DATAQ® INSTRUMENTS

The way PC-based instrumentation should be

Programmer's Software Development Kit

User's Manual

Manual Revision G

Software Release Level 1

DATAQ® INSTRUMENTS

Designed and manufactured in the
United States of America

Table of Contents

Programmer’s Software Development Kit **1**

Programmer’s SDK Installation Instructions 1
Manually Modifying the AUTOEXEC.BAT File 3
Batch File Details 5
Setting The Device Driver Environment Variable 6
Installing the Device Driver 6
On-board DSP Program Installation 7

General SDK Information **8**

Programmer’s SDK Data Types and Ranges 8
Using INCLUDE Files 8
Input and Output Data Buffer Management 9
Buffer Allocation 9
Buffer Access 9
All instruments except DI-401 and DI-700 9
DI-401 and DI-700 Instruments 9
All instruments except DI-401 and DI-700 10
DI-401 and DI-700 Instruments 10
Input and Output Data Buffer Architecture 10
Converting Counts to Volts 12
Sampling Different Channels at Different Rates 14
How Data is Received From an Input Buffer 18

Function Reference **19**

Programmer’s SDK Functions 19
Hardware Support for Programmer’s SDK Functions 19
Initialization and Information Functions 20
Buffer Functions 20
Immediate Functions 20
Scanning Functions 20
Counter/Timer Functions 21
Miscellaneous Functions 21
Equivalent HP VEE Functions 21
Programming Sequences and Flow Charts 23
Function Reference 29
di_anin 30
di_anout 35
di_buffer_alloc 37
di_buffer_free 38
di_buffer_size 40
di_buffer_status 41
di_burst_rate 42
di_close 46
di_copy_array 47
di_copy_buffer 48
di_copy_header 49
di_copy_mux 51
di_ct_event 52
di_ct_one_shot 55

di_ct_status 57
di_ct_stop 58
di_ct_wave 59
di_digin 61
di_digout 63
di_get_acq_header 64
di_info 65
di_inlist 68
di_list_length 78
di_mode 79
di_open 86
di_outlist 87
di_set_data_mode 89
di_start_scan 90
di_status_read 92
di_stop_scan 93
di_strerr 95
di_trigger_status 96

Programmer's Software Development Kit

The Programmer's software development kit provides everything you need to program DI-200 Series, DI-400 Series, DI-500 Series, DI-700, DI-720, and DI-730 waveform recording instruments from virtually any standard programming language.

Note that nothing in this SDK pertains to WINDAQ software. If you purchased WINDAQ software, then you already have everything you need to record, playback and analyze waveform signals (ignore this manual and refer back to the hardware manual or to the WINDAQ/Lite or WINDAQ/Pro and WINDAQ/Pro+ User's Manual). You do not need to install this software.

If, however you intend to write your own programs (including LabVIEW, TestPoint, HP VEE, etc.) to record, playback and analyze waveform signals using DI-200 Series, DI-400 Series, DI-500 Series, DI-700, DI-720, or DI-730 hardware, then this manual contains the information you need. Proceed with the following Programmer's SDK installation instructions.

Programmer's SDK Installation Instructions

The following procedure can be used to install the Programmer's SDK onto your computer (DI-700 user's can ignore the following procedure since SDK files for the DI-700 are automatically installed, along with WINDAQ/Lite and the WINDAQ Waveform Browser, when the DI-700 is installed).

1. Start Windows™.

2. Insert the *Programmer's Software Development Kit* disk into your 3½″ floppy drive.

3. Click the Start button on the taskbar and then click Run… (Windows 95, Windows 98, Windows NT), or from the Windows™ Program Manager window choose Run from the File menu (Windows 3.1).

4. In the text box that appears, type ***d:setup*** (where ***d*** specifies the drive containing the Programmer's SDK distribution disk) and press ENTER.

   A welcome dialog box appears onscreen.

5. Choose the OK button to start the installation.

6. Follow the instructions onscreen to specify the directory where you want to install Programmer's SDK files.

   We recommend you accept the default path, but you can name this new directory anything you like. Simply substitute the desired drive and directory in the Destination Directory: text box.

7. Choose the OK button.

   A dialog box is displayed asking if you would like to make backup copies of all files replaced during the installation. This is offered as a safety courtesy, backup copies are not required.

   Choose the No button if you don't want to make backups.

Choose the Yes button to create backups. If you decide to create backups, you will be prompted to specify a backup file directory.

8. Specify the instrument that will be used with the Programmer's SDK.

   For example, choose the DI-400/401 Plug-in Card button if you have a DI-400 board installed.

   When the appropriate hardware button is selected and the OK button is chosen, an informational window is displayed showing the progress of the installation.

9. Specify a destination (or group window) for the Programmer's SDK icons.

   Again we recommend the default path, but you can specify any group window you like.

10. Choose the OK button to continue.

11. The remaining installation steps vary by instrument, and in some cases by operating system (i.e., Windows 3.1, Windows 95, Windows 98, etc.). In most cases, the on-screen prompts provide enough information to successfully get you through the installation. However, if you are unsure of what to do next or if you need additional information, refer to the following instrument specific notes.

