reference Manual*.

Information on the Analog Trigger example is in Chapter 10 of the *DAQ-STC Technical Reference Manual*. The following example is written for DAQCard-AI-16E-4, which uses 8-bit CALDAC for analog trigger. The

DAQCard-AI-16XE-50 does not support analog trigger. The write cycle for the 8-bit DAC is illustrated in Chapter 5 of this manual in the section. The example uses low-hysteresis mode and PGIA as the triggering source. The low value is set to be 0 V, and the high value is set to be relative 0x81 V. The software scans the analog channel 0 five times. Each scan is at a gain of 1 and in RSE mode. The scan interval is 1 ms.Within each scan, the sample interval should be 100 polled input for obtaining the data.

1. Perform Analog Input Example 1 Step 1.
2. Call the configure_Board() to clear all 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 Analog Input Example 1 Steps 3-9.
4. Call the function Number_of Scans() to load the 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 Analog Input Example 2 Steps 5-8.
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);

Misc_Command_Register (8 bits)
Int/Ext Trigger =1 (PGIA)

Writing to Serial CALDAC11
CALDAC11=0x80;

Writing to Serial CALDAC12
CALDAC12=0x81;

7. Perform Analog Input Example 2 Steps 9-10.
8. Poll the AIFIFO not empty flag in the AI_Status_Register until not empty and read the ADC FIFO 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 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. Group B handles the general purpose counter 1 interrupts, and one pass-through interrupt.

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.

5

Calibration

This chapter explains how to calibrate the analog input and output sections of the 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 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.

EEPROM

The EEPROM is used to store all non-volatile information about the board, including the factory and user calibration constants. The 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.

☞ **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) or the reserved area (the lower 256 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.

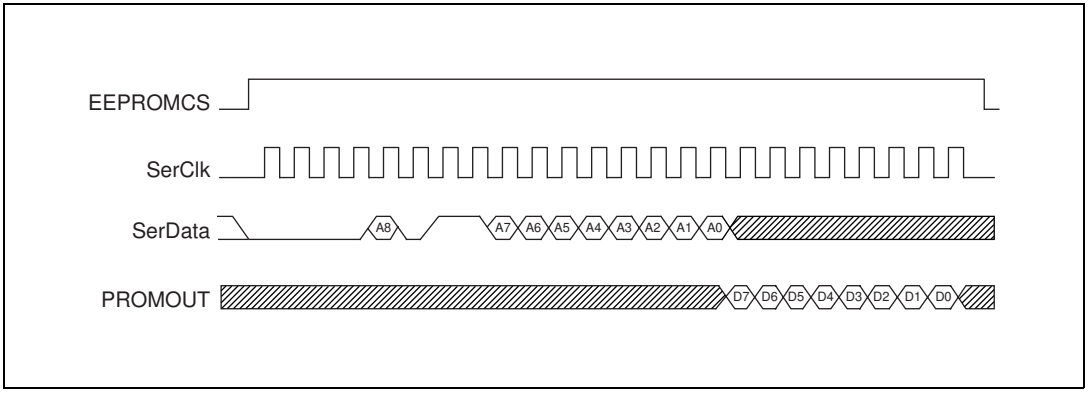

Figure 5-1. EEPROM Read Timing

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 will ensure that no writes to the EEPROM will 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 will be updated only when the LdCalDAC<2..0> bit is pulsed.

Tables 5-1, and 5-2, show a selected portion of the EEPROM map for each of the boards. The lower 256 bytes of the EEPROM are reserved. 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. DAQCard-AI-16E-4 EEPROM Map

LOC	HEX	Decimal	Description
511		Board Code	DAQCard-AI-16E-4 NI-DAQ Board Code
510	00	Revision	Revision
509	00	Sub-revision	Sub-revision
508	00	Year	Year of last factory calibration
507	00	Month	Month of last factory calibration
506	00	Day	Day of last factory calibration
437	00	0	Factory Reference LSB
436	00	0	Factory CALDAC 4 value (AI)
435	00	0	Factory CALDAC 1 value (AI)
434	00	0	Factory CALDAC 3 value (AI)[1]
433	00	0	Factory CALDAC 2 value (AI)

[1] When writing to CALDAC 3, also write the same value to CALDAC 14 for future compatibility.

