DAQ

DAQ-DIO Technical Reference Manual

? 1998 Edition
Part Number 341330A-01

Preliminary

The contents of this document are preliminary and are subject to change without notice. This document may contain errors, omissions, or information that no longer applies.

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

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

Chapter 8 RTSI Connection

Tables

About This Manual

The DAQ-DIO is an application-specific integrated circuit (ASIC) designed by National Instruments. The *DAQ-DIO Technical Reference Manual* describes the programmable features of the DAQ-DIO and is intended for programmers who need to program the DAQ-DIO on an existing data acquisition (DAQ) board at the register-level.

Before using this manual, you should be familiar with the board that contains your DAQ-DIO and should have read the user manual for that board. Next, read the register-level programmer manual for the same board. The register-level programmer manual refers to some of the sections in this manual.

When you are familiar with the material in the register-level programmer manual, you can refer directly to the *DAQ-DIO Technical Reference Manual*. Programmers should read the sections on *Features*, *Programming Information*, and *Bitfield Descriptions* in each chapter to program the DAQ-DIO. Hardware engineers may need to read further for more detailed information on pin interfaces and timing signals.

Organization of the Product User Manual

The *DAQ-DIO Technical Reference Manual* is organized into two general parts. Chapters 2 through 5 describe the DAQ-DIO data transfer capabilities and digital I/O functions. Chapters 6 through 8 describe the communication between the DAQ-DIO and your computer.

- Chapter 1, *Introduction*, contains information on the DAQ-DIO operating modes. This chapter also includes important general information on programming and bus interfaces that you need to know before you start programming the DAQ-DIO at the register level.
- Chapter 2, *Unstrobed I/O*, describes features, pin interfaces, and bitfields for unstrobed I/O.

- Chapter 3, *Handshaking*, describes additional features, pin interfaces, and bitfields needed to implement the various two-way handshaking modes of the DAQ-DIO.
- Chapter 4, *Pattern Generation*, describes additional features and bitfields, in addition to those in Chapters 2 and 3, needed to implement pattern generation. This chapter includes procedures for implementing digital data acquisition and digital waveform generation, selecting internal or external timing, configuring interval counters, and establishing start and stop triggers.
- Chapter 5, *Change Detection and Pattern Detection*, describes features and bitfields for implementing change detection (transition sensing) and pattern detection (triggering on a specific input pattern).
- Chapter 6, *DMA Controls*, describes the pin interfaces and programming considerations on using the DAQ-DIO with Direct Memory Access (DMA).
- Chapter 7, *Interrupt Controls*, describes the DAQ-DIO interrupt control and explains its features, the different conditions that can trigger interrupts, and the programming of a typical interrupt service routine.
- Chapter 8, *RTSI Connection*, describes the DAQ-DIO interface for communication with a RTSI bus or PXI trigger bus.
- Appendix A, *Specifications*, lists the specifications for the DAQ-DIO.
- Appendix B, *Register Information*, contains information about the registers in the DAQ-DIO.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- *Glossary*, contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- *Index*, contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

<>	Angle brackets containing numbers separated by an ellipsis represent a range of values associated with a bit or signal name (for example, DIOB<3..0>).
☞	This icon to the left of bold italicized text denotes a note, which alerts you to important information.
⚠	This icon to the left of bold italicized text denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.
bold	Bold text denotes the names of menus, menu items, parameters, dialog box, dialog box buttons or options, icons, windows, Windows 95 tabs, or LEDs.
bold italic	Bold italic text denotes a note, caution, or warning.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept.
`monospace`	Text in this font denotes text or characters that 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.
`MONOSPACECAP`	Monospace in uppercase letters in pseudocode programs denotes a variable name.
Underline_Space	Two or more words with an underscore in the place of a space denotes a register name.
OneWord	Two or more words combined in one denotes a bitfield name.

Related Documentation

The following National Instruments documents contain general information and operating instructions for the DAQ-DIO:

- The *DIO 6533 Register-Level Programmer Manual*
- The *DIO 6533 User Manual*

The following document also contains information you will need:

- *Application Note 010: Programming Interrupts for Data Acquisition on 80 × 86-Based Computers*
- Your computer user manual

Customer Communication

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

Introduction

Chapter 1

This chapter contains information on the DAQ-DIO operating modes. This chapter also includes important general information on programming and bus interfaces that you need to know before you start programming the DAQ-DIO at the register level.

1.1 Application

The DAQ-DIO has three basic modes:

- Unstrobed I/O, for basic monitoring and control applications—for controlled communications, described in Chapter 2, *Unstrobed I/O*, each I/O line is individually configurable for direction, and drive type (standard or wired-OR).
- Handshaking, for high-speed data transfer—for high-speed data transfer, the DAQ-DIO supports request-acknowledge handshaking protocols of up to 20 MHz. Chapter 3, *Handshaking,* describes how to program the different protocols.
- Pattern generation, for digitized data acquisition and waveform generation—for pattern generation, described in Chapter 4, *Pattern Generation*, the DAQ-DIO can be programmed to input or output data at fixed, precise intervals. Pattern generation features include dual-channel DMA, interrupts, internal FIFOs, internal or external timing, programmable timebase and interval, and support pattern generation.

1.2 Programming

This section presents the DAQ-DIO information and considerations that you need to know for bitfield programming of the DAQ-DIO, including window programming.

1.2.1 Register and Bitfield Programming Considerations

Register addresses are calculated by adding the register offset to the base address assigned to the DAQ-DIO on the particular board. Table B-1 in Appendix B, *Register Information*, lists the register offsets. See the *DIO 6533 Register-Level Programmer Manual* for information about base addresses.

A bitfield is a collection of bits that controls or monitors a specific function. A bitfield can be a bit in a register, a set of functionally related bits in one register, or a set of registers that jointly perform a function. If a bitfield consists of several bits within one register, the locations of the bits are contiguous. Two or four 8-bit register sets are needed for loading certain delay values and reading the transfer count. Each set is treated as a single bitfield in this document.

Most of the DAQ-DIO registers are memory-type registers; that is, writable registers hold the values you place in them, and readable registers report their current values when you read them. In a few cases, the read or write operation itself affects the DAQ-DIO operation as follows:

- Writing one of the clear registers resets a value or function
- Reading or writing the FIFO data registers advances to the next data point in the FIFO
- In numbered mode, if all other registers have been configured, writing a transfer count register initiates a transfer

To assign values to bitfields, you should set or clear the bits of the bitfield without changing the states of the remaining bits in the register. However, except for the Window_Address Register, the writable registers on the DAQ-DIO do not have read-back capability. You cannot read these registers to determine which bits have been set or cleared in the past, so writing to these registers affects all register bits. Therefore, your software should maintain a copy of every write register, except Window_Address, in system memory. To change the state of a single bitfield without disturbing the remaining bits, perform the following steps:

1. Clear the bitfield in the software copy.
2. Place the new value in the bitfield in the software copy.
3. Write the value of the software copy to the register.

To change the state of a bitfield that spans two or more registers, you need to write to all registers. You can read or write to addresses within a double word simultaneously, as a set, if the card and bus permit. For example, on the PCI-DIO-32HS you can write up to 32 bits at a time.

Software should write only zeroes to reserved write-register bits and should treat reserved read-register bits as unknown values.

1.2.2 Start up and Shut down Considerations

At power-on or reset, the system bus or board logic provides the DAQ-DIO with an asynchronous reset signal. When reset, all configuration and internal registers are placed in their default states. The default values for most registers, other than protocol registers, are zero. At power-up, the protocol registers are not configured for any protocol. You must set the protocol registers before performing a handshaking or pattern-generation operation. RunMode is also set to zero for both groups, so that no handshaking can take place. All interrupts and DMA requests are disabled.

1.2.3 Instruction Notations

In this document, bitfield names are distinguished by the use of upper and lower cases, such as InvertReq and InvertClock. Bitfield read instructions are in the form of <`VARIABLE`> = <bitfield name>, where `VARIABLE` is a variable name. Variable names are presented in `monospace` font, such as `Flag`, and are capitalized in pseudocode, such as `FLAG`. Register names are two or more words separated by an underscore, such as Window_Data. Functions names are two or more words separated by an underscore and presented in `monospace` font, such as `Disables_Strobes_IO`.

The programming procedures are in language-independent pseudocode; that is, they instruct you to write a value to a given bitfield or register, or to detect the state of a bitfield or a register, without presenting the actual code. The procedures are presented in top-down fashion. Most of the programming sequences presented in this manual must be executed exactly as shown. The functions needed to perform a complete program are listed in the beginning of each section in the order they have to be called. The next subsections then present each function in detail. Pseudocode sequences are enclosed in braces { }. Pseudocode that contains only bitfield assignments can normally be executed in any order. The Σ symbol marks the boundary between two groups of assignments that have to be executed sequentially.

Programming the DAQ-DIO involves writing to and reading from the registers on the board. The *Bitfield Description* section of each Chapter describes details and locations for most bitfields used in that chapter. To locate descriptions of bitfields in other chapters, refer to the *Bitfield Locations* section in Appendix B, *Register Information.*

As you read this manual, you may need to refer back to functions presented in other chapters. The function map in Appendix B, *Register Information*, provides all functions used in the procedures throughout this manual.

1.2.4 Window_Address Register

Some devices, such as the PCI-DIO-32HS, allow access to all DAQ-DIO registers directly, using a 128-byte address space. Other devices, such as the AT-DIO-32HS, provide direct access to a smaller number of registers, typically the first 16 bytes. To read or write to the remaining registers on such a device, you must use windowing.

To use windowing, perform the following steps:

1. Write the offset of the register you want to access to the Window_Address Register. The DAQ-DIO rounds the offset down to the nearest multiple of four.
2. Read or write one or more of the Window_Data registers, using an offset equal to the rounding-down amount from step 1; that is, Window_Data offset = register offset MOD 4 = the remainder from the register offset divided by 4.

For example, the following steps demonstrate how to perform a 16-bit read of register offsets 30 and 31:

1. <Window_Address> = 30. Do an 8-bit write to set the Window_Address to 30. The DAQ-DIO rounds the Window_Address down by 2 to 28, to get a multiple of 4.
2. <DATA> = <Window_Data + 2>. Do a 16-bit read from Window_Data offset 2 to get the value of input registers 30 and 31 (the port C and port D input registers).

1.2.5 Chip-ID Register

The chip ID registers are read-only registers at offsets 24 through 27 that always return 68 (ASCII “D”), 73 (“I”), 79 (“O”), and the DAQ-DIO version number, which is presently 1. You can read this register to ensure that you are successsfully communicating with the DAQ-DIO and the windowing, if used, is working correctly.

1.3 Bus Interface

The DAQ-DIO offers two bus-interface modes. MITE mode is a specialized mode for communication with the National Instruments PCI MITE ASIC, which allows high-speed communication over PCI and PXI buses. This mode is available on the PCI-DIO-32HS and PXI-6533. ISA mode is a general-purpose interface for ISA-type buses. This mode available on the AT-DIO-32HS and DAQCard-6533. A few pins take on different roles in the two different bus modes.

1.3.1 Pin Interfaces

The following table describes the DAQ-DIO pin signals and, if applicable, their bus-interface modes.

Table 1-1. Mode Pins

Pin Name	Type	Description
BusMode <1..0>	Input, pulled up	Bus Interface Mode Value: 11—Strobed interface, ISA mode 0X—MITE mode. BusMode[0] serves as IODTK*.
BusType <0..1>	Input, pulled up	Bus Type—These two lines specify width and addressing scheme. Value: 11—16-bit with full address and one byte enable (ISA addressing) 01—32-bit with full address and transfer size (MITE addressing)

Table 1-2. Addressing Pins

Pin Name	Type	Description
CS <1..2>	Input	Chip Selects—CS1 is an active-low chip select. CS2 definition depends on mode: ISA Mode: CS2 is an active-high chip select (AEN). MITE Mode: CS2 is an active-low write-enable signal (Write*).
Address <6..2>	Input	Bus address lines—These address lines specify the double word to read or write. To save address space, some cards use a subset of the address lines, in which case you must use windowing to access some registers.
AQ <3..0>	Input	Address Qualifier—These lines specify the bytes to access within the double word. Definitions depend on BusType. ISA addressing: AQ[3:2] represent Address[1:0], the address of the lowest byte to access. AQ[1] represents SBHE*, system byte high enable, indicating whether the access includes the high half of the word. MITE addressing: AQ[3:2] represent Address[1:0], the address of the lowest byte to access. AQ[1:0] represent hword* and word* the MITE transfer size: Value: 11—1 byte 01—2 bytes 00 (or 10)—4 bytes.

Table 1-3. Bus Interface Pins

Pin Name	Type	Description
Read*	I/O, 4mA	Read Signal—In ISA mode, this line serves as Rd* and strobes read operations. In MITE mode, this line serves as an active-low read-enable signal sampled on the rising BusClock.
BusClock/wt*	Input	Bus Clock or Write Signal—In ISA mode, this line serves as WI* and strobes write operations. In MITE mode, this line is used for the PCI bus clock from the MITE.
Data <31..0>	I/O, 4 mA	Bus Data Lines—Data[0] connects to the least significant data bit on the bus.
Reset	Input	Reset Signal—This line carries an active-high signal that can reset all DAQ-DIO registers asynchronously.
Oscillator	Input	Oscillator—20 MHz source clock for the handshaking state machines. This signal also times clear pulses resulting from writes to the clear register.

Table 1-4. I/O Pins

Pin Name	Type	Description
DIOA<0..7>	I/O, 24mA	Port A bidirectional data lines — I/O for port A. DIOA7 is the MSB; DIOA0 is the LSB.
DIOB<0..7>	I/O, 24mA	Port B bidirectional data lines — I/O for port B. DIOB7 is the MSB; DIOB0 is the LSB.
DIOC<0..7>	I/O, 24mA	Port C bidirectional data lines — I/O for port C. DIOC7 is the MSB; DIOC0 is the LSB.
DIOD<0..7>	I/O, 24mA	Port D bidirectional data lines — I/O for port D. DIOD7 is the MSB; DIOD0 is the LSB.

Chapter
2

Unstrobed I/O

This chapter describes features, pin interfaces, and bitfields for unstrobed I/O.