For DI-200, DI-210, DI-400, DI-401, and DI-410 Plug-in Boards:
A dialog box is displayed, asking if you would like to have the installation program automatically modify your AUTOEXEC.BAT file or if you would like to do it yourself manually. We recommend that you have the installation program do it automatically (by choosing Yes) simply because it's easier and less likely to create a problem. However, you can do it either way.

If you choose to manually modify the AUTOEXEC.BAT file yourself (by choosing No), you *must* use a text editor that saves files as unformatted (ASCII) text (some text editors refer to this as "text only" format). WordPad and/or Notepad are examples of text editors that will do the job and come free-of-charge with Windows. Steps for manually modifying the AUTOEXEC.BAT file appear at the end of this installation procedure.

After the AUTOEXEC.BAT decision is made, a second dialog box is displayed prompting you for a base address. This is the same value you configured the DIP switches for when you installed the board (if you didn't change the DIP switch settings during installation, then choose the default value of 180_{hex}). If you don't remember how you configured the DIP switches, refer back to the hardware User's Manual.

After the base address is specified on all plug-in boards except the DI-200 and DI-210, you must restart (reboot) your computer to complete the installation. Remove the distribution disk from your drive and choose the OK button to restart your computer.

On DI-200 and DI-210 boards, a third dialog box is displayed providing the opportunity to alter advanced configuration settings such as hardware interrupt level, input DMA channel, output DMA channel, and pre-allocated input data buffer size. Choose the No button, thus declining the opportunity to alter the advanced configuration options. *You should only modify these options if, after the initial installation, you were unable to run the Programmer's SDK* ***and*** *you talked to DATAQ Instruments technical support*. They can help you determine which option(s) need to be changed in order to run the Programmer's SDK. When finished, you must restart (reboot) your computer to complete the installation. Remove the distribution floppy from your drive and choose the OK button to restart your computer.

For DI-220, DI-221TC, DI-222, DI-500 Series, DI-720, DI-730, and DI-5001 Instruments:
A dialog box is displayed, asking if you would like to have the installation program automatically modify your AUTOEXEC.BAT file or if you would like to do it yourself manually. We recommend that you have the installation

program do it automatically (by choosing Yes) simply because it’s easier and less likely to create a problem. However, you can do it either way.

If you choose to manually modify the AUTOEXEC.BAT file yourself (by choosing No), you *must* use a text editor that saves files as unformatted (ASCII) text (some text editors refer to this as “text only” format). WordPad and/or Notepad are examples of text editors that will do the job and come free-of-charge with Windows. Steps for manually modifying the AUTOEXEC.BAT file appear at the end of this installation procedure.

After the AUTOEXEC.BAT decision is made, a second dialog box is displayed prompting you to specify the printer (LPT) or parallel port number to which the instrument is connected. When the proper LPT port is specified, you must restart (reboot) your computer to complete the installation. Remove the distribution disk from your drive and choose the OK button to restart your computer.

Manually Modifying the AUTOEXEC.BAT File

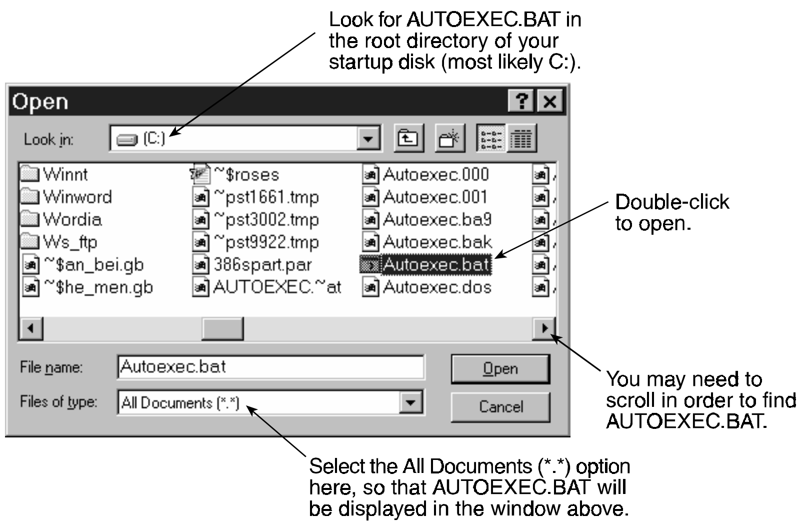
The following steps will guide you through the process of manually modifying the AUTOEXEC.BAT file. *Note that this procedure is only necessary when, during the installation, you chose to manually modify the AUTOEXEC.BAT file*. As a precaution, you may want to print a copy of your AUTOEXEC.BAT file before starting. This will give you a hard copy record of your existing AUTOEXEC.BAT file before any changes are made.

a. Start your text editor.

   For example, in Windows 95 start WordPad by clicking the Start button on the taskbar, pointing to Programs, pointing to Accessories, and clicking WordPad.

b. Using the text editor, open the AUTOEXEC.BAT file.