Table 5-2. DAQCard-AI-16XE-50 EEPROM Map

LOC	HEX	Decimal	Description
511	??	Board Code	DAQCard-AI-16XE-50 NI-DAQ Board Code
510	00	Revision	Revision
509	00	Sub-revision	Sub-revision
508	00	Year	Year of last factory calibration
507	00	Month	Month of last factory calibration
506	00	Day	Day of last factory calibration
447	00	0	Factory Reference MSB
446	00	0	Factory Reference LSB
445	00	0	Factory CALDAC 8 value MSB (AI) - bipolar
444	00	0	Factory CALDAC 8 value LSB (AI) - bipolar
443	00	0	Factory CALDAC 2 value (AI) - bipolar

Table 5-2. DAQCard-AI-16XE-50 EEPROM Map (Continued)

LOC	HEX	Decimal	Description
442	00	0	Factory CALDAC 0 value (AI) - bipolar
441	00	0	Factory CALDAC 1 value (AI) - bipolar
440	00	0	Factory CALDAC 8 value MSB (AI) - unipolar
439	00	0	Factory CALDAC 8 value LSB (AI) - unipolar
438	00	0	Factory CALDAC 2 value (AI) - bipolar
437	00	0	Factory CALDAC 0 value (AI) - bipolar
436	00	0	Factory CALDAC 1 value (AI) - bipolar

Calibration DACs

The calibration DACs are used to adjust the analog signal paths for errors such as offset and gain. The DAQCard-AI-16E-6 uses either the MB88341 or the AD8804 serial DAC for calibration, which contains twelve 8-bit DACs. Notice that the last two DACs are used for the analog trigger circuitry on the DAQCard-AI-16E-4. The DAQCard-16XE-50 uses a DAC8800 and DAC8043 serial DAC for calibration.

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 8-bit DACs, and load for the 12-bit DAC pins, respectively.

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).

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

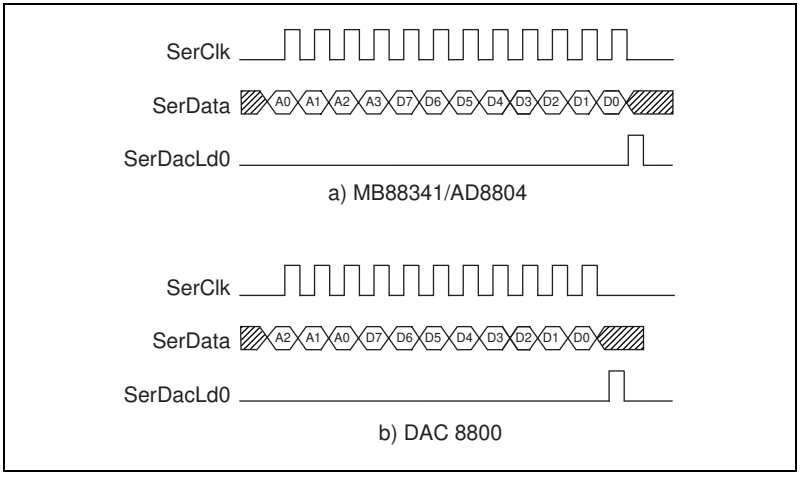

Figure 5-2. Calibration DAC Write Timing

NI-DAQ Calibration Function

The NI-DAQ function called `Calibrate_E_Series` can calibrate the analog input, analog output, and internal reference on the DAQCard 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: _____

_____

_____

_____

_____

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

Hardware revision _____

Interrupt level of hardware _____

Base I/O address of hardware _____

Programming choice _____

National Instruments software _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels 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: *DAQCard E Series Register-Level Programmer Manual*

Edition Date: November 1998

Part Number: 341080A-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
AC	alternating current
A/D	analog-to-digital
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
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
DC	direct current
DitherEn	dither enable bit
DIV	divide by signal

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
GPCT1	general-purpose counter timer 1 bit
GPCT0	general-purpose counter timer 0 bit
GroundRef	ground reference bit

H

hex	hexadecimal
Hz	hertz

I

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

L

LASTCHANNEL	last channel bit
LSB	least significant bit

M

m	meters
MB	megabytes of memory
MSB	most significant bit
MUX	multiplexer

N

NRSE	nonreferenced single-ended input

O

op-amp	operational amplifiers
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

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
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

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

U

W