2.1 Overview

The DAQ-DIO can perform unstrobed I/O using its four parallel I/O ports. Unstrobed I/O is digital I/O that employs no handshaking or hardware-controlled timing. You simply write or read directly to or from the ports. Programmable, pin-by-pin controls of all of the I/O lines allow data-path flexibility. In addition, there are four extra input lines and four extra output that you can use to send or receive serial data or auxiliary signals.

2.2 Features

The DAQ-DIO includes the following basic features for unstrobed I/O:

- Four ports with 8 I/O data pins each, totaling 32 parallel bidirectional data lines
- Pin-by-pin software control of direction
- Pin-by-pin software control of drive type for each output pin (standard or wired-OR)
- Four extra serial data inputs and four extra serial data outputs

2.2.1 Pin Directions

You can configure the direction (input or output) for each individual pin of each parallel port. Each 8-bit configuration register allows you to program the directions of the eight pins that belong to one port. By default, at board reset all I/O pins are configured as inputs.

2.2.2 Output Drive Types

You can configure the driver type (standard or wired-OR) for each individual output pin of each port. The pin masks determine which output pins are wired-OR (open-collector) drivers. The default is standard (active) drive.

2.2.3 Port Access

The four digital I/O ports of the DAQ-DIO are labeled A, B, C, and D. Each port consists of 8 parallel I/O data pins. In unstrobed I/O, you can write to or read from the port directly. When you read from a port containing input pins, the DAQ-DIO reports the input values on those pins. When you read a port containing output pins, the DAQ-DIO reports the current values driven on those pins, also. For standard (active) outputs, the value should be the same as the value you wrote; that is, you have a *read-back* capability on the value you wrote. Values being written to a port's output pins are latched in an output register.

2.2.4 Extra I/O

When you are performing only unstrobed I/O, the DAQ-DIO does not require its handshaking control and status signals to carry timing information. Therefore, you can use the control lines as extra serial inputs and outputs. Refer to Chapter 3, *Handshaking*, for information on handshaking control.

You can program the two request (REQ) lines and the two stoptrigger (STOPTRIG) lines as extra data inputs. You can read data in the REQ1, REQ2, STOPTRIG1, and STOPTRIG2 bits of the Group_Status Register.

You can program the two acknowledge (ACK) and the two peripheral (PCLK) lines as extra data outputs. Changing the values of the InvertAck and InvertClock pins, which are also the polarity controls for the ACK line and PCLK, then writes data to the ACK and PCLK lines respectively. You can also configure the output lines to drive wired-OR type signals.

2.3 Pin Interface

The following table describes the specific use of some pin signals for unstrobed I/O. For the pin signals of the four I/O ports, refer to Section 1.3, *Bus Interface*.

Table 2-1. Unstrobed I/O Pins

Pin Name	Type	Use in Unstrobed I/O
ACK1 (OUT3)	Output, 24mA	Group 1 acknowledge — For unstrobed I/O, this line can serve as an extra output.
ACK2 (OUT4)	Output, 24mA	Group 2 acknowledge — For unstrobed I/O, this line can serve as an extra output.
PCLK1 (OUT1)	Output, 24mA	Group 1 peripheral clock — For unstrobed I/O, this line can serve as an extra output.

Table 2-1. Unstrobed I/O Pins (Continued)

Pin Name	Type	Use in Unstrobed I/O
PCLK2 (OUT2)	Output, 24mA	Group 2 peripheral clock — For unstrobed I/O, this line can serve as an extra output.
REQ1 (IN3)	Input, 24mA	Group 1 request — For unstrobed I/O, this line can serve as an extra input.
REQ2 (IN4)	Input, 24mA	Group 1 request — For unstrobed I/O, this line can serve as an extra input.
STOPTRIG1 (IN1)	Input, 24mA	Group 1 stop trigger — For unstrobed I/O, this line can serve as an extra input.
STOPTRIG2 (IN2)	Input, 24mA	Group 2 stop trigger — For unstrobed I/O, this line can serve as an extra input.

2.4 Programming Information

This section contains programming details for unstrobed I/O mode for the data ports and extra I/O lines. Most of the bitfields used in the following pseudocode procedures are described in Section 2.5, *Bitfield Descriptions*. Refer to the *Bitfield Locations* in Appendix B, *Register Information*, to locate bitfield descriptions not documented in this chapter. To locate other fuction calls in this manual, refer to the *Index*. For general programming considerations, instruction notation, and window programming information, refer to Chapter 1, *Introduction*.

2.4.1 Programming Steps and Pseudocode Example for Unstrobed I/O

This section contains the programming steps and a sample program for bit-level programmers to configure and perform the unstrobed I/O mode.

Use the following steps for programming the unstrobed I/O mode:

1. Select the I/O ports you want to configure.
2. If you have used the ports to perform handshaking I/O, disable the corresponding FIFOs, releasing the ports from their handshaking groups.
3. Configure the drive types in the Port_Pin_Mask registers while the pin directions are set to inputs.
4. You can use the Port_Output registers to write the initial data values to the ports before making any pins into outputs.
5. Write the port directions to all lines of the ports.

6. Write directly to the port output registers for output, or read from the port input registers for input.

⚠ **Caution:** ***To avoid using the incorrect drive type between writes to the direction and mask registers, it is recommended that you configure the mask registers while the pin directions are set to inputs.***

Unstrobed I/O Pseudocode

```
{
    call Disable_Strobed_IO; /* Call this function only if the port you want to use is
    assigned to a handshaking group and you do not intend to use it for handshaking any
    more */
    call Configure_Port;
    call Unstrobed_Data_Transfer;
}
```

2.4.1.1 Disabling Strobed I/O

Function Disable_Strobed_IO

```
{
    /* If the port that you wish to use is assigned to a strobed-I/O group, write the values in
    this function to the registers of that group to release the port for unstrobed I/O. */
    RunMode = 0; /* Stop strobed I/O for the group*/

    /* If you have funneling enabled for the port you wish to use, turn it off. */
    If (Funneling <> 0) and (you wish to use port A for group 1 or port C for group 2)
        Funneling = 0; /* Turn off funneling */

    /* If you have funneling disabled but FIFO buffering enabled for the port you wish to use,
    turn FIFO buffering off. Enabling FIFO buffering assigns a port to strobed I/O. */
    If (Funneling == 0)
        FIFOEnable = 0; /* Disable the FIFO for each port to be used in unstrobed I/O. */
}
```

2.4.1.2 Configuring Ports

Function Configure_Port

```
{
    /* Write the values in this function to the registers of the ports you wish to use. */
    /* Bit 0 of each 8-bit value corresponds to pin 0 of the I/O port, and so on. */

    /* For all output pins, set the pin mask to select the drive type: 0 = standard, or 1 =
    wired-OR. */
    /*For input pins, the mask value has no effect on unstrobed I/O, although it does affect
    pattern generation and change detection. */
    PinMask = desired mask value (8 bits for each port);
}
```

2.4.1.3 Transferring Data

Function Unstrobed_Data_Transfer

```
{
    /* Read and write the values in this function to and from the registers of the ports you
    wish you use. */
        declare variable DATA;
        if (performing input on one or more pins)
        {
            DATA = Pin_In; /* Ignore the bits coming from the output pins. */
        }
        if (performing output on one or more pins)
        {
            Pin_Out = output data
    }
}
```

2.4.1.4 Unstrobed I/O Example

The following example toggles the I/O lines of port A and port B. This program also illustrates how to employ window programming to access different DAQ-DIO registers on a device that does not give direct access to all registers, due to address space limitations.

Window Programming Steps:

1. Write the port address to the Window_Address register.
2. Write data to the Window_Data register in the case of output or read the data from the register in the case of input.

Function Unstrobed_IO_Example

```
{
    /* First, call Disable_Strobed_IO, if you have been doing strobed I/O with port A
    or B. */
    /* Next, call Port_Config to configure port A to pin directions 0xFF (all outputs) and
    port B to pin directions 0x00 (all inputs). */

    /*Write data to port A using window registers. */
    declare variable DATA;
    [WindowAddress] = 28;          /* Port_A_Output address. */
    [WindowData] = data (8-bit);      /*Since Port_A_Output address is a multiple
                                      of 4, no need to adjust WindowData address. */

    /* Read data to port B using window registers. */
    declare variable DATA;
    [WindowAddress] = 29;             /* Port_B_Input address.*/
    DATA = [WindowData + 1];          /* Add 29 MOD 4, or 1, to WindowData address. */
}
```

2.4.2 Extra Input and Output line Programming

This section illustrates how to program the REQ and STOPTRIG data lines as extra unstrobed inputs, and the PCLK and ACK lines as extra unstrobed outputs.

Programming steps:

1. Enable the corresponding lines as extra inputs or outputs.
2. Configure driver types for any extra outputs.
3. Write or read data to or from the specified register.

Extra I/O lines pseudocode

```
{
    call Extra_Lines_Config;
    call Extra_Lines_Data_Transfer;
}
```

2.4.2.1 Configuration of the Control Lines for Extra I/O

Function Extra_Lines_Config

```
{
    /* The STOPTRIG line does not need a configuration step */

    ReqSource = 0 (default);          /* Source = I/O connector */
    InvertReq = 0;                    /* No inversion */
    ReqConditioning = 1;              /* No conditioning */

    Σ
    AckLine = 0 (default);            /* For glitch-free configuration, tristate the
                                      ACK line before configuring OpenAck */
    OpenAck = drive type;             /* 0 = standard or 1 = open collector */
    Ackline = 1;                      /* Use ACK as extra output */
    Σ
    Σ
    PclkLine = 0;                  /* For glitch-free configuration, tristate the Pclk
                                   line before configuring OpenClock */
    OpenPclk = drive type;         /* 0 = standard or 1 = open collector */
    PclkLine = 1;
    Σ
}
```

2.4.2.2 Transferring Data using extra I/O lines

Function Extra_Lines_Data_Transfer

```
{
    /* Read data from STOPTRIG line and REQ line */
    declare variable DATA;
    DATA = StopTrigStatus;
    DATA = ReqStatus;

    /* Write data to ACK and PCLK lines through the InvertAck and InvertReq registers. */
    InvertAck(2) = data (one bit);     /* Serial data to output to ACK line */
    InvertClock(2) = data (one bit);   /* Serial data to output to PCLK line */
}
```

2.5 Bitfield Descriptions

This section describes details and locations for most bitfields used for unstrobed I/O. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate any bitfields not documented in this section.Table B-1 in Appendix B also contains information on all DAQ-DIO registers.

AckLine

bits:	**type:**	**register:**	**address:**
<5..6>	Write	Protocol_Register_7	74 (Group 1) 106 (Group 2)

These bits determine the signal on the ACK line.

0: Tristate the ACK Line.

1: Use the ACK line as an extra output. Drive the value of InvertAck onto the line.

ClockLine

bit:	**type:**	**register:**	**address:**
<5..6>	Write	Protocol_Register_2	66 (Group 1)
			98 (Group 2)

These bits determine the clock signal to drive on the PCLK, if any.

0: Tristate the PCLK line.
1: Use the PCLK line as an extra output. Drive the value of InvertClock onto the line.

InvertAck

bit:	**type:**	**register:**	**address:**
0	Write	Protocol_Register_6	73 (Group1)
			105 (Group 2)

This bit inverts the polarity of the ACK signal if set. When using the ACK line as extra output, use this bit to write data to the ACK line.

InvertPclk

bit:	**type:**	**register:**	**address:**
2	Write	Protocol_Register_6	73 (Group1)
			105 (Group 2)

This bit inverts the polarity of the PCLK signal if set. When using the PCLK line as extra output, use this bit to write data to the PCLK line.

InvertReq

bit:	**type:**	**register:**	**address:**
1	Write	Protocol_Register_6	73 (Group1)
			105 (Group 2)

Setting this bit inverts the polarity of the REQ signal. When using the REQ line as extra input, set this bit to zero to avoid inverting the polarity of the incoming signal received on the REQ line.

PinDirections

bits:	**type:**	**register:**	**address:**
<0..7>	Write	Port_A_Pin_Directions	32 (Port A)
		Port_B_Pin_Directions	33 (Port B)
		Port_C_Pin_Directions	34 (Port C)
		Port_D_Pin_Directions	35 (Port D)

These bits set the direction for each pin of the ports. Each bit in this register corresponds to one pin at the port.

0: input
1: output

PortInput

bit:	**type:**	**register:**	**address:**
<0..7>	Write	Port_A_Input	28 (Port A)
		Port_B_Input	29 (Port B)
		Port_C_Input	30 (Port C)
		Port_D_Input	31 (Port D)

These bits allow you to read values of each pin of a port in unstrobed I/O mode. Each bit corresponding to one pin of the corresponding port.

PinMask

bits:	**type:**	**register:**	**address:**
<0..7>	Write	Port_A_Pin_Mask	36 (Port A)
		Port_B_Pin_Mask	37 (Port B)
		Port_C_Pin_Mask	38 (Port C)
		Port_D_Pin_Mask	39 (Port D)

For output pins, mask bits determine whether the corresponding pins use standard (active) or wired OR (open-collector) drivers. The default, 0, indicates standard (active) drive. Each bit in this register corresponds to one pin at the port. For input pins, the pin masks determine which pins are significant for change detection and pattern detection (described in Chapter 5, *Change Detection and Pattern Detection*). The default, 0, indicates significance.

PortOutput

bit:	**type:**	**register:**	**address:**
<0..7>	Write	Port_A_Output	28 (Port A)
		Port_B_Output	29 (Port B)
		Port_C_Output	30 (Port C)
		Port_D_Output	31 (Port D)

These bits allow you to write values to each output pin of a port in unstrobed I/O mode. Each bit corresponds to one pin of the corresponding port.

ReqConditioning

bits:	**type:**	**register:**	**address:**
<3..5>	Write	Protocol_Register_4	70 (Group 1)
			102 (Group 2)

These bits determine the synchronization to apply to the REQ line.

0: Synchronize the REQ line.
1: Do not condition the REQ line at all. Use REQ without synchronization.

ReqStatus

bit:	**type:**	**register:**	**address:**
<2> (Group 1)	Read	Group_Status	5
<6> (Group 2)			

This bit shows the current value of the group request line (or the extra input line if no handshaking is used), after any inversion and conditioning.

StopTripStatus

bit:	**type:**	**register:**	**address:**
<3> (Group 1)	Read	Group_Status	5
<7> (Group 2)			

This bit shows the current value of the group STOPTRIG line (or the extra input line if no stop trigger is in use), after any inversion and conditioning.