   For example, in WordPad click on the File menu and click Open… This displays the Open dialog box as follows:


Look for AUTOEXEC.BAT in the root directory of your startup disk (most likely C:).

Double-click to open.

You may need to scroll in order to find AUTOEXEC.BAT.

Select the All Documents (*.*) option here, so that AUTOEXEC.BAT will be displayed in the window above.

File name: Autoexec.bat

Files of type: All Documents (*.*)

c. When opened, scroll down to the end of your AUTOEXEC.BAT file.

d. At the end of your AUTOEXEC.BAT file, but above the “win” command (if included), add the following two commands (each command must begin on a separate line):

<u>A typical AUTOEXEC.BAT file:</u>

```
@echo off
path
c:\;c:\dos;c:\windows
set temp=c:\temp
doskey
smartdrv
win
```

<u>Add these two commands:</u>

cd\DATAQSDK
call diXXXX.bat

Use the same directory specified in step 6

Use the appropriate batch file for your hardware (see table below)

The first line directs your computer to go to the directory where Programmer's SDK files are installed. The second line executes a batch file that is required for software operation. The proper batch file to enter in this line can be found in the following table. Note that this batch file is dependent on the instrument you are using:

If you have this hardware:	Use this batch file:
Plug-in boards	
DI-200	**di200.bat**
DI-201	**Di201m.bat**
DI-210	**di210.bat**
DI-400	**di400.bat**
DI-401	**di400.bat**
DI-410	**di400.bat**
Printer port instruments	
DI-220/DI-222	**di220.bat**
DI-221TC	**di221.bat**
DI-500	**di500.bat**
DI-510	**di500.bat**
DI-720	**di720.bat**
DI-730	**di730.bat**
DI-5001	**di720.bat**

For example, say we have a DI-500-16-P instrument. A typical AUTOEXEC.BAT file, with the added commands, would look similar to this:

```
@echo off
path c:\;c:\dos;c:\windows
set temp=c:\temp
doskey
smartdrv
cd\dataqsdk
call di500.bat
win
```

Note that if the "win" command is not included in your AUTOEXEC.BAT file, the two lines just entered will be at the bottom of the file.

e. When finished, save the changes to the AUTOEXEC.BAT file (save as a text file only!) and exit the text editor.

f. Restart (re-boot) your computer.

The batch file you just added to your AUTOEXEC.BAT file automatically sets an environment variable, installs a device driver, downloads the DSP program to the hardware, and on parallel port instruments, specifies the printer (LPT) port to which the hardware is connected. This batch file installs all default values for these parameters. If you need to install a parameter other than the default value, you must edit the appropriate batch file. This can be done with your text editor using an approach similar to that used to edit the AUTOEXEC.BAT file. Complete batch file details follow this installation procedure.

Note
Using DI-220, DI-221TC, or DI-222 Instruments with Windows 3.1 or 3.11:
If you have a DI-220, DI-221TC, or DI-222 battery-powered instrument and you are using Windows 3.1 or 3.11, power will be applied to the instrument (thus beginning battery-powered operation) immediately after booting. This may not be desirable, especially when using the instrument "in the field", where conserving battery power is an important consideration. However, power can be turned off by issuing the loader command without an argument. For example, typing **220LDR** and pressing the ENTER key turns instrument power off. To turn instrument power back on when you are ready to run your application, simply issue the loader command with the proper argument (type **220LDR DI-220.BNM**).

Note
Using DI-220, DI-221TC, or DI-222 Instruments with Windows 95:
Under Windows 95, a copy of the "DI-22X Power On" icon is placed in your Startup folder if you are using a DI-220, DI-221TC, or DI-222 battery-powered instrument. It is this icon that automatically applies power to the instrument immediately after booting. If this is undesirable, you can delete this item from your Startup folder and manually control instrument power. This is done by issuing the loader command with the proper argument. For example, instrument power is turned on by clicking the Start button on the taskbar, clicking Run, and typing **220LDR DI-220.BNM** in the dialog box that appears. To turn instrument power off when the instrument is not in use, simply issue the loader command without an argument (type **220LDR** in the dialog box).

g. You are now ready to start developing applications with the Programmer's SDK. The following illustrates a "roadmap" of what was installed:

```
All Instruments except DI-700                                    DI-700 Instruments
Root Directory                                                   WINDAQU (default) Directory
|                                                                |
|-DOS                                                            |-SDK
|    C        → "C" files and sample programs for DOS            |    3 Windows "C" header files
|    QB       → Quick BASIC files and sample programs for DOS    |    Library that links to the DLL
|    TPASCAL  → Turbo PASCAL files and sample programs for DOS   |    Sample program source code
|    VBDOS    → Visual BASIC files and sample programs for DOS   |    Sample program
|-WIN                                                            |    Make file
|    C        → "C" files and sample programs for Windows        |
|    VB       → Visual BASIC files and sample programs for Windows
|-Batch files, drivers, executables, readme's, etc.              |-WINDAQ/Lite, WWB, etc.
```

Batch File Details

The following command structure lists the major items in the batch file. You should only be concerned with the following descriptions if you need to edit the batch file.

Setting The Device Driver Environment Variable

Form:	`SET DI=`*value*
Where:	Only one space exists in the command line, immediately following SET. *Value* defines the software interrupt level to be used for the instrument's device driver. *Value* can be 60, 61, 62, 63, 64, 65, or 66. Default value is 60.
Example:	`SET DI=60`

Installing the Device Driver

Form (2 kinds):

For Plug-in Boards:	For Printer Port Instruments:
`DI-x00 [PORT] [HI] [DI] [DO]`	`DI-xxx [LPT]`

Where: All arguments (shown in brackets) are in hexadecimal format and must be separated by a space.

PORT - specifies the hardware base address selected for the DI-200, or DI-400 Series instrument as instructed by their respective hardware user's manual. The three least significant bits must be zero. The range for PORT is 100H ≤ PORT ≤ 3F8. The default value is 180H.

HI - specifies the hardware interrupt level selected for the DI-200 Series instrument as instructed by their respective hardware user's manual. HI can be A, B, E, or F, which corresponds to interrupt levels 10, 11, 14, and 15 respectively. The default value is A (10).

DI - specifies the input DMA channel (the DMA channel number being used for the reporting of A/D conversions and readings from the digital input port) selected for the DI-200 Series instrument. The range for DI is 5 ≤ DI ≤ 7. Set DI = 0 to disable DMA input operations. The default value is 6.

DO - specifies the output DMA channel (the DMA channel number being used to receive analog output and digital output values) selected for the DI-200 Series instrument. The range for DO is 5 ≤ DO ≤ 7. Set DO = 0 to disable DMA output operations. The default value is 7. To avoid DMA channel conflicts, do not set DO equal to DI (DO ≠ DI).

LPT - specifies the printer (LPT) or parallel port number to which the instrument is connected. The range for LPT is 1 (LPT1), 2 (LPT2), 3 (LPT3), or 4 (LPT4). The default value is 1 (LPT1).

Example 1:

```
DI-200 180 A 6 0
```

The above command installs the DI-200 Series device driver and configures the driver for a base address of 180H, a hardware interrupt level of A (10 decimal), and a DMA input channel of 6. The DMA output channel has been disabled.

Example 2:

```
DI-500 2
```

This command installs the DI-500 Series device driver on LPT 2.

Upon proper installation of the driver, a copyright notice will appear identifying the driver, its version, and date.

Error Messages:

```
“DI-xxx device driver has already been installed”
“Invalid port address”
“Invalid hardware interrupt level”
“Invalid software interrupt level”
“Invalid DMA input channel number”
“Invalid DMA output channel number”
“Dataq Interrupt invalid or not found in environment space”
“Software interrupt already installed”
```

On-board DSP Program Installation

Form:

`200LDR` *`argument`* **(For all DI-200 Series instruments except the DI-220, DI-221TC, and the DI-222)**

`220LDR` *`argument`* **(For DI-220, DI-221TC, and DI-222 instruments)**

`400LDR` *`argument`* **(For DI-400 Series instruments)**

`500LDR` *`argument`* **(For DI-500 Series, DI-720, DI-730, and DI-5001 instruments)**

Where: One space must separate *argument* from 200LDR, 220LDR, 400LDR, or 500LDR.

Argument specifies the name of the binary file containing the program for the device's on-board DSP. *Argument* files are identified with a .BNM extension and, in most cases, have the same name as the hardware (i.e., if you are using a DI-210 instrument, the argument file would be DI-210.BNM). The following table shows the loader arguments that are available for each batch file.

Batch File	Available Arguments
DI200.BAT	di-200.bnm
DI201M.BAT	di-201m.bnm
DI210.BAT	di-210.bnm
DI220.BAT	di-220.bnm
DI221.BAT	di-221.bnm
DI400.BAT	di-400.bnm
DI500.BAT	di-500.bnm
DI720.BAT	di-720.bnm
DI730.BAT	di-730.bnm

If you are experiencing a problem, consult the “readme” text file created during software installation or contact Dataq Instruments technical support to find out which .BNM file to use.

Example 1:

```
200LDR DI-210.BNM
```

The above command down-loads the binary file DI-210.BNM containing the DSP program to the DI-210 board.

Example 2:

```
400LDR DI-400.BNM
```

The above command down-loads the binary file DI-400.BNM containing the DSP program to the DI-400 Series board.

Example 3:

```
500LDR 1 DI-720.BNM
```

The above command down-loads the binary file DI-720.BNM containing the DSP program to the DI-720 instrument.

General SDK Information

Programmer’s SDK Data Types and Ranges

The Programmer’s SDK uses several data types defined as follows:

- **Integer—**a 16-bit signed value with a range of -32,768 to +32,767
- **Unsigned integer—**a 16-bit unsigned value with a range of 0 to 65,535
- **Float—**a 32-bit value using the IEEE single precision floating point format to represent numbers having a fractional format
- **Array—**a sequential organization of similar data types, usually integer or floating point numbers. SDK functions require the address of this type of argument (passed-by-reference).
- **String—**a sequence of ASCII characters terminated with a NULL byte. SDK functions require the address of this type of argument (passed-by-reference).
- **Structure—**a sequential organization of related data, not necessarily having similar data types. SDK functions require the address of this type of argument (passed-by-reference).