WindowAddress

bits:	**type:**	**register:**	**address:**
<0..6>	Write	Window_Address	4

When performing windowed accesses, write to this register the address offset of the register you want to read or write. The DAQ-DIO ignores bits <0..1> of the offset you write, thereby rounding the offset down to the nearest multiple of four.

WindowAddressStatus

bit:	**type:**	**register:**	**address:**
<2..6>	Read	Interrupt_and_Window_Status	4

These bits show bits <2..6> of the last offset written to the Window_Address Register, giving you readback capability on these bits. Note that bits <0..1> and <7> of the Interrupt_and_Window_Status are not part of the WindowAddressStatus and should be ignored.

WindowData

bit:	**type:**	**register:**	**address:**
(see below)	Read	Window_Data	special (see below)

This four-byte bidirectional register is used for window programming. You can access any DAQ-DIO register by first writing the offset to the WindowAddress, and then reading or writing the Window_Data Register. To do a 16-bit or 32-bit read or write (if your device and bus support them) by windowing, do a 16-bit or 32-bit read or write to the Window_Data Register.

The address of the WindowData itself is equal to bits <0..1> of the register offset; in other words, equal to the portion of the register offset lost in rounding during the write to the Window_Address Register. For example, to access register 10, you would write a 10 to WindowAddress, which would be rounded to 8, because the DAQ-DIO ignores bits <0..1>. Then you would access the WindowData at address 2 to make up for the rounding.

In windowing, the DAQ-DIO uses address bits <2..6> from the WindowAddress value and bits <0..1> from the address you use to access the Window_Data Register. This allows maximum flexibility for supporting any type of transfer your device and your bus allow.

Chapter 3

Handshaking

This chapter describes additional features, pin interfaces, and bitfields needed to implement the various two-way handshaking modes of the DAQ-DIO. Handshaking is closed-loop strobed operation in which data transfers are timed and confirmed by an exchange of ACK and REQ signals.

3.1 Overview

Besides the simple, unstrobed I/O described in Chapter 2, *Unstrobed I/O*, the DAQ-DIO can perform high-speed data transfer by strobed I/O. Strobed I/O is data transfer in which the DAQ-DIO hardware regulates timing or performs handshaking functions. Two-way handshaking is one category of strobed I/O. Pattern generation, or one-way handshaking, is another category of strobed I/O and is discussed in Chapter 4, *Pattern Generation*.

In two-way handshaking mode, control information passes both to and from the peripheral. The term *peripheral* refers to any external device that the DAQ-DIO controls, monitors, tests, or communicates with. The DAQ-DIO and the peripheral each provides the other with strobe signals when data is available or acquired. By withholding strobe signals, either the DAQ-DIO or the peripheral can slow down the transfer, if necessary.

3.2 Features

The DAQ-DIO includes the following basic features for two-way handshaking:

- Two handshaking channels, referred to as groups, with separate timing logic and control registers for each group
- An independent set of handshaking control lines for each group
- Software control of handshaking protocol and other options for each group
- Four internal FIFOs, each associated with a port
- Funneling that allows software to modify the routing of FIFOs to ports
- Counters to control the number of transfers performed

3.2.1 Port Registers

In strobed I/O, unlike unstrobed I/O, you do not use the port input or output registers to read or write data. Instead, you use the internal FIFOs to perform data transfers. The value of an output port's output register only serves as the port initial value and reset value. When you reset a group by writing a zero to the RunMode field in the Op_Mode Register, the output ports reset to the values in the port output registers. Once the group starts to transfer data using the group's FIFOs, the values written to the port's output register do not appear on the data lines. However, you can still configure any port that is not allocated to a group for unstrobed I/O functions.

3.2.2 Internal FIFOs

The DAQ-DIO provides an internal FIFO for every port. This FIFO is a bidirectional, first-in first-out, 8-bit wide memory buffer. Four internal FIFOs are provided in total. They are labeled A, B, C, and D, according to the port they are associated with by default. Each internal FIFO is 16 words deep. By using FIFOs, the hardware performs all handshaking timing without involvement from the CPU, and you can, therefore, configure all timing parameters through software.

3.2.3 Groups

For strobed I/O, you must allocate ports and FIFOs to handshaking groups. A group consists of a strobed-I/O timing controller and a set of four associated control lines: REQ, ACK (STARTTRIG), STOPTRIG, and PCLK. The DAQ-DIO supports up to two independent groups.

To allocate a FIFO and its corresponding I/O port to a group, set the enable bit for that FIFO in the group's Data_Path Register. For example, by allocating FIFOs A and B to group 1, you can perform a 16-bit strobed transfer using the group 1 controller. You can modify the association of ports to FIFOs and groups using funneling, as described in Section 3.2.4, *Funneling*. Do not assign a single FIFO or port to both groups.

Set the GroupDirection bit to 1 for output or 0 for input. Also, for each port you are using, set all of the port's direction bits in the PinDirections bitfield to match the direction of the group. Unlike unstrobed I/O, strobed I/O does not allow pin-by-pin direction control.

After assigning a port to a strobed-I/O group, use the group's FIFO I/O registers, instead of the port I/O registers, to read data from an input group or write data to an output group.

3.2.4 Funneling

Funneling allows you to modify the connection of ports to FIFOs. Funneling allows you to configure an I/O size smaller than your group size; that is to use fewer ports than FIFOs. 8:16 funneling allows you to do 8-bit I/O to a 16-bit group, assuming that you transfer an even number of points. The advantage of using 16-bit group to do 8-bit I/O is that you can do 16-bit reads, writes, and DMA operations, increasing efficiency. Some DAQ-DIO devices, such as the AT-DIO-32HS, may support only 16-bit DMA; see Section 3.2.5, *TransferWidth*, for more information.

For group 1, 8:16 funneling always uses port A, and you should enable FIFOs A and B when using this mode. For group 2, 8:16 funneling always uses port C, and you should enable FIFOs C and D.

3.2.5 TransferWidth

The TransferWidth bitfield configures the width of the read, write, or DMA accesses the group supports. This must be less than or equal to the total width of the group. For example, if you plan to use 16-bit DMA, choose a transfer width of 16 bits, even if you are doing a 32-bit operation using all four FIFOs and all four ports. Your bus and your device limit the maximum transfer width. For example, the PCI-DIO-32HS and PXI-6533 support 8-bit, 16-bit, and 32-bit reads, writes, and DMA, while the AT-DIO-32HS supports only 8-bit and 16-bit reads and writes, and only 16-bit DMA. The DAQCard-6533 supports only 8-bit and 16-bit reads and writes.

Note: ***Set the TransferWidth bitfield before beginning read, write, or DMA accesses to or from the FIFOs.***

3.2.6 Run Mode

The RunMode bitfield controls the start, stop, and reset of a strobed operation. Always set a group's RunMode to reset when altering any configuration registers that affect the operation of the group.

The DAQ-DIO supports two transfer modes: numbered, in which you perform a finite, predetermined number of transfers, and unnumbered, in which the transfer continues indefinitely until you set the RunMode to reset.

3.2.7 Control Signals

Each group has its own, independent set of handshaking control and status lines for strobed I/O. In two-way handshaking mode, the control lines used are ACK, REQ, and, for burst mode, PCLK. Therefore, the DAQ-DIO can perform up to two handshaking operations simultaneously. The operations can be input transfers, output transfers, or one of each.

The American National Standards Institute (ANSI) does not transmit any handshaking information on the data lines themselves.

In two-way handshaking, the REQ signal for each group is an input signal for accepting requests from the peripheral, and the ACK signal is an output signal for sending acknowledgment to the peripheral. The terms *request* and *acknowledge* refer only to the direction of the handshaking signal, and the acknowledge signal may precede the request signal in some protocols. Typically, the data receiver produces its ready signal first, and the data sender, when ready, initiates the transfer of data.

3.2.8 Protocols

The DAQ-DIO supports several handshaking protocols. For descriptions and timing of these protocols, see your user manual.

The supported protocols are the followings:

- 8255 emulation mode
- Level-ACK mode
- Leading-edge pulse mode
- Long pulse pulse mode
- Trailing-edge pulse mode
- Burst mode

3.2.9 Running Mode Options

[Vince, rewrite?]

3.3 Pin Interface

The following table describes the specific use of some pin signals in two-way handshaking I/O. For the pin signals of the four I/O ports, refer to Section 1.3, *Bus Interface*.

Table 3-1. Two-Way Handshaking Pins

Pin Name	Type	Use in Handshaking
ACK1 (STARTTRIG1)	Output, 24mA	Group 1 output handshaking acknowledge signal — In output mode, this signal becomes active when data has been written to the data lines. In input mode, this signal becomes active when the available data on the data lines has been read.
ACK2 (STARTTRIG2)	Output, 24mA	Group 2 output handshaking acknowledge signal — In output mode, this signal becomes active when data has been written to the data lines. In input mode, this signal becomes active when the available data on the data lines has been read.
PCLK1	I/O, 24mA	Group 1 peripheral clock — In burst-mode handshaking, this line carries a clock signal to or from a peripheral.
PCLK2	I/O, 24mA	Group 2 peripheral clock — In burst-mode handshaking, this line carries a clock signal to or from a peripheral.

Table 3-1. Two-Way Handshaking Pins (Continued)

Pin Name	Type	Use in Handshaking
REQ1	Input, 24mA	Group 1 input handshaking request line—In output handshaking, the peripheral should activate this signal to indicate that it is ready to receive data. In input handshaking, the peripheral should activate this signal when data is available to be read on the data lines.
REQ2	Input, 24mA	Group 2 input handshaking request line—In output handshaking, the peripheral should activate this signal to indicate that it is ready to receive data. In input handshaking, the peripheral should activate this signal when data is available to be read on the data lines.

3.4 Programming Information

This section illustrates the steps to perform two-way handshaking data transfer, which includes accessing the internal FIFOs, and things you need to do after each run. This section also tells you the steps for configuring different protocols described in the previous section.

Most of the bitfields used in the following pseudocode procedures are described in Section 3.5, *Bitfield Descriptions*, except for the interrupt-related flags and the flag-clearing bits. Refer to *Bitfield Locations* in Appendix B, *Register Information*, to locate bitfield descriptions not documented in this chapter. For port configuration, refer to the function calls in Chapter 2, *Unstrobed I/O*. To locate other function calls in this manual, refer to the *Index*. For general programming considerations, instruction notation, and window programming information, refer to Chapter 1, *Introduction*.

3.4.1 Programming Steps and Pseudocode Example for Handshaking

This section contains the programming steps and a sample program for bit-level programmers to configure and perform the handshaking I/O mode.

Use the following steps for programming handshaking mode:

I. Configure handshaking mode

1. Select a group for the transfer. Reset the handshaking group by setting the RunMode bitfield to 0.
2. Assign one or more FIFOs (with corresponding ports) to the group using the Data_Path Register.
3. Assign the group and port pin directions, either to input or output.
4. Set the TransferSize bitfield to specify the size of the reads and writes you intend to perform (8-bit, 16-bit, or 32-bit).
5. Select and program a handshaking protocol, using the values from Table 3-2 in this chapter.
6. In the case of output, preload initial data into the FIFOs.

II. Run handshaking mode

7. If using numbered mode, set the numbered mode bit and write a transfer count.
8. Set RunMode to run.
9. Perform the transfers.
10. If using numbered mode, wait for the CountExpired flag to assert.
11. Set RunMode to 0 (reset).

III. Clean up FIFOs and flags

12. In the case of input, check the DataLeft flag. Empty the FIFOs if there is data left.
13. Clear the CountExpired flag and any other flags used.

Caution: ***The RunMode field should be held at 0 during any configuration changes.***

Full Handshaking Pseudocode

```
{
    call Handshaking_Config;
    call Handshaking_Run;
    call Cleanup;
}
```

3.4.2 Configuring Handshaking Mode

Function Handshaking_Config

```
{
    RunMode = 0;
    call Group_Config;
    call Protocol_Config;
    Clear_Flags = 255                              /*Clear all flags*/
    if (group is output) call Preload_FIFOs;       /* Preload data for output groups */
}
```

3.4.2.1 Group Configuration

Function Group_Config

```
{
    set one or more FIFOEnable bits to assign FIFOs to the group;
    Funneling = 0 (no funneling) or 2 (8-bit funneling);
    GroupDirection = 0 (input) or 1 (output); /* The direction should match the direction of
    the protocol you are going to use. */

    /* Group 2 can be Configured the same way if used */

    call Port_Config for each port to be used      /* Pin directions should match the
                                                   group direction */

    TransferWidth = 0 (32-bit FIFO reads and writes) or 2 (8-bit FIFO reads and writes) or 3
    (16-bit FIFO reads and writes);
}
```

3.4.2.2 Protocol configuration

To program the DAQ-DIO for different protocols, set the protocol registers to the parameters listed in Table 3-2.