The following table lists the various data types described above and indicates the equivalent data type for each supported language:

Data Type	BASIC	“C”	Pascal
integer	%	int	integer
unsigned integer*	%	unsigned int	word
float	!	float	single
long	&	long	longint
array	DIM	int []	array [1..10] of integer
string	$	char[] or char*	array [1..10] of char
structure	type	struct	record

*Note that Quick BASIC does not support the unsigned integer data format. Unsigned integers in the range of 32,768 to 65,535 must use the negative signed integer whose representation is the same as its unsigned counterpart. Use the following equation to convert between signed and unsigned equivalents:

signed equivalent = 32,768-(unsigned value)

Using INCLUDE Files

The Programmer’s SDK provides files 200SDKQ.BI and 200SDKV.BI which must be included in your program if you are using Microsoft Quick BASIC or Visual BASIC for DOS respectively. These files provide definitions of all constants, function prototypes, and structures used by the driver. For example, the Quick BASIC metacommand syntax required to perform this inclusion is as follows:

```
REM $INCLUDE:'200SDKQ.BI'
```

or

```
'$INCLUDE:'200SDKQ.BI'
```

File “GLOBAL.BAS” (found in the \WIN\VB subdirectory) performs the same function for the Visual BASIC environment, and must become part of your Visual BASIC program.

Microsoft C and Quick C users should note that the “.H” files contain similar equates and structure definitions and must be included in your program. Use “200SDK.H” for all Windows™ and non-Windows™ programming.

Input and Output Data Buffer Management

Most instruments allow up to two data buffers to coexist. One may be designated as an input buffer, and the other an output buffer (DI-401 and DI-700 instruments do not support output operations, therefore there is no output buffer). When configured to do so, an instrument may simultaneously send and receive data to and from the buffers (simultaneous input and output operations are not possible with DI-401 and DI-700 instruments). For example, analog input operations may occur while analog output operations are being performed. This capability allows an instrument to act not only as a data acquisition tool, but also as a stimulator.

Buffer Allocation

An input buffer (on DI-401 and DI-700 instruments) or both input and output buffers (on the rest of the instruments) are allocated with the **di_buffer_alloc** command. When a buffer is allocated successfully, the **di_buffer_alloc** command returns a far integer pointer which may be used by the issuing program to gain access to the buffer. A NULL is returned if the allocation attempt failed because of insufficient memory.

Buffer Access

The buffer allocation command **di_buffer_alloc** returns a far integer pointer if the allocation was successful. This pointer may be used directly if the programming language is C. For example, the following C program allocates a buffer, then initializes it with a ramp function ranging from 0 to 1024:

<u>All instruments except DI-401 and DI-700 (these instruments do not support output operations)</u>

```
#include  "200SDK.H"

int far  *output_buffer;
int      i;

main()
{
        if((output_buffer=di_buffer_alloc(1,4096)==NULL){
             printf("Insufficient memory or output buffer already allocated...\n");
             exit(0);
        }
        for(i=0;i<1024;i++)
             *(output_buffer+i)=i;
}
```

<u>DI-401 and DI-700 Instruments</u>

```
#include  "200SDK.H"

int far  *some_buffer;
int      i;

main()
{
        if((some_buffer=di_buffer_alloc(0,4096)==NULL){
             printf("Insufficient memory or input buffer already allocated...\n");
             exit(0);
        }
        for(i=0;i<1024;i++)
             *(some_buffer+i)=i;
}
```

However, Quick BASIC or Visual BASIC cannot use pointers returned by **di_buffer_alloc**. For these languages, two special functions are provided in the 200SDK library: **di_copy_buffer** for moving the contents of a buffer into a BASIC array, and **di_copy_array** for writing the contents of a BASIC array to a buffer. They are used as follows:

All instruments except DI-401 and DI-700 (these instruments do not support output operations)

```
dim anlgout%[4096]
dim analgin%[4096]
NULL=val("")

in_buf&=di.buffer.alloc(0,4096)
IF in.buf&=NULL THEN PRINT "Insufficient memory":END

out.buf&=di.buffer.alloc(1,4096)
IF out.buf&=NULL THEN PRINT "Insufficient memory":END

'****Copy input buffer to BASIC array analgin%
i%=di.copy.buffer(0,analgin%(0),4096)
IF i%<>0 THEN PRINT "copy buffer function error"

'****Copy the contents of BASIC array analout% into the output buffer
i%=di.copy.array(0,anlgout%(0),4096)
IF i%<>0 THEN PRINT "copy array function error"
```

DI-401 and DI-700 Instruments

```
dim analgin%[4096]
NULL=val("")

in.buf&=di.buffer.alloc(0,4096)
IF in.buf&=NULL THEN PRINT "Insufficient memory":END

'****Copy input buffer to BASIC array analgin%
i%=di.copy.buffer(0,analgin%(0),4096)
IF i%<>0 THEN PRINT "copy buffer function error"
```

Input and Output Data Buffer Architecture

All instruments may write data to the input buffer from analog input signals or from the digital input port. Similarly, data contained in the output buffer may be written to the DAC or to the digital output port on DI-200 Series instruments; or to either DAC (DAC1 or DAC2) or to the digital output port on DI-400 Series, DI-500 Series, DI-720, and DI-730 instruments (DI-401 and DI-700 instruments do not support output operations, therefore there is no output buffer). The following describes the format of any word appearing in the input or output buffer. These data word formats assume that only the SDK is running. If WINDAQ software is running in conjunction with the SDK, Y1 and Y0 are undefined. Because DI-200 Series instruments have only one I/O port, both DMA data *and* control information (i.e., commands, status, etc.) must share this single port. Therefore, in order to distinguish the word as DMA data (as opposed to control information), the two LSB's of the data word *must* be zero:

Analog In Word

All 12-bit instruments (DI-200, DI-400, DI-401, and DI-500 Series) with signal averaging off, or DI-220, DI-221TC, and DI-222 always (regardless of signal averaging).

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	data	data	data	data	data	data	data	data	data	data	LSB	0	0	Y1*	Y0*

D=indicated digital I/O line; *If you are looking at the first channel in the scan list, Y1 reflects the inverted state of the remote start/stop flag and Y0 reflects the inverted state of the event marker flag. Otherwise, Y1 and Y0 are zero. If analog data is a unipolar signal, the number is a straight binary value. Bipolar signals are in 2's complement format.

Analog In Word

All 12-bit instruments (DI-200, DI-400, DI-401, and DI-500 Series) with signal averaging on, or all 14-bit instruments (DI-210, DI-410, DI-700, DI-720 and DI-730).

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	data	data	data	data	data	data	data	data	data	data	data	data	LSB	Y1*	Y0*

D=indicated digital I/O line; *If you are looking at the first channel in the scan list, Y1 reflects the inverted state of the remote start/stop flag and Y0 reflects the inverted state of the event marker flag. Otherwise, Y1 and Y0 are zero. If analog data is a unipolar signal, the number is a straight binary value. Bipolar signals are in 2's complement format.

Analog In Word

All 16-bit instruments (DI-700, DI-720, and DI-730).

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	data	data	data	data	data	data	data	data	data	data	data	data	data	data	LSB

D=indicated digital I/O line; If analog data is a unipolar signal, the number is a straight binary value. Bipolar signals are in 2's complement format.

Analog Out Word

All instruments that have this capability (for example, all except the DI-401 and DI-700).

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
MSB	data	data	data	data	data	data	data	data	data	data	LSB	R	R	0	0

D=indicated digital I/O line; If analog data is a unipolar signal, the number is a straight binary value. Bipolar signals are in 2's complement format. R=reserved.

Digital In Word

All instruments.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X	X	Y1*	Y0*

D=indicated digital I/O line; X=don’t care; *If you are looking at the first channel in the scan list, Y1 reflects the inverted state of the remote start/stop flag and Y0 reflects the inverted state of the event marker flag. Otherwise, Y1 and Y0 are zero.

Digital Out Word

All instruments.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X	X	X	0	0

D=indicated digital I/O line; X=don’t care;

A/D Coding

A/D Input	Coding	A/D Output (Hex)** 12-bit	14-bit	16-bit
+FS	Unipolar*	7FF0	7FFC	7FFF
1/2FS		0	0	0
0		-8000	-8000	-8000
+FS	Bipolar	7FF0	7FFC	7FFF
0		0	0	0
-FS		-8000	-8000	-8000

FS = full scale; *DI-221TC, DI-400 Series, DI-500 Series, DI-700, DI-720, DI-730, and DI-5001 instruments do not support unipolar operation. **All data is left-justified, with extraneous bits (if any) equal to zero.

Converting Counts to Volts

Data is returned from the instrument in the form of A/D converter counts. These counts may be converted to volts for all instruments as follows:

$$\text{Volts} = \left(\text{Count} \times \frac{(V_{max} - V_{min})}{65536.0} \right) + \frac{(V_{max} + V_{min})}{2}$$

where: Count is a signed integer value returned from the instrument. All bits after the LSB should be zero.
V_{max} is the maximum input voltage accepted at the selected gain.
V_{min} is the minimum input voltage accepted at the selected gain.