Function Protocol_Config;

```
{
    Σ
    select a handshaking group and a protocol;
    set the group's protocol configuration registers, Protocol_Register_1 through
    Protocol_Register_14, to the values listed in Table 3-2;
    Σ
    if (using burst mode) ReadyLevel = 0-4;        /* see bitfield descriptions */
    if (group use stand-alone burst input mode, or is the last one in the line of cascading input mode,
    or use serial mode and clock source is internal) <end of line> ClockSpeed = desired factor to slow
    down transfer
}
```

Table 3-2. Handshaking Parameters

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Protocol\Protocol Register	**Op Mode**	**ClockReg**	**Sequence**	**ReqReg**	**BlockMode**	**LinePolarities**	**AckSer**	**StartDelay**	**ReqDelay**	**ReqNotDelay**	**AckDelay**	**AckNotDelay**	**Data1Delay**	**ClockSpeed**	**DAQ Options**
8255 Input Emulation	96	5	38	0	0[1]	3	96	pd x 2 + 1	2	1	2	1	1	0	0
8255 Output Emulation	0	0	38	0	0 [1]	3	96	pd x 2 + 1	2	1	2	1	1	0	0
Level Mode Input	0 non 64 latch[2]	1	7	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1 [4]	3	pd x 2 + 2 [4]	pd x 2 + 2 [4]	2	1	0	0
Level Mode Output	0 non 96 latch[2]	0	7	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1 [4]	3	pd x 2 + 2 [4]	pd x 2 + 2 [4]	2	1	0	0
Leading Edge Pulse Mode Input	0 non 64 latch[2]	1	11	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1 [4]	3	pd x 2 + 2 [4]	pd x 2 + 2 [4]	3	1	0	0
Leading Edge Pulse Mode Output	0 non 96 latch[2]	0	11	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1 [4]	3	pd x 2 + 2 [4]	pd x 2 + 2 [4]	3	1	0	0
Long Pulse Mode Input	0 non 64 latch[2]	1	7	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1	3	pd x 2 + 2	2	3	1	0	0
Long Pulse Mode Output	0 non 64 latch[2]	0	7	0	0 [1]	1 inv ACK[3] 2 inv REQ [3] 3 both inv[3]	96	pd x 2 + 1	3	pd x 2 + 2	2	3	1	0	0
Trailing Edge Mode Input	0 non 96 db buf latch	1	2	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1	pd x 2 + 7 [4]	3	2	pd x 2 + 7 [4]	1	0	0
Trailing Edge Mode Output	0 non 96 db buf[2]	0	2	0	0 [1]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	pd x 2 + 1	pd x 2 + 7 [4]	3	2	pd x 2 + 7 [4]	1	0	0
Burst Mode Input (drive PCLK)	0	96	0	8	4 [5]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	1	1	1	1	1	1	pd	0
Burst Mode Input (receive PCLK)															0
Burst Mode Output (receive PCLK)	0	16	0	32	4 [5]	1 inv ACK[3] 2 inv REQ[3] 3 both inv[3]	96	1	1	1	1	1	1	0	0

Table 3-2. Handshaking Parameters (Continued)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Protocol\Protocol Register	**Op Mode**	**ClockReg**	**Sequence**	**ReqReg**	**BlockMode**	**LinePolarities**	**AckSer**	**StartDelay**	**ReqDelay**	**ReqNotDelay**	**AckDelay**	**AckNotDelay**	**Data1Delay**	**ClockSpeed**	**DAQ Options**
Burst Mode Output (????)															0

[1] Set to 0; if performing 8:16 funneling, set to 24.

[2] Use the former value if you wish to disable REQ-edge latching (default mode for output). Use the latter value if you wish to enable REQ-edge latching (default mode for input).

[3] Set to 0 for active-high ACK and REQ, 1 for active-low ACK, 2 for active-low REQ, or 3 for active-low ACK and REQ.

[4] pd = programmable delay, from 0 to 7, in hundreds of ns; see your hardware user manual for the efect of this delay on protocol timing.

[5] Set to 4; if performing 8:16 funneling, set to 0.

3.4.2.3 FIFO Access

In strobed I/O, you must use the FIFOs, rather than the port I/O registers, to read or write data. To read or write the group 1 FIFOs use offset 8, and for group 2 use offset 12, regardless of which FIFOs belong to the group. You can read only from input group and write only to output groups. The size of your read or write must match the TransferWidth you configured, either 8-bit, 16-bit, or 32 bit; see Section 3.2.5, *TransferWidth*, for more information.

The following pseudocode demonstrates the Group FIFO accesses.

Function Preload_FIFOs

```
/* Preload data into the FIFOs in output mode */
{
    Declare variable FLAG
    FLAG = TransferReady
    {
        GroupFIFO (of output group) = data (8-bit, 16-bit, or 32-bit, depending on
        TransferWidth)
        FLAG = TransferReady
    }
```

3.4.3 Running Handshaking

Function Handshaking_Run

```
{
    if (using numbered mode)
        {
            Σ
            Numbered = 1;
            TransferCount = number of points to transfer;
            Σ
        }

    RunMode = 7;
    Σ
    call function Perform_Transfer;
    call function Stop_Transfer;
    Σ
}
```

3.4.3.1 Start Running

The following pseudocode shows how to perform the two-way handshaking transfer using the FIFOs. The example uses numbered mode.

Note: ***For continuous mode, replace the for loop with a continuous loop. For stoptriggered mode, replace the for loop with a while Count Expired = 0 loop; see Chapter 4, Pattern Generation, for more information.***

Function Perform_Transfer

```
{
    declare variables FLAG, DATA, INDEX;
    if (group is output)
    {
        for INDEX = 1 to number written to TransferCount minus the number of preloaded
        data
        {
            do
                FLAG = TransferReady (of the output group);
            while FLAG  = 0; /* until a 1 in the TransferReady flag is detected */
            GroupFIFO (of the output group) = data (8-bit, 16-bit, or 32-bit, depending on
            TransferWidth);
        }
    }
    if (group is input)
    {
        for INDEX = 1 to number written to TransferCount
        {
            do
                FLAG = TransferReady (of the input group);
            while FLAG = 0; /* wait until the TransferReady flag is detected */
            DATA (8-bit, 16-bit, or 32-bit, depending on TransferWidth) = GroupFIFO (of
            the input group);
        }
    }
}
```

3.4.3.2 Stop Running

The following pseudocode shows how to stop the handshaking transfers. For output, allow any data in the FIFOs to handshake out before stopping. For either input or output, set RunMode to 0 to stop the strobed I/O to or from the peripheral. For input, if necessary, empty any remaining data from the FIFOs. Clear any flags that remain set.

Function Stop_Transfer

```
{
    declare variable FLAG, DATA;
    if (group is output)
    {
            /* Wait for FIFOs to empty. */
            if (use numbered mode)
            {
                    do
                        FLAG = CountExpired;
                    while FLAG = 0;
            }
            else
            {
                    do
                        FLAG = DataLeft;
                    while FLAG = 1;
            }
    }
    RunMode = 0;        /* Stop strobed I/O */
    if (group is input and not using numbered mode)
            call Empty_FIFOs
    }
    Clear Flags = 255;   /* Clear all flags; reset FIFOs. */
}
```

3.4.3.3 Empty Input FIFOs

When using numbered mode, you know the number of data to read, so no data should remain in the FIFOs at the end of the transfer. However, in unnumbered mode, you may not know the amount of data to acquire and, therefore, data may remain in the FIFOs after stopping.

Function Empty_FIFOs

```
{
    /* Read any data left in FIFOs */
    FLAG = TransferReady;
    if (FLAG = 1)
        do
        {
            DATA (8-bit, 16-bit, or 32-bit, depending on TransferWidth) = GroupFIFO;
            FLAG = TransferReady;
        }
        while FLAG = 1;
}
```

3.4.3.4 Restarting

After stopping a transfer, you can start another transfer using the same configuration by calling the `Handshaking_Run` function, or reconfigure the group by calling the `Handshaking_Config` function. If you wish to move FIFOs from one group to another, remove them from the first group before assigning them to the second.

Function PG_Config

```
/* Call the Group_Config function, presented earlier in this chapter, to configure a group for
output (pattern generation) or input (data acquisition) and to enable one or more FIFOs. */
RunMode = 0;
call Group_Config;

/* Select start and stop triggers, if any, and make appropriate register adjustments. */
StartSource = 0 (software triggering), 1, 2, or 3
StopSource = 0, 1, 2, or 3
call Make_Adjustments;

/* Select trigger polarities. */
if (StartSource = 1 or 2) and (triggering on falling edge)
    InvertStart = 1;
if (StopSource = 1 or 2) and (trigger on falling edge)
    InvertStop = 1;
if (StartSource = 3 or StopSource = 3) and (trigger on mismatch)
    InvertMatch = 1;
```

Function Make_Trigger_Adjustments

```
/* This function computes the correct values of the Prestart bit and the TransferCount
adjustment, based on the trigger selected, to synchronize the trigger logic with the data path
and the transfer counter. */

declare variable COUNTADJUSTMENT;
/* Add the value of this variable whenever writing to the TransferCount */
/* register to set the number of transfers or post-stop-trigger transfers. */

/* Make adjustments based on triggers. */
if (using internal requests)
{
    COUNTADJUSTMENT = 0;
    Prestart = 0;
}
if (using external requests)
{
    if (group is input and StartSource = 0 and StopSource = 0)
        COUNTADJUSTMENT = 0
    else
        COUNTADJUSTMENT = -1
    if (StartSource = 3 or (StartSource > 0 and group is input))
        Prestart = 1;
    else
        Prestart = 0;
}
```

3.5 Bitfield Descriptions

This section describes bitfields details and locations for most bitfields used for two-way handshaking I/O. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate other bitfields not documented in this section. Bitfield descriptions of interrupt-related flags and flag-clearing bits are documented in Chapter 7, *Interrupt Controls*. Table B-1 in Appendix B also contains information on all DAQ-DIO registers.

DataLeft

bit:	**type:**	**register:**	**address:**
<0> (Group 1) <4> (Group 2)	Read	Group_Status	5

This flag indicates that one or more of the group's FIFO contains data. In funneling mode, it is possible to have data left at the end of a transfer, but not enough data to do another full transfer. The programmer can use this flag to help detect this condition.

ExchangePins

bit:	**type:**	**register:**	**address:**
<7>		Protocol_Register_7	

FIFOEnables

bits:	**type:**	**register:**	**address:**
<0..3>	Write	Data_Path	64 (Group 1)
			96 (Group 2)

Setting these bits assigns the corresponding FIFOs (and ports, unless funneling is in effect) to the group. Do not enable a single FIFO in both groups.

Bit 0:	Enable FIFO A
Bit 1:	Enable FIFO B
Bit 2:	Enable FIFO C
Bit 3:	Enable FIFO D

FIFOReset

bit:	**type:**	**register:**	**address:**
<7>	Write	Group_1_First_Clear	6 (Group 1)
		Group_2_First_Clear	7 (Group 2)

Write this bit to empty and reset the FIFOs.

Funneling

bits:	**type:**	**register:**	**address:**
<4..5>	Write	Data_Path	64 (Group 1)
			96 (Group 2)

These bits configure funneling, which allows software to modify the routing of FIFOs to ports.

0:	No funneling. Normal operation.
2:	8-bit funneling. Only port A (group 1) or port C (group 2) is used for I/O. However, transfers take place between the designated ports and each of the group's enabled FIFOs, in succession.

GroupDirection

bit:	**type:**	**register:**	**address:**
<7>	Write	Data_Path	64 (Group 1)
			96 (Group 2)

This bit specifies the group direction.

0:	Input
1:	Output

You must also configure the pin directions for each port used to match the group direction.

GroupFIFO

bits:	**type:**	**register:**	**address:**
<0..31>	Read/Write	Group_1_FIFOs	8(LSB)-11(MSB) (Group 1)
		Group_2_FIFOs	12(LSB)-15(MSB) (Group 2)

Writing or reading these bits accesses the FIFOs enabled for the group. The width of your read or write (8-bit, 16-bit, or 32-bit) must match the programmed TransferWidth.

Numbered

bit:	**type:**	**register:**	**address:**
<3>	Write	Protocol_Register_1	65 (Group 1) 97 (Group 2)

Setting this bit enables the handshaking numbered mode, which causes the group to handshake only for the number of transfers specified by the TransferCount bitfield.

ReadyLevel

bits:	**type:**	**register:**	**address:**
<0..2>	Write	FIFO_Control	72 (Group 1) 104 (Group 2)

RunMode

bits:	**type:**	**register:**	**address:**
<0..2>	Write	Protocol_Register_1	65 (Group 1) 97 (Group 2)

These bits enable and reset the handshaking process.

7:	run
0:	reset

TransferCount

bits:	**type:**	**register:**	**address:**
<0..31>	Write	Group_1_Transfer_Count	20 (LSB)-23 (MSB) (Group 1)
		Group_2_Transfer_Count	24 (LSB)-27 (MSB) (Group 2)

These bits specify the number of transfers you want to perform when using numbered mode.

TransferReady (DMAReq)

bit:	**type:**	**register:**	**address:**
<0>	Read	Group1_Flags	6 (Group 1)
		Group2_Flags	7 (Group 2)

This flag indicates that a group is ready for a data read or write. If DMA is enabled, this bit indicates a group is requesting DMA service. Operation of this bit is controlled by the Transfer_Size_Control Register.

TransferWidth

bits:	**type:**	**register:**	**address:**
<0..1>	Write	Transfer_Size_Control	77 (Group 1)
			109 (Group 2)

These bits indicate the transfer width. Each DMA cycle or FIFO read or write performed should match the specified width.

0 or 1:	32-bit DMA or group-FIFO transfers.
2:	8-bit DMA or group-FIFO transfers. However, transfers take place between these pins and each of the group's FIFOs in succession.
3:	16-bit DMA or group-FIFO transfers. Transfers take place to or from all of the group's FIFOs, two at a time, in succession. A group performing 16-bit DMA must contain two or four FIFOs.

Protocol Configuration Registers:

Initially, configure these registers according to the handshaking protocol you select (see Table 3-2). In some cases, as described in this chapter, you must later change the values of some bitfields in these registers. For example, you must change the RunMode bitfield in the Op_Mode Register from its initial value to begin a transfer.

ProtocolRegister1 (OpMode)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		65 (Group 1)
			97 (Group 2)

ProtocolRegister2 (Startup & Clocking)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		66 (Group 1)
			98 (Group 2)

ProtocolRegister3 (Sequence)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		67 (Group 1)
			99 (Group 2)

ProtocolRegister4 (REQ Management)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		70 (Group 1) 102 (Group 2)

ProtocolRegister5 (Block Mode and Tristate Control)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		71 (Group 1) 103 (Group 2)

ProtocolRegister6 (LinePolarities)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		73 (Group 1) 105 (Group 2)

ProtocolRegister7 (Serial Mode and AckLine)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		74 (Group 1) 106 (Group 2)

ProtocolRegister8 (StartDelay)

bits:	**type:**	**register:**	**address:**
<0..31>	Write		88 - 91 (Group 1) 120 - 123 (Group 2)

The most significant bits are located in the highest address and least significant bits are located in the lowest address. Therefore, if you write a 1 according to the Table 3-2, it should be written to address 88 for group 1 and address 120 for group 2.

ProtocolRegister9 (ReqDelay)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		82 (Group 1) 114 (Group 2)

ProtocolRegister10 (REQ* Delay)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		83 (Group 1) 115 (Group 2)

ProtocolRegister11 (AckDelay)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		84 (Group 1) 116 (Group 2)

ProtocolRegister12 (ACK* Delay)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		85(Group 1) 117 (Group 2)

ProtocolRegister13 (Data1Delay)

bits:	**type:**	**register:**	**address:**
<0..7>	Write		86 (Group 1) 118 (Group 2)

ProtocolRegister14 (ClockSpeed)

bits:	**type:**	**register:**	**address:**
<0..5>	Write		68 (LSB)-69 (MSB) (Group 1) 100 (LSB)-101 (MSB) (Group 2)

Write to this register in a single 16-bit write, or else write the LSB first.