Note that V_{max} , V_{min} , and gain factor vary by instrument. The following tables list each value of V_{max} and V_{min} by instrument and gain factor:

Instrument	Gain	Unipolar Mode		Bipolar Mode	
		V_{max}	V_{min}	V_{max}	V_{min}
DI-200, DI-210* & DI-222	1	10	0	+10	-10
	2	5	0	+5	-5
	4	2.5	0	+2.5	-2.5
	8	1.25	0	+1.25	-1.25
	10	1	0	+1	-1
	100	0.1	0	+0.1	-0.1
	1,000	0.01	0	+0.01	-0.01
DI-201 & DI-220	1	5	0	+5	-5
	2	2.5	0	+2.5	-2.5
	4	1.25	0	+1.25	-1.25
	8	0.625	0	+0.625	-0.625
	10	0.5	0	+0.5	-0.5
	100	0.05	0	+0.05	-0.05
	1,000	0.005	0	+0.005	-0.005
DI-221TC	1			+5	-5
	10			+0.5	-0.5
	100			+0.05	-0.05
	1,000			+0.005	-0.005
DI-400 & DI-410*	1			+10	-10
	2			+5	-5
	4			+2.5	-2.5
	8			+1.25	-1.25
	10			+1	-1
	100			+0.1	-0.1
DI-401	1			+5	-5
DI-700	1			+10	-10
	10			+1	-1
	100			+0.1	-0.1
	1,000			+0.01	-0.01
DI-720	1			+10	-10
	2			+5	-5
	4			+2.5	-2.5
	8			+1.25	-1.25
DI-730	1			+1,000	-1,000
	10			+100	-100
	100			+10	-10
	1,000			+1	-1
	10,000			+0.1	-0.1
	100,000			+0.01	-0.01
DI-5001**	1			+5 or +10	-5 or -10
	2			+2.5 or +5	-2.5 or –5
	4			+1.25 or +2.5	-1.25 or –2.5
	8			+0.625 or +1.25	-0.625 or –1.25

*DI-210 and DI-410 instruments only support gains of 1, 2, 4, and 8.

**DI-5001 instruments have a jumper-selectable input range feature that allows them to be configured for either ±5 volts full scale or ±10 volts full scale.

Instrument		Gain	Bipolar Mode V_{max}	V_{min}
All DI-500 Series instruments with signal conditioned inputs, such as:	DI-500-16, DI-510-32, DI-510-32 Expander, and Channels 1 thru 16 on the DI-510-48	1	+5	-5
		2	+2.5	-2.5
		4	+1.25	-1.25
		8	+0.625	-0.625
All DI-500 Series instruments with high level inputs, such as:	DI-500-32, DI-500-32 Expander, DI-510-64, DI-510-64 Expander, and Channels A1 thru A32 on the DI-510-48	1	+10	-10
		2	+5	-5
		4	+2.5	-2.5
		8	+1.25	-1.25

Refer to the following examples:

Example 1: Say we are using a DI-200 board, configured for a gain of 1, operating in bipolar mode. From the chart, V_{max} = +10V and V_{min} = -10V. Plugging these values into the above equation:

$$\begin{aligned}\text{Volts} &= \left(\text{Count} \times \frac{(10-(-10))}{65536}\right) + \frac{(10+(-10))}{2} \\ &= \text{Count} \times \frac{20}{65536}\end{aligned}$$

Example 2: Say we are using a DI-200 board, configured for a gain of 1, operating in unipolar mode. From the chart, V_{max} = 10V and V_{min} = 0V. Plugging these values into the above equation:

$$\begin{aligned}\text{Volts} &= \left(\text{Count} \times \frac{(10-0)}{65536}\right) + \frac{(10+0)}{2} \\ &= \left(\text{Count} \times \frac{10}{65536}\right) + 5\end{aligned}$$

Example 3: Say we are using a DI-200 board, configured for a gain of 2, operating in bipolar mode. From the chart, V_{max} = +5V and V_{min} = -5V. Plugging these values into the above equation:

$$\begin{aligned}\text{Volts} &= \left(\text{Count} \times \frac{(5-(-5))}{65536}\right) + \frac{(5+(-5))}{2} \\ &= \text{Count} \times \frac{10}{65536}\end{aligned}$$

For DI-500 Series instruments, this equation converts the data returned into volts, just like any other instrument. If you have a DI-500 Series instrument with high level inputs (i.e., DI-500-32, DI-500-32 expander, DI-510-64, DI-510-64 expander, or channels A1 through A32 of the DI-510-48), this equation can be used to convert the returned data into volts. However, if you have a DI-500 Series instrument with signal conditioned inputs (i.e., DI-500-16, DI-510-32, DI-510-32 expander, or channels 1 through 16 of the DI-510-48), an additional calculation must be made to convert volts to whatever meaningful units the DI-5B module is measuring.

Sampling Different Channels at Different Rates

Except for DI-300 Series instruments, all of our instruments have the unique capability of allowing analog and digital I/O data to be written or acquired at a different rate per channel. This feature is possible through the use of a counter attached to each input and output scan list element (a total of 272 counters exist). The number loaded into the counter defines the rate at which that scan element will read or write data according to the following equation:

$$S = \frac{B}{L(C+1)}$$

Where: S = desired sampling rate of the input list entry, B = burst rate of the instrument, L = length of the input **or** output list **(whichever is greater)**, and C = "count weight" or input counter list entry (the value represented by *i* in the command format).

The effect of counter values on the placement of data in the input and output buffers is significant, and deserves special treatment. We will examine the use of data buffers in this special case by applying several examples of analog input operations at varying rates per channel.