ProtocolRegister15 (DAQOptions)

bits:	**type:**	**register:**	**address:**
	Write		79 (Group 1) 111 (Group 2)

Chapter 4 Pattern Generation

This chapter describes additional features and bitfields, in addition to those in Chapters 2 and 3, needed to implement pattern generation. This chapter includes procedures for implementing digital data acquisition and digital waveform generation, selecting internal or external timing, configuring interval counters, and establishing start and stop triggers.

4.1 Overview

Pattern generation is open-loop strobed operation that uses only a REQ signal, with no confirming ACK signal. Pattern generation includes both data acquisition (input) and waveform generation (output). The REQ signal can be provided externally, or generated internally from the DAQ-DIO's interval counters. Pattern generation also refers to single-direction, open-loop timing—usually periodic, fixed-rate input or output. You can control timing either internally or externally; that is, either the DAQ-DIO or the peripheral can regulate the transfer to ensure precise intervals. The other handshaking signals, if any, are ignored.

The DAQ-DIO samples or drives data on every active-going edge of the REQ signal. If the DAQ-DIO regulates the timing, you can program the rate at which an REQ signal is generated.

4.2 Features

The DAQ-DIO includes the following basic feature for pattern generation:

- Precise transfer timing
- Internal FIFOs with overflow/underflow flags
- Software or hardware start and stop triggers
- Dual-channel DMA
- Interrupts

Chapters 6 and 7describe how to use DMA channels and interrupts.

4.2.1 Protocol Registers

In internal-request pattern generation, the protocol registers control the request interval and pulse width. Pattern generation uses all of the ?????.

4.2.2 Overflow/Underflow (Waited) Flag

The waited flag serves as a FIFO overflow error flag in input mode, or an underflow error flag for output, indicating that the transfer had to wait because the CPU or DMA controller did not keep up with the programmed transfer rate.

4.2.3 Starting and Stopping Trigger Sources

There are several triggers you can use to start and stop a pattern generation operation:

- No trigger
- Hardware trigger
- Pattern detection trigger

With no trigger, the transfer starts when you set the RunMode bitfield to run—assuming, if in numbered mode, that you have already set the TransferCount. The transfer ends when you set RunMode back to reset or, in numbered mode, when the count expires.

With a hardware start trigger, you still must set RunMode to run after, in numbered mode, setting a TransferCount. However, the start of the transfer is delayed until a specified edge occurs on the group’s ACK (STARTTRIG) line.

With a hardware stop trigger, the transfer runs continuously until a specified edge occurs on the group’s STOPTRIG line. After the edge on the STOPTRIG line, the transfer continues to run for an additional number of points that correlates to the programmed TransferCount. This allows you to do both pretrigger and posttrigger acquisition around the STOPTRIG edge. Depending on the mode, you may need to adjust the TransferCount by one point to account for trigger synchronization.

With pattern detection you can control the lines to compare and the bit pattern for which to search on a port-by-port and pin-by-pin basis. Each group has only one pattern detection circuit, so you cannot use pattern detection for both start and stop triggering at the same time. You can also control the polarity of each trigger. For example, you can search for a rising or falling edge hardware trigger, and a pattern match or pattern mismatch pattern search.

4.3 Pin Interface

The following table describes the specific use of some pin signals in pattern generation. For the pin signals of the four I/O ports, refer to Section 1.3, *Bus Interface*.

Table 4-1. Pattern Generation Pins

Pin Name	Type	Use in Data Acquisition and Pattern Generation
ACK1 (STARTTRIG1)	Input, 24mA	Group 1 acknowledge—For pattern generation, this line can carry the start trigger signal for group 1.
ACK2 (STARTTRIG2)	Input, 24mA	Group 2 acknowledge—For pattern generation, this line can carry the start trigger signal for group 2.
REQ1	I/O, 24mA	Group 1 request line—For pattern generation, this line can serve as either internal or external data request for group 1.
REQ2	I/O, 24mA	Group 2 request line—For pattern generation, this line can serve as either internal or external data request for group 2.
STOPTRIG1	Input	Group 1 stop trigger line—For pattern generation, this line can serve as the stop trigger for group 1.
STOPTRIG2	Input	Group 2 stop trigger line—For pattern generation, this line can serve as the stop trigger for group 2.

4.4 Programming Information

The basic programming considerations and programming steps are the same as those of the two-way handshaking mode described in 3, *Handshaking*. The differences in configuration are summarized in Table 4-1.

Most of the bitfields used in the following pseudocode procedures are described in Section 4.5, *Bitfield Descriptions*. Refer to the *Bitfield Locations* in Appendix B, *Register Information*, to locate bitfield descriptions not documented in this chapter. To locate the function calls in this manual, refer to the *Index*. For general programming considerations, instruction notation, and window programming information, refer to 1, *Introduction*.

Data Acquisition and Pattern Generation Pseudocode

```
{
    call PG_Config;
    call PG_Run;
}
```

4.4.1 Configuring the Pattern Generation Mode

Function PG_Config

```
{
    Σ
    RunMode = 0;
    call Group_Config;      /* Call this function as explained in  3, Handshaking,
                            to configure one of the groups as output to perform waveform
                            generation or as input to perform data acquisition. Also
                            enable the FIFO or FIFOs you want to use. */
    configure suitable protocol according to Table 4-2;

    StartSource = 0 (software triggering), 1, 2, or, for input groups, 3 (pattern match for input
    groups only);

    if (use StartSource = 0)
        {
            if ((use PulseBurst mode) and (group is input))
                Prestart = 1;
            InvertStart = 1;
        }
    if (using StartSource = 3)
        {
            call Pattern_Detection_Config; /* See  5, Change Detection and Pattern
            Detection, for this function and details on pattern detection */
            if (use DetectionMethod = 1) PreStart = 1;
        }
    StopSource = 0 or 1 or 2 or 3;

    if (StopSource = 1) and (using active low hardware stop trigger)
        InvertHWStop = 1;

    if (group is output) call Preload_FIFOs; /* Call this function as explained in  3,
    Handshaking, to preload data for output groups */

}
```

The following table summarizes the four types of protocols:

Table 4-2. Pattern Generation Parameters

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Protocol\Protocol Register	**OpMode**	**ClockReg**	**Sequence**	**ReqReg**	**BlockMode**	**LinePolarities**	**AckSer**	**StartDelay**	**ReqDelay**	**ReqNotDelay**	**AckDelay**	**AckNotDelay**	**Data1Delay**	**ClockSpeed**	**DAQOptions**
Input with internal REQs	0	0	1	4	0 [1]	3	224	10 x int-1	1	1	1	11	1	—	0 [4]
Output with internal REQs	0	0	1	4	0 [1]	3	224	10 x int-1	1	1	1	11	1	—	0 [4]
Input with external REQs	0	0	0	0	0 [2]	0	0	1	1	1	1	12	16	0	96
Output with external REQs	0	0	0	0	0 [2]	0	0	1	1	1	1	12	16	0	96

[1] Set to 0; if performing 8:16 funneling, set to 24.

[2] Set to 4; if performing 8:16 funneling, set to 0.

[3] Set to 10 times the pattern generation interval minus one, unless timebase is 20 MHz in which case set to the pattern generation interval minus one.

[4] Set according to pattern generation timebase: 0 if 20 MHz, 1 if 10 MHz, 10 if 100 KHz, 100 if 10 KHz., 1000 if 1KHz, or 10000 if 100 Hz.

4.4.2 Running Pattern Generation Mode

Function PG_Run

```
{
    if (using numbered mode or a stop trigger)
    {
        Numbered = 1;
        TransferCount = number of transfer or post-stop-trigger transfers to perform;
    }
    Σ
    RunMode = 7;                          /* call this function as explained in Chapter 3,
                                          Handshaking, to perform a transfer */
    call Perform_Transfer;
    wait until stop trigger is detected   /* if using a stop trigger, you must acquire
                                          data continuously */;
    RunMode = 0;
    call Cleanup;
    Σ
}
```

4.5 Bitfield Descriptions

This section describes details and locations for most bitfields used in pattern generation. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate any bitfields not documented in this section. Table B-1 in Appendix B also contains information on all DAQ-DIO registers.

InvertStart

bit:	**type:**	**register:**	**address:**
2	Write	Protocol_Register_15	79 (Group 1)
			111 (Group 2)

This bit inverts the polarity of the start signal and can be changed even when RunMode = 7. This bit is useful for software start triggering.

InvertStop

bit:	**type:**	**register:**	**address:**
<7>	Write	Protocol_Register_2	66 (Group 1)
			98 (Group 2)

Set this bit to configure the hardware stop trigger (STOPTRIG) line to be active low.

Prestart

bit:	**type:**	**register:**	**address:**
<7>	Write	Protocol_Register_15	79 (Group 1)
			111 (Group 2)

This bit is used to control data synchronization for certain start trigger modes. Set if using external requests, and StartSource = 3.

ReqSource

bit:	**type:**	**register:**	**address:**
<6>	Write	Protocol_Register_15	79 (Group 1)
			111 (Group 2)

This bit specifies the source of the request pulses.

0: internal
1: external

StartSource

bits:	**type:**	**register:**	**address:**
<0..1>	Write	Protocol_Register_15	79 (Group 1)
			111 (Group 2)

These bits specify the start trigger for data acquisition and pattern generation.

0: Software start trigger. Trigger is high as soon as RunMode = > unless InvertStart is set.
1: An active-going edge of the ACK (STARTTRIG) line.
2: An active level on the ACK (STARTTRIG) starts the transfer line.
3: (Input groups only) Pattern match. See 5, *Change Detection and Pattern Detection*.

StopSource

bits:	**type:**	**register:**	**address:**
<3..4>	Write	Protocol_Register_15	79 (Group 1)
			111 (Group 2)

These bits specify the source of the stop trigger for data acquisition of pattern generation.

0: No stop trigger—the trigger for sending the transfer occurs immediately when RunMode is set to 7, unless InvertStop is set.
1: An active-going edge of the STOPTRIG line stops the transfer.
2: A high level on the STOPTRIG line stops the transfer.
3: (Input groups only) Pattern match. See 5, *Change Detection and Pattern Detection*

Change Detection and Pattern Detection

Chapter 5

This chapter describes features and bitfields for implementing change detection (transition sensing) and pattern detection (triggering on a specific input pattern).

5.1 Overview

For most configurations, the DAQ-DIO control logic ignores the data being transferred. However, in two input modes, change detection mode and pattern detection mode, the control logic monitors the acquired input data. These modes are types of pattern generation input.

In change detection, a handshaking group generates its own request whenever the current data on the lines does not match the last latched data.

In pattern detection, when the specified pattern appears the group generates a data acquisition start or stop trigger. You can choose either to compare the input pins directly to the pattern all times (usually used for start trigger) or to compare the pattern only at the sample times (usually used for stop triggers).

5.2 Features

The DAQ-DIO includes the following basic features for change detection and pattern detection:

- One change-detection circuit per group
- One pattern-detection circuit per group
- Mask registers that allow you to ignore specified bits when searching for a data change or pattern
- Software for pattern detection that controls the pattern and selects polarity (search for match or search for mismatch)
- Pattern detection can serve as a start or stop trigger for data acquisition using pattern detection

5.2.1 Pin Masks

For both change detection and pattern detection, you can mask certain bits as insignificant in a detection by setting the corresponding bits in the Pin_Mask Register.

Note: ***This is the same register that, for output pins, selects the drive type.***

5.2.2 Pattern Programming

You can select the pattern to detect at a certain port by writing to the Port_Pattern registers, so that you can detect any combination of high and low lines.

5.3 Programming Information

This section contains programming details for configuring change detection mode and pattern detection mode. Apart from this configuration, the programming procedure is the same as for standard pattern generation, as described in Chapter 4, *Pattern Generation*.

Most of the bitfields used in the following pseudocode procedures are described in Section 5.4, *Bitfield Descriptions*. Refer to the *Bitfield Locations* in Appendix B, *Register Information*, to locate bitfield descriptions not documented in this chapter. To locate the function calls in this manual, refer to the *Index*. For explanation on the general programming considerations, instruction notation, and window programming information, refer to Chapter 1, *Introduction*.

5.3.1 Pseudocode Example for Change Detection

Change detection mode is similar to normal pattern generation mode. It differs only in that the request signal is internally generated when a change in the input data is detected.

To enable change detection, set the protocol registers to the values in Table 5-1.

Change Detection Configuration Pseudocode

```
{
    Σ
    RunMode = 0;
    call Group_Config;
    Σ
    call Change_Detection_Config;
}
```

Function Change_Detection_Config

Table 5-1. Change Detection Parameters

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Protocol\Protocol Register	**OpMode**	**ClockReg**	**Sequence**	**ReqReg**	**BlockMode**	**LinePol**	**AckSer**	**StartDelay**	**ReqDelay**	**ReqNotDelay**	**AckDelay**	**AckNotDelay**	**Data1Delay1**	**ClockSpeed**	**DAQ Options**
Change Detection	0	1	17	3	0 [1]	0	224	1	2	1	1	2	1	0	0

[1] Set to 0; if performing 8:16 funneling, set to 24.

5.3.2 Programming Steps and Pseudocode Example for Pattern Detection

Use the following steps to program pattern detection:

1. Choose a group and enable the corresponding ports you want to use. Configure a handshaking protocol.
2. Choose one of the two Detection Methods. Detection Method = 0 for start triggers or 1 for stop triggers is recommended.
3. Set the pattern to detect in the Port_Pattern Register of each port. Mask out any pins to be ignored in detection. Set InvertMatched, if necessary.
4. If you use pattern detection to start data acquisition, set StartSource = 3. If you use 1 as Detection Method, you also need to set the Prestart bit. Refer to Chapter 4, *Pattern Generation*, for more information on configuring start and stop triggers.
5. Begin and end the transfer as described in Chapter 3, *Handshaking*.
6. If another transfer is needed, clean up as described in Chapter 3 before reconfiguring groups.

Pattern Detection Configuration Pseudocode

```
{
    call Handshaking_Config;
    call Pattern_Detection_Config;
}
```

Function Pattern_Detection_Config

```
{
    PinMask = pins to be ignored by change detection if any, for each port in the group;
    Detection Method = 0 or 1; /* See bitfield descriptions */
    PinPattern = pattern you want to detect, for each port in the group; the pattern bit for any
    ignored pin must be zero;
}
```

5.4 Bitfield Descriptions

This section describes details and locations for most bitfields used for change detection and pattern detection. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate any bitfields not documented in this section. Table B-1, in Appendix B also contains information on all DAQ-DIO registers.

DetectionMethod

bit:	**type:**	**register:**	**address:**
<0>	Write	Port_A_Pattern	81 (Group 1)
			113 (Group 2)

This bit specifies the detection method used for pattern detection.

0: Compare the input pins directly to the pattern at all times.

1: Compare acquired data to the pattern. A pattern is ignored unless it occurs with a request strobe.

PinPattern

bits:	**type:**	**register:**	**address:**
<0..7>	Write	Port_A_Pattern	48 (Port A)
		Port_B_Pattern	49 (Port B)
		Port_C_Pattern	50 (Port C)
		Port_D_Pattern	51 (Port D)

These bits specify the pattern you want to detect at a specific port. For any insignificant bit (bit for which the corresponding PinMask bit is set), be sure to put a 0 in the PinPattern.

InvertMatch

bit:	**type:**	**register:**	**address:**
<1>	Write	Pattern_Detection	81 (Group 1)
			113 (Group 2)

If this bit is set, the PatternMatched flag is asserted when the unmasked data lines do not equal the input pattern.

Pattern matching, if used as a start or stop trigger, is also affected by the InvertStart or InvertStop bit.

Chapter 6

DMA Controls

This chapter describes the pin interfaces and programming considerations on using the DAQ-DIO with Direct Memory Access (DMA).

6.1 Overview

Direct memory access (DMA) enables the DAQ-DIO to move data directly to or from system memory, without using the CPU. DMA can greatly reduce the burden that digital I/O places on your CPU while also increasing the speed of data transfer.

The DAQ-DIO supports up to two DMA channels, if provided by your board or system. If you have two DMA channels available, you can allocate them as you wish: one channel to each group, or two channels to a single group. Allocating both channels to one group enables the DAQ-DIO to support dual DMA, the process of switching back and forth between channels during a single operation. Dual DMA can improve performance on cards, such as the AT-DIO-32HS, that do not access to linking or chaining DMA.

To perform DMA, the DAQ-DIO must work with a DMA controller. For example, the PCI-DIO-32HS and PXI-6433 contain linking and chaining DMA controller onboard within the National Instruments PCI MITE ASIC. The AT-DIO-32HS uses the system DMA controller. The DAQCard-6533 does not offer DMA.

When the FIFOs are ready for a transfer—that is, when they contain input data or space for output data—the DAQ-DIO asserts a DMA request (DRQ) to a DMA controller. The DMA controller responds by asserting DMA acknowledge (DACK) and performing a read or write.

To program DMA, you must program both the DAQ-DIO and the DMA controller. This manual describes only the programming of the DAQ-DIO.

6.2 Features

The DAQ-DIO includes the following basic features for DMA:

- Support for two DMA channels
- Software controlled 8-bit, 16-bit, or 32-bit DMA width

6.3 Pin Interface

The following table describes the specific use of some pin signals in DMA. For the pin signals of the four I/O ports, refer to Section 1.3, *Bus Interface*.

Table 6-1. DMA Pins

Pin Name	Type	Use in DMA
DRQ <1..2>	Output, 4 mA	DMA Request line — This line drives high when requesting DMA and drives low when not requesting DMA.
DACK* <1..2>	Input, pulled low	DMA Acknowledge line — These active low request signals cause a read or write to take place from FIFO data; the address lines are ignored when a DMA cycle occurs.
TC	Input, pulled low	DMA terminal count signal — A signal indicating DMA transfer is completed, in ISA-type DMA. When this signal occurs with DMAAck1 or DMAAck2, the terminal count flag will be set, and an interrupt can be generated. This signal is not used for PCI-type (MITE) DMA.

6.4 Programming Information

This section contains programming details for configuring and running DMA transfers. Most of the bitfields used in the following pseudocode procedures are described in Section 6.5, *Bitfield Descriptions and Values Used in DMA*. Refer to *Bitfield Locations* in Appendix B, *Register Information*, to locate bitfield descriptions not documented in this section. To locate the function calls in this manual, refer to the *Index*. For general programming considerations, instruction notation, and window programming information, refer to Chapter 1, *Introduction*.

6.4.1 Programming Steps and Pseudocode Example for DMA

This section contains the programming steps and a sample program for bit-level programmers to configure and perform the DMA transfers.

Use the following steps for programming DMA transfers:

1. Configure a group and ports for the group.
2. Program a protocol.
3. Set the DMA channel to use and set the DMA TransferWidth, TransferLength, and RequireRLevel.
4. Write and enable service routines to handle the TC interrupt from the DAQ-DIO, or, if available, an end-of-operation interrupt from your DMA controller. Refer to Chapter 7, *Interrupt Controls*, for information on interrupts.
5. Program the DMA controller.
6. Start DMA transfer, either with or without numbered mode.
7. For numbered mode, wait for a flag or interrupt that indicates the transfer is done. Wait for the done flag from your DMA controller or the DMA TC flag in the case of input or the CountExpired flag in the case of output. If numbered mode is not used, you can end the transfer just like other handshaking transfers, as described in Chapter 3, *Handshaking*.
8. Stop the operation as described in Chapter 3.

DMA Pseudocode

```
{
    call Handshaking_Config;
    call DMA_Config;
    call DMA_Run;
    DMAReset = 1;
    call Cleanup;
}
```

6.4.2 Configuring DMA

You can choose which channel is used by a certain group by programming the DMA_Line_Control Register. In addition to the bitfields TransferWidth, TransferOrder, and ReadyLevel, which are needed for normal handshaking, you also need to configure the group TransferLength and RequireRLevel in the Transfer_Size_Control Register.

The channel referred to here is the DAQ-DIO DMA channel—either 1 or 2. The mapping of DAQ-DIO DMA channels to board or system DMA channels is a function of your board or system. For more information, refer to ?

Function DMA_Config

```
{
    DMAChannel = desired DMA channel to use;
    TransferLength = 0 (no limit) or 1 (4 transfers) or 2 (8 transfers) or 3 (16 transfers);
    RequireRLevel = 0 or 1;
}
```

6.4.3 Running Handshaking with DMA

Function DMA_Run

```
/* Performing the full-handshaking transfer using the FIFOs */
{
    Σ
    if (using numbered mode)
    {
        Numbered = 1;
        TransferCounts = number of points to transfer;
    }

    ClearPrimaryTC and ClearSecondaryTC = 1;
    RunMode = 7;
    Σ
    if (group is output)
    {
        for i = 1 to number written to transferCount minus number of preloaded data if
        numbered mode is used
            DMA write data (8-bit, 16-bit, or 32-bit, depending on TransferWidth);
    }

    if (group is input)
    {
        for i = 1 to desired number of transfers, or number written to transferCount if number
        mode used
            DMA read data (8-bit, 16-bit, or 32-bit, depending on TransferWidth);
    }

/* Stop the handshaking transfers gracefully. */
    if (group is output)
        {
            if (use numbered mode) wait until flag CountExpired = 1;
            else wait until Waited flag = 1;
        }
    if (group is input)
    {
            if (use numbered mode)
                wait until flag PrimaryTC or SecondaryTC (depend of the channel
                used) = 1;
    }

    Σ
    RunMode = 0; /* Stop the group's handshaking operations */
    wait until EndOfCycle flag = 1;
    Σ

}
```

6.5 Bitfield Descriptions and Values Used in DMA

This section describes details and locations for most bitfields used in DMA. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate any bitfields that are not documented in this section. Table B-1 in Appendix B also contains information on all registers in the DAQ-DIO.

DMAChannel

bit:	**type:**	**register:**	**address:**
<0..3>	Write	DMA_Line_Control	76 (Group 1) 108 (Group 2)

These bits specify which DMA channel to use. DMA begins on the primary channel and switches between the primary and secondary channels each time the DAQ-DIO receives a terminal count (TC). To use only one channel, program the same value for both primary and secondary DMA channels. Mapping from the DMA channels to the board or system DMA channels depends on the board or system.

<0..1>: Primary DMA Channel (1 to 2), or 0 for none.
<2..3>: Secondary DMA Channel (1 to 2), or 0 for none.

DMAReset

bit:	**type:**	**register:**	**address:**
6	Write	Group_1_First_Clear Group_2_First_Clear	6 7

Write this bit to reset the group’s DMA process.

RequireRLevel

bit:	**type:**	**register:**	**address:**
<5>	Write	Transfer_Size_Control	77 (group 1) 109 (group 2)

This bit determines how long DMAReq is asserted. Set this bit for output DMA transfers, not input transfers or non-DMA transfers.

TransferLength

bits:	**type:**	**register:**	**address:**
<3..4>	Write	Transfer_Size_Control	77 (group 1) 109 (group 2)

These bits determine the maximum TransferReady (DRQ) duration, which is useful for DMA on some platforms. The DAQ-DIO provides a time-out mechanism, which is required by certain DMA controllers to prevent the DMA process from monopolizing the bus. When the time-out expires, the DMAReq is held low until DMAAck deactivates.

0:	No limit. Recommended for MITE-type (PCI) DMA.
1:	Maximum of 4 transfers.
2:	Maximum of 8 transfers.
3:	Maximum of 16 transfers. Recommended for ISA-type DMA.

Chapter
7

Interrupt Controls

This chapter describes the DAQ-DIO interrupt control and explains its features, the different conditions that can trigger interrupts, and the programming of a typical interrupt service routine.

7.1 Overview

The DAQ-DIO can increase bus efficiency by using an interrupt channel. You can use an interrupt channel for event notification without the use of polling techniques. The DAQ-DIO has one interrupt output line. You can program both of the strobed I/O groups to generate interrupt signals to the interrupt output line when certain conditions are satisfied.

7.2 Features

The DAQ-DIO has the following basic features for interrupts:

- One software controllable interrupt line
- A master interrupt enable bit
- Individual interrupt enable bits for all possible interrupt sources

7.2.1 Interrupt Sources

The Interrupt_Control Register for each handshaking group enables interrupt triggering by different interrupt sources. An interrupt can be generated when one of the following flags and the corresponding enable bit are set:

- TransferReady
- CountExpired
- Waited (underflow/overflow for pattern generation)
- PrimaryTC
- SecondaryTC

7.2.2 Interrupt Line

The Master_Interrupt_Control Register configures the interrupt line. You can choose to tristate, invert, or make the interrupt line open-collector type output. Both groups use the same line to trigger an interrupt channel. The interrupt service routine should check the InterruptStatus flags and Group flags to determine which group and which one of the above sources causes the interrupt.

7.3 Pin Interfaces

The following table describes the specific use of some pin signals in interrupts. For the pin signals of the four I/O ports, refer to Section 1.3, *Bus Interface*.

Table 7-1. Interrupt Pins

Pin Name	Type	Descriptions
InterruptLine	Output, 4mA	Interrupt line — This line generates the IRQ signal to the CPU. It is tristated at power-up.

7.4 Programming Information

This section contains detailed programming information to do bit-level programming of the interrupt interface for specialized applications. Most of the bitfields used in the following pseudocode procedures are described in Section 7.5, *Bitfield Descriptions*. Refer to *Bitfield Locations* in Appendix B, *Register Information*, to locate bitfield descriptions not documented in this chapter. To locate the function calls in this manual, refer to the *Index*. For general programming considerations, instruction notation, and window programming information, refer to Chapter 1, *Introduction*.

7.4.1 Programming Steps and Psuedocode Example for Interrupts

This section contains the programming steps and a sample program for bit-level programmers to configure and enable interrupts.

Use the following steps to enable interrupts:

1. Configure groups and ports before enabling any interrupts.
2. Choose interrupt sources and set the corresponding bits in the Interrupt_Control Register of one or both groups.
3. Begin the interrupt by setting the Master_Interrupt_Control Register to enable the Interrupt Line.

Interrupt Enable Pseudocode

```
{
    Σ
    call Group_Config;
    set IntEnable(0-7) of one or both groups to desired interrupt conditions;
    Σ
    OpenInt = 0 (default) or 1 (open-collector-type output);
    InvertInt = 0 (default, active high) or 1 (invert, active low);
    InterruptLine = 3;      /* Enable interrupts globally */
}
```

7.4.2 Programming Steps and Psuedocode Example for Interrupt Service Routine

This section contains the programming steps and a sample program for bit-level programmers to configure and perform the interrupt service routine.

Use the following steps to program the interrupt service routine:

1. Read the IntStatus flag to determine which group generates the interrupt.
2. Read the corresponding GroupStatus flag to determine which sources of the group triggers the interrupt.
3. Clear the flags. SerialRose, ReqRose, Paused, Waited, PrimaryTC, SecondaryTC flags can be cleared by writing to the Clear Register. TransferReady can be cleared by reading or writing the group FIFOs. ExpiredCount can be cleared by
4. Perform the desired work.
5. Read IntStatus again to make sure that all interrupts are taken care of before exiting an interrupt service routine.

Interrupt Service Routine Pseudocode

```
{
    declare variable STATUS
    STATUS = IntStatus;
    if (bit 0 or bit 1 of STATUS set)
    do
        {
            if (bit 0 of STATUS set)
                call Service(1);
            else
                call Service(2);
            STATUS = IntStatus;
        }
    while (bit 0 or bitt 1 of STATUS set);
}
```

Function Int_Service(variable GROUP)

```
{
      declare variable FLAGS;
      if (Group = 1)
            FLAGS = Group_1_Flags;
      else
            FLAGS = Group_2_Flags;
      if (bit 0 of IntEnables = 1 and bit 0 of FLAGS = 1)/* TransferReady causes the
                                                               interrupt */
            {
                  service the TransferReady Interrupt;      /* You have to read (input) or write
                                                               (output) group FIFOs to clear this
                                                               interrupt condition. */
            }

      if (bit 1 of IntEnables - 1 and bit 1 of FLAGS - 1)  /* CountExpired causes the
                                                               interrupt */
            {
                  ClearExpired = 1;
                  service the CountExpired Interrupt;
            }

      /* It is usually unnecessary to interrupt on underflow/overflow. In cases where waiting
      is an error (pattern generation), you can simply check the waited flag at the end of the
      transfer. */

      if (bit 5 of IntEnables = 1 and bit 5 of FLAG = 1)/* Waited causes the interrupt */
            {
                  ClearWaited = 1;
                  service the Waited Interrupt;
            }

      if (bit 6 of IntEnables = 1 and bit 6 of FLAGS = 1)/* PrimaryTC causes the interrupt */
            {
                  ClearPrimaryTC = 1;
                  service the PrimaryTC Interrupt;
            }

      if (bit 7 of IntEnables = 1 and bit 7 of FLAGS = 1 /* SecondaryTC causes the
                                                               interrupt */
            {
                  ClearSecondaryTC = 1;
                  service the SecondaryTC Interrupt;
            }
}
```

7.5 Bitfield Descriptions

This section describes details and locations for bitfields used for interrupt controls. Some bitfields in this section are organized under categories of clear registers and flags categories for easy reference. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate the any bitfields not documented in this section. Table B-1 in Appendix B also contains information on all the DAQ-DIO registers.

IntStatus

bit:	**type:**	**register:**	**address:**
0 (group 1)	Read	Interrupt_and_Window_Status	4
1 (group 2)			

Each bit indicates that the corresponding group is requesting interrupt service and that the interrupt request pin, if enabled, is active.

IntEnables

bit:	**type:**	**register:**	**address:**
<0..7>	Write	Interrupt_Control	75 (Group 1) 107 (Group 2)
<2>		Pattern_Detection	81 (Group 1) 113 (Group 2)

Setting these bits enables interrupt generation when the corresponding group flags are set.

In Interrupt_Control Register:

0: Interrupt on TransferReady.
1: Interrupt on CountExpired.
5: Interrupt on Waited.
6: Interrupt on PrimaryTC.
7: Interrupt on SecondaryTC.

In Pattern_Detection Register:

2: Interrupt on PatternDetected.

InterruptLine

bit:	**type:**	**register:**	**address:**
<0..1>	Write	Master_DMA_and_ Interrupt_Control	5

These bits determine the use of the interrupt line.

0: Tristate the interrupt line (power-up state).
1: Drive the interrupt line low (disable interrupts).
3: Enable the interrupt line for normal operation.

Clear Registers:

ClearExpired

bit:	**type:**	**register:**	**address:**
0	Write	Group_1_Second_Clear	46 (Group 1)
		Group_2_Second_Clear	47 (Group 2)

Write this bit to clear the CountExpired flag.

ClearPrimaryTC

bit:	**type:**	**register:**	**address:**
4	Write	Group_1_First_Clear	6(Group 1)
		Group_2_First_Clear	7 (Group 2)

Write this bit to clear the PrimaryTC flag.

ClearSecondaryTC

bit:	**type:**	**register:**	**address:**
5	Write	Group_1_First_Clear	6 (Group 1)
		Group_2_First_Clear	7 (Group 2)

Write this bit to clear the SecondaryTC flag.

ClearWaited

bit:	**type:**	**register:**	**address:**
3	Write	Group_1_First_Clear	6 (Group 1)
		Group_2_First_Clear	7 (Group 2)

Write this bit to clear the Waited flag.

Flags:

CountExpired

bit:	**type:**	**register:**	**address:**
<1>	Read	Group_1_Flags	6 (Group 1)
		Group_2_Flags	7 (Group 2)

This flag indicates that the TransferCount has expired (or has not begun) in numbered mode. If numbered mode is not used, ignore this flag.

PrimaryTC

bit:	**type:**	**register:**	**address:**
<6>	Read	Group_1_Flags	6 (Group 1)
		Group_2_Flags	7 (Group 2)

This flag indicates that a DMA terminal count has occurred on the primary group DMA channel.

SecondaryTC

bit:	**type:**	**register:**	**address:**
<7>	Read	Group_1_Flags	6 (Group 1)
		Group_2_Flags	7 (Group 2)

This flag indicates that a DMA terminal count has occurred on the secondary group DMA channel.

Waited

bit:	**type:**	**register:**	**address:**
<5>	Read	Group_1_Flags	6 (Group 1)
		Group_2_Flags	7 (Group 2)

This flag indicates that the strobed I/O logic had to stop and wait for the CPU or DMA controller. To clear this flag, set the ClearWait bit in the Clear Register.

This flag can be used to detect FIFO overflow (input) or underflow (output). In pattern generation applications, overflow or underflow may represent an error condition.

Chapter 8

RTSI Connection

This chapter describes the DAQ-DIO interface for communication with a RTSI bus or PXI trigger bus.

8.1 Overview

The DAQ-DIO can connect directly to a 20 MHz auxiliary clock line, such as a 20 MHz signal on a RTSI clock line or PXI trigger line. Indirectly, the DAQ-DIO can connect to a flexible trigger bus such as the ones provided by RTSI or PXI using a RTSI crossbar switch.

DAQ-DIO devices implement this connection in different ways and some devices, such as the DAQCard-6533, do not implement a RTSI connection at all. See your device user manual and register-level programming manual for more information.

8.2 Features

The DAQ-DIO has the following basic features for RTSI connection:

- An auxiliary 20 MHz clock line
- Address decoding and control for a RTSI crossbar switch

8.3 Pin Interfaces

The following table describes the specific use of some pin signals in RTSI connection. For the pin signals of the four I/O ports, refer to Section 1.3, *Bus Interface*.

Table 8-1. RTSI Connection Pins

Pin Name	Type	Description
RTSIClk	I/O, 8 mA	Auxiliary 20 MHz clock.
RTSIWt*	output, 4 mA	Write signal for a RTSI crossbar switch.

8.4 Programming Information

The DAQ-DIO decodes writes to offsets 124 through 127 and sends them to a RTSI crossbar switch, if implemented on your device. See your device user manual and the *DIO-6533 Register-Level Programmer Manual* for full RTSI information.

8.5 Bitfield Descriptions and Values Used for RTSI

This section describes details and locations for bitfields used for RTSI. Refer to the *Bitfield Locations* section in Appendix B, *Register Information*, to locate any bitfields not documented in this section. Table B-1 in Appendix B also contains information on all DAQ-DIO registers.

RTSIClocking

bits:	**type:**	**register:**	**address:** 45
<4..5>	Write	Master_Clock_Routing	

These bits determine the treatment of RTSIClock line, an alternate source for the master 20 MHz clock used to control strobed I/O.

0: Standalone: use the Oscillator line from an onboard oscillator as the master clock. Tristate the RTSIClock line.
2: Receive. Use the RTSIClock line as the master clock.
3: Drive. Use the Oscillator line as the master clock and drive it onto the RTSIClock line.

Appendix A

Specifications

This appendix lists the specifications for the DAQ-DIO. These specifications are typical at 25° C unless otherwise noted.

Digital I/O

Number of channels	32 input/output; 4 dedicated output and control; 4 dedicated input and status
Compatibility	TTL/CMOS (standard or wired-OR[1])
Hysteresis	500 mV

Digital logic levels

Level	Min	Max
Input low voltage	0 V	0.8 V
Input high voltage	2 V	5 V
Input leakage current digital I/O	—	10 µA
Output low voltage (I_{OL} = 24 mA)	—	0.4 V
Output high voltage (I_{OH} = 24 mA)*	2.4 V	—

* When configured as standard outputs. Drivers configured as wired-OR outputs are tri-stated when logically high.

Absolute max input voltage range	–0.3 to 5 V
Power-on state for outputs	Tri-stated

[1] As of NI-DAQ 5.1, LabVIEW does not support wired-OR outputs.

Data transfers	Programmed I/O, DMA (if supported by board and bus)

Strobed I/O

Pattern Generation

Direction	Input or output
Modes	Internally or externally timed
Sample rate (absolute maximum)[1]	10 MS/s
Sample rate (absolute maximum) [1]	20 MS/s
Sample rate (min internally timed)	1 S/10 min.
Sample rate (min externally timed)	No limit

Handshaking

Direction	Input or output
Modes	6 (burst, level-ACK, leading-edge pulse, trailing-edge pulse, long pulse, and 8255 emulation)
Transfer rate (absolute maximum for burst mode) [1]	20 MS/S at 32 bits (80 MB/S)

Pattern and Change Detection

Pattern-detection triggers	Start or stop on pattern
Pattern-detection resolution	60 ns or one REQ period, depending on mode
Change-detection function	Sample on change
Change-detection resolution	150 ns

[1] These are absolute maximums for the protocols described in this manual. Actual transfer rate depends on the board, system, and software. See your board's user manual for information.

Start and Stop Triggers

Compatibility	TTL/CMOS
Trigger types	Rising or falling edge, or digital pattern
Pulse width for edge triggers (min)	10 ns
Pattern triggers detection capabilities	Detect pattern match or mismatch on user-selected bits

Bus Interfaces

PCI-type interface	National Instruments PCI MITE for PCI master and target operations
AT-type interface	16-bit slave with dual DMA support

Environment

Operating temperature	0 to 55° C
Storage temperature	–20 to 70° C
Relative humidity	5% to 90% noncondensing

Register Information

Appendix
B

This chapter contains information about the registers in the DAQ-DIO.

Register Map

This section provides register map information for DAQ-DIO registers. Table B-1 lists the register map information by offset. Table B-2 lists the register map information by register name.

Table B-1. Register Map Listed by Offset

Offset (in decimal)	Write Register	Read Register
0-3	Window_Data	Window_Data
4	Window_Address	Interrupt_and_Window_Status
5	Master_DMA_and_Interrupt_Control	Group_Status
6	Group_1_First_Clear	Group_1_Flags
7	Group_2_First_Clear	Group_2_Flags
8–11	Group_1_FIFOs	Group_1_FIFOs
12–15	Group_2_FIFOs	Group_2_FIFOs
20	Transfer_Count (Group 1) byte 0, LSB	Reserved
21	Transfer_Count (Group 1) byte 1	Reserved
22	Transfer_Count (Group 1) byte 2	Reserved
23	Transfer_Count (Group 1) byte 3, MSB	Reserved
24	Transfer_Count (Group 2) byte 0, LSB	Chip ID: 'D'
25	Transfer_Count (Group 2) byte 1	Chip ID: 'I'
26	Transfer_Count (Group 2) byte 2	Chip ID: 'O'
27	Transfer_Count (Group 2) byte 3, MSB	Chip Version: 1
28	Port_A_Output	Port_A_Input

Table B-1. Register Map Listed by Offset (Continued)

Offset (in decimal)	Write Register	Read Register
29	Port_B_Output	Port_B_Input
30	Port_C_Output	Port_C_Input
31	Port_D_Output	Port_D_Input
32	Port_A_Pin_Directions	Reserved
33	Port_B_Pin_Directions	Reserved
34	Port_C_Pin_Directions	Reserved
35	Port_D_Pin_Directions	Reserved
36	Port_A_Pin_Mask	Reserved
37	Port_B_Pin_Mask	Reserved
38	Port_C_Pin_Mask	Reserved
39	Port_D_Pin_Mask	Reserved
40	Port_A_Polarities	Reserved
41	Port_B_Polarities	Reserved
42	Port_C_Polarities	Reserved
43	Port_D_Polarities	Reserved
45	Master_Clock_Routing	Reserved
46	Group_1_Second_Clear	Reserved
47	Group_2_Second_Clear	Reserved
48	Port_A_Pattern	Reserved
49	Port_B_Pattern	Reserved
50	Port_C_Pattern	Reserved
51	Port_D_Pattern	Reserved
49-52	Reserved	Reserved
56-63	Reserved for board-level circuitry	Reserved for board-level circuitry
64 (Group 1), 96 (Group 2)	Data_Path	Reserved

Table B-1. Register Map Listed by Offset (Continued)

Offset (in decimal)	Write Register	Read Register
65 (Group 1), 97 (Group 2)	Protocol_Register_1	Reserved
66 (Group 1), 98 (Group 2)	Protocol_Register_2	Reserved
67 (Group 1), 99 (Group 2)	Protocol_Register_3	Reserved
68-69 (Group 1), 100-101 (Group 2)	Protocol_Register_14	Reserved
70 (Group 1), 102 (Group 2)	Protocol_Register_4	Reserved
71 (Group 1), 103 (Group 2)	Protocol_Register_5	Reserved
72 (Group 1), 104 (Group 2)	FIFO_Control	Reserved
73 (Group 1), 105 (Group 2)	Protocol_Register_6	Reserved
74 (Group 1), 106 (Group 2)	Protocol_Register_7	Reserved
75 (Group 1), 107 (Group 2)	Interrupt_Control	Reserved
76 (Group 1), 108 (Group 2)	DMA_Line_Control	Reserved
77 (Group 1), 109 (Group 2)	Transfer_Size_Control	Reserved
78 (Group 1), 110 (Group 2)	Reserved	Reserved
79 (Group 1), 111 (Group 2)	Protocol_Register_15	Reserved
80 (Group 1), 112 (Group 2)	Reserved	Reserved
81 (Group 1), 113 (Group 2)	Pattern_Detection	Reserved

Table B-1. Register Map Listed by Offset (Continued)

Offset (in decimal)	Write Register	Read Register
82 (Group 1), 114 (Group 2)	Protocol_Register_9	Reserved
83 (Group 1), 115 (Group 2)	Protocol_Register_10	Reserved
84 (Group 1), 116 (Group 2)	Protocol_Register_11	Reserved
85 (Group 1), 117 (Group 2)	Protocol_Register_12	Reserved
86 (Group 1), 118 (Group 2)	Protocol_Register_13	Reserved
87 (Group 1), 119 (Group 2)	Reserved	Reserved
88-91 (Group 1), 120-123 (Group 2)	Protocol_Register_8	Reserved
92-95	Reserved	Reserved
124-127	Reserved for board-level circuitry (RTSI Switch)	Reserved

Table B-2. Register Map Listed by Register Name

Register Name	Offset (in decimal)	Type
Chip_ID	24-27	Read
Data_Path	64 (Group 1) 96 (Group 2)	Write
DMA_Line_Control	76 (Group 1) 108 (Group 2)	Write
FIFO_A	16	Read / Write
FIFO_B	17	Read / Write
FIFO_C	18	Read / Write
FIFO_D	19	Read / Write
FIFO_Control	72 (Group 1) 104 (Group 2)	Write
Group_1_FIFOs	8–11	Read / Write
Group_1_First_Clear	6	Write
Group_1_Flags	6	Read
Group_1_Second_Clear	46	Write
Group_2_FIFOs	12–15	Read / Write
Group_2_First_Clear	7	Write
Group_2_Flags	7	Read
Group_2_Second_Clear	47	Write
Group_Status	5	Read
Interrupt_and_Window_Status	4	Read
Interrupt_Control	75 (Group 1) 107 (Group 2)	Write
Master_Clock_Routing	45	Write
Master_DMA_and_Interrupt_Control	5	Write
Pattern_Detection	81 (Group 1) 113 (Group 2)	Write
Port_A_Input	28	Read

Table B-2. Register Map Listed by Register Name (Continued)

Register Name	Offset (in decimal)	Type
Port_A_Output	28	Write
Port_A_Pattern	48	Write
Port_A_Pin_Directions	32	Write
Port_A_Pin_Mask	86	Write
Port_A_Polarities	40	Write
Port_B_Input	29	Read
Port_B_Output	29	Write
Port_B_Pattern	49	Write
Port_B_Pin_Directions	33	Write
Port_B_Pin_Mask	37	Write
Port_B_Polarities	41	Write
Port_C_Input	30	Read
Port_C_Output	30	Write
Port_C_Pattern	50	Write
Port_C_Pin_Directions	34	Write
Port_C_Pin_Mask	38	Write
Port_C_Polarities	42	Write
Port_D_Input	31	Read
Port_D_Output	31	Write
Port_D_Pattern	51	Write
Port_D_Pin_Directions	35	Write
Port_D_Pin_Mask	39	Write
Port_D_Polarities	43	Write
Protocol_Register_1	65 (Group 1) 97 (Group 2)	Write
Protocol_Register_2	66 (Group 1) 98 (Group 2)	Write

Table B-2. Register Map Listed by Register Name (Continued)

Register Name	Offset (in decimal)	Type
Protocol_Register_3	67 (Group 1) 99 (Group 2)	Write
Protocol_Register_4	70 (Group 1) 102 (Group 2)	Write
Protocol_Register_5	71 (Group 1) 103 (Group 2)	Write
Protocol_Register_6	73 (Group 1) 105 (Group 2)	Write
Protocol_Register_7	74 (Group 1) 106 (Group 2)	Write
Protocol_Register_8	88 - 91 (Group 1) 120 - 123 (Group 2)	Write
Protocol_Register_9	82 (Group 1) 114 (Group 2)	Write
Protocol_Register_10	83 (Group 1) 115 (Group 2)	Write
Protocol_Register_11	84 (Group 1) 116 (Group 2)	Write
Protocol_Register_12	85 (Group 1) 117 (Group 2)	Write
Protocol_Register_13	86 (Group 1) 118 (Group 2)	Write
Protocol_Register_14	68-69 (Group 1) 100-101 (Group 2)	Write
Protocol_Register_15	79 (Group 1) 111 (Group 2)	Write
Reserved for board-level circuitry (RTSI switch)	124-127	Write
Subspace_1	56-63	Read / Write
Transfer_Count	20-23 (Group 1) 24-27 (Group 2)	Write

Table B-2. Register Map Listed by Register Name (Continued)

Register Name	Offset (in decimal)	Type
Transfer_Size_Control	77 (Group 1) 109 (Group 2)	Write
Window_Address	4	Write
Window_Data	0 - 3	Read / Write

Bitfield Locations

This section provides bitfield locations for DAQ-DIO registers.

Data_Path

Group 1	Offset: 64
Group 2	Offset: 96
Type: Write Only	

0	FIFOEnables(A)
1	FIFOEnables(B)
2	FIFOEnables(C)
3	FIFOEnables(D)
4	Funneling
5	Funneling
6	
7	GroupDirection

DMA_Line_Control

Group 1	Offset: 76
Group 2	Offset: 108
Type: Write Only	

0	DMAChannel
1	DMAChannel
2	DMAChannel
3	DMAChannel
4	Reserved
5	Reserved
6	Reserved
7	Reserved

FIFO_Control

Group 1	Offset: 72
Group 2	Offset: 104
Type: Write Only	

0	ReadyLevel
1	ReadyLevel
2	ReadyLevel
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

Group_FIFOs

Group_1_FIFOs	Offset: 8-11
Group_2_FIFOs	Offset: 12-15
Type: Read/Write	

0	FIFOdata
1	FIFOdata
2	FIFOdata
3	FIFOdata
4	FIFOdata
5	FIFOdata
6	FIFOdata
7	FIFOdata

Group_First_Clear

Group_1_First_Clear Offset: 6
Group_2_First_Clear Offset: 7
Type:

0	
1	
2	
3	ClearWaited
4	ClearPrimaryTC
5	ClearSecondaryTC
6	DMAReset
7	FIFOReset

Group_Flags

Group_1_Flags Offset: 6
Group_2_First_Clear Offset: 7
Type: Read Only

0	TransferReady
1	CountExpired
2	
3	
4	
5	Waited (overflow/underflow)
6	PrimaryTC
7	SecondaryTC

Group_Second_Clear

Group_1_Second_Clear Offset: 46
Group_2_Second_Clear Offset: 47
Type: Write Only

0	ClearExpired
1	
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

Group_Status

Offset: 5
Type: Read Only

0	DataLeft(1)
1	Reserved
2	Req(1)
3	StopTrig(1)
4	DataLeft(2)
5	Reserved
6	Req(2)
7	StopTrig(2)

Interrupt_and_Window_Status

Offset: 4
Offset:
Type: Read Only

0	IntStatus(1)
1	IntStatus(2)
2	WindowAddressStatus
3	WindowAddressStatus
4	WindowAddressStatus
5	WindowAddressStatus
6	WindowAddressStatus
7	Reserved

Interrupt_Control

Group 1 Offset: 75
Group 2 Offset:107
Type: Write Only

0	IntEnables(0) (Transfer Ready)
1	IntEnables(1) (Count Expired)
2	
3	
4	
5	IntEnables(5) (waited)
6	IntEnables(6) (primary TC)
7	IntEnables(7) (secondary TC)

Master_DMA_and_Interrupt_Control

Offset: 5
Type: Write Only

0	InterruptLine
1	InterruptLine
2	OpenInt
3	
4	Reserved
5	Reserved
6	Reserved
7	Reserved

Master_Clock_Routing

Offset:45
Type: Write Only

0	Reserved
1	Reserved
2	Reserved
3	Reserved
4	RTSIClocking
5	RTSIClocking
6	Reserved
7	Reserved

Pattern_Detection

Group 1	Offset: 81
Group 2	Offset: 113
Type: Write Only	

0	DetectionMethod
1	InvertMatch
2	IntEnables(8)
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

Port_Input

Port_A_Input	Offset: 28
Port_B_Input	Offset: 29
Port_C_Input	Offset: 30
Port_D_Input	Offset: 31
Type: Read Only	

0	InputData
1	InputData
2	InputData
3	InputData
4	InputData
5	InputData
6	InputData
7	InputData

Port_Output

Port_A_Output	Offset: 28
Port_B_Output	Offset: 29
Port_C_Output	Offset: 30
Port_D_Output	Offset: 31
Type: Write Only	

0	OutputData
1	OutputData
2	OutputData
3	OutputData
4	OutputData
5	OutputData
6	OutputData
7	OutputData

Port_Pattern

Port_A_Pattern	Offset: 48
Port_B_Pattern	Offset: 49
Port_C_Pattern	Offset: 50
Port_D_Pattern	Offset: 51
Type: Write Only	

0	Pattern
1	Pattern
2	Pattern
3	Pattern
4	Pattern
5	Pattern
6	Pattern
7	Pattern

Port_Pin_Directions

Port_A_Pin_Directions Offset: 32
Port_B_Pin_Directions Offset: 33
Port_C_Pin_Directions Offset: 34
Port_D_Pin_Directions Offset: 35
Type: Write Only

0	PinDirections
1	PinDirections
2	PinDirections
3	PinDirections
4	PinDirections
5	PinDirections
6	PinDirections
7	PinDirections

Port_Pin_Mask

Port_A_Pin_Mask Offset: 36
Port_B_Pin_Mask Offset: 37
Port_C_Pin_Mask Offset: 38
Port_D_Pin_Mask Offset: 39
Type: Write Only

0	PinMask
1	PinMask
2	PinMask
3	PinMask
4	PinMask
5	PinMask
6	PinMask
7	PinMask

Protocol_Register_1

Group 1 Offset: 65
Group 2 Offset: 97
Type: Write Only

0	RunMode
1	RunMode
2	RunMode
3	Numbered
4	Protocol(1)
5	Protocol(1)
6	Protocol(1)
7	

Protocol_Register_2

Group 1 Offset: 66
Group 2 Offset: 98
Type: Write Only

0	Protocol(2)
1	Protocol(2)
2	Protocol(2)
3	Protocol(2)
4	Protocol(2)
5	Protocol(2)
6	Protocol(2)
7	InventStopTrig

Protocol_Register_3

Group 1 Offset: 67
Group 2 Offset: 99
Type: Write Only

0	Protocol(3)
1	Protocol(3)
2	Protocol(3)
3	Protocol(3)
4	Protocol(3)
5	Protocol(3)
6	Protocol(3)
7	Protocol(3)

Protocol_Register_4

Group 1 Offset: 70
Group 2 Offset: 102
Type: Write Only

0	Protocol(4)
1	Protocol(4)
2	Protocol(4)
3	Protocol(4)
4	Protocol(4)
5	Protocol(4)
6	Protocol(4)
7	Protocol(4)

Protocol_Register_5

Group 1 Offset: 71
Group 2 Offset: 103
Type: Write Only

0	Protocol(5)
1	Protocol(5)
2	Protocol(5)
3	Protocol(5)
4	Protocol(5)
5	Protocol(5)
6	Protocol(5)
7	Protocol(5)

Protocol_Register_6

Group 1 Offset: 73
Group 2 Offset: 105
Type: Write Only

0	InvertAck
1	InvertReq
2	InvertClock
3	InvertSerial
4	OpenAck
5	OpenClock
6	Reserved
7	Reserved

Protocol_Register_7

Group 1 Offset: 74
Group 2 Offset: 106
Type: Write Only

0	Protocol (7)
1	Protocol (7)
2	Protocol (7)
3	Protocol (7)
4	Protocol (7)
5	Protocol (7)
6	Protocol (7)
7	ExchangePins

Protocol_Register_8

Group 1 Offset: 88-91
Group 2 Offset: 120-123
Type: Write Only

0	Protocol(8)
1	Protocol(8)
2	Protocol(8)
3	Protocol(8)
4	Protocol(8)
5	Protocol(8)
6	Protocol(8)
7	Protocol(8)

Protocol_Register_9

Group 1 Offset: 82
Group 2 Offset: 114
Type: Write Only

0	Protocol(9)
1	Protocol(9)
2	Protocol(9)
3	Protocol(9)
4	Protocol(9)
5	Protocol(9)
6	Protocol(9)
7	Protocol(9)

Protocol_Register_10

Group 1 Offset: 83
Group 2 Offset: 115
Type: Write Only

0	Protocol(10)
1	Protocol(10)
2	Protocol(10)
3	Protocol(10)
4	Protocol(10)
5	Protocol(10)
6	Protocol(10)
7	Protocol(10)

Protocol_Register_11

Group 1 Offset: 84
Group 2 Offset: 116
Type: Write Only

0	Protocol(11)
1	Protocol(11)
2	Protocol(11)
3	Protocol(11)
4	Protocol(11)
5	Protocol(11)
6	Protocol(11)
7	Protocol(11)

Protocol_Register_12

Group 1 Offset: 85
Group 2 Offset: 117
Type: Write Only

0	Protocol(12)
1	Protocol(12)
2	Protocol(12)
3	Protocol(12)
4	Protocol(12)
5	Protocol(12)
6	Protocol(12)
7	Protocol(12)

Protocol_Register_13

Group 1 Offset: 86
Group 2 Offset: 118
Type: Write Only

0	Protocol(13)
1	Protocol(13)
2	Protocol(13)
3	Protocol(13)
4	Protocol(13)
5	Protocol(13)
6	Protocol(13)
7	Protocol(13)

Protocol_Register_14 (16 bits)

Group 1 Offset: 68-69
Group 2 Offset: 100-101
Type: Write Only

0	Protocol(14)
1	Protocol(14)
2	Protocol(14)
3	Protocol(14)
4	Protocol(14)
5	Protocol(14)
6	Protocol(14)
7	Protocol(14)

Protocol_Register_15

Group 1 Offset: 79
Group 2 Offset: 111
Type: Write Only

0	StartSource
1	StartSource
2	InvertStart
3	StopSource
4	StopSource
5	Reserved
6	ReqSource
7	PreStart

Transfer_Count (32 bits)

Group 1 Offset: 20
Group 2 Offset: 24
Type: Write Only

0	TransferCount
1	TransferCount
2	TransferCount
3	TransferCount
4	TransferCount
5	TransferCount
6	TransferCount
7	TransferCount

Transfer_Size_Control

Group 1 Offset: 77
Group 2 Offset: 109
Type: Write Only

0	TransferWidth
1	TransferWidth
2	Reserved
3	TransferLength
4	TransferLength
5	RequireRLevel
6	Reserved
7	Reserved

Window_Data

Offset: 0-3
Type: Read/Write

0	WindowData
1	WindowData
2	WindowData
3	WindowData
4	WindowData
5	WindowData
6	WindowData
7	WindowData

Window_Address

Offset: 4

Type: Write Only

0	Ignored
1	Ignored
2	WindowAddress
3	WindowAddress
4	WindowAddress
5	WindowAddress
6	WindowAddress
7	Reserved

Appendix C

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 (Quebec)	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:____

____

____

____

____

____

DAQ-DIO 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

DAQ hardware ________________________________

Interrupt level of hardware ________________________________

DMA channels of hardware ________________________________

Base I/O address of hardware ________________________________

Programming choice ________________________________

Software and version ________________________________

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: *DAQ-DIO Technical Reference Manual*

Edition Date: ? 1998

Part Number: 341330A-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
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}

Numbers/Symbols

A

Index