To keep the examples manageable, we will set the burst rate at 1000 samples per second, and fix the length of the scan list to four elements. In practice, you would want to set the burst rate as high as possible to minimize time skew between channels. Setting a lower burst rate for the sake of the example, however, allows a small change in the counter value to translate into a large change in sample rate. The length of the input buffer will also be fixed at twenty samples.

We will look at three examples: The first will set the counter value to zero for all four channels; the second will apply the same, but non-zero counter value to all channels; the third will apply a different counter value per channel.

All Counter Values Equal Zero
This is the most common operating mode of data acquisition products where the sample throughput rate of the instrument is divided equally among the enabled channels. In our example of a burst rate of 1000Hz and four enabled channels, each channel is sampled at a rate of 250Hz.

Channel Number	Counter Value	Sample Rate (Hz)
0	0	250
1	0	250
2	0	250
3	0	250

When executed, A/D values will appear in the input buffer in the order of lowest to highest as follows:

Input Buffer Position	Acquired Channel #	Input Buffer Position	Acquired Channel #
0	0	10	2
1	1	11	3
2	2	12	0
3	3	13	1
4	0	14	2
5	1	15	3
6	2	16	0
7	3	17	1
8	0	18	2
9	1	19	3

All instruments sample A/D data in the burst mode of operation. In the example we've defined above, five bursts of A/D conversions result in a total of twenty samples delivered to the input buffer as follows:

Burst Number	Channel Number 0	1	2	3
1	•	•	•	•
2	•	•	•	•
3	•	•	•	•
4	•	•	•	•
5	•	•	•	•

• means the indicated channel number was acquired during the burst

Counter Values Equal, but Non-Zero

Adjusting counter values to equal but non-zero values has the effect of simply adjusting the sample rate at which the analog data is acquired. For example, setting the counter value to three has the following effect on sample rates…

Channel Number	Counter Value	Sample Rate (Hz)
0	3	62.5
1	3	62.5
2	3	62.5
3	3	62.5

…but does not affect the order in which channel data is stored in the input buffer:

Input Buffer Position	Acquired Channel #	Input Buffer Position	Acquired Channel #
0	0	10	2
1	1	11	3
2	2	12	0
3	3	13	1
4	0	14	2
5	1	15	3
6	2	16	0
7	3	17	1
8	0	18	2
9	1	19	3

However, looking at how the data is acquired in relation to the burst number shows a clearly slower sample rate. The burst rate is fixed at 1000Hz and the counters are decremented at that rate. When a counter passes through zero, a sample is acquired for that channel, the counter is reset to its initial value (3 in this example), and data acquisition resumes. Since the counters for all channels are set to the same value, sampling occurs for each on the same burst number.

Burst Number	Channel Number 0	1	2	3
1				
2				
3				
4	•	•	•	•
5				
6				
7				
8	•	•	•	•
9				
10				
11				
12	•	•	•	•
13				
14				
15				
16	•	•	•	•
17				
18				
19				
20	•	•	•	•

• means the indicated channel number was acquired during the burst

Different Counter Values

So far, there has been little differentiation from other alternative products regarding sample rate selection. But with the ability to apply a different count value per element, the sample rate of each channel may vary. This adds significant flexibility to your data acquisition tasks. Selecting a count value of 0, 1, 2, and 3 to channels 0 through 3 respectively yields the following sample rates per channel:

Channel Number	Counter Value	Sample Rate (Hz)
0	0	250
1	1	125
2	2	83.33
3	3	62.5

When acquired, the order of channels appearing in our 20-sample input buffer is as follows:

Input Buffer Position	Acquired Channel #	Input Buffer Position	Acquired Channel #
0	0	10	1
1	0	11	2
2	1	12	0
3	0	13	0
4	2	14	1
5	0	15	3
6	1	16	0
7	3	17	2
8	0	18	0
9	0	19	1

When broken down by burst sample number, the following scan order is revealed. Clearly demonstrating the varying sample rates per acquired channel:

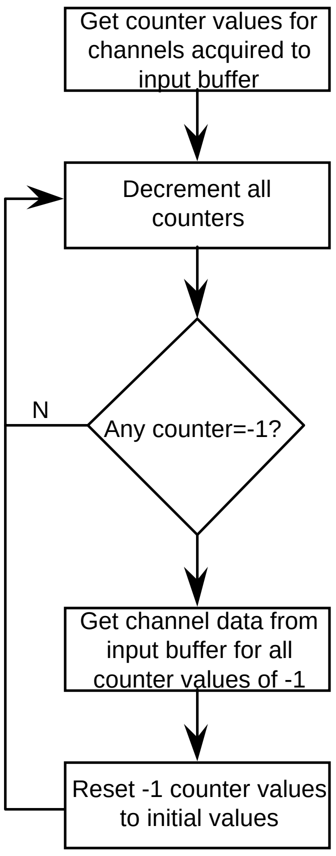
	Channel Number			
Burst Number	**0**	**1**	**2**	**3**
1	•			
2	•	•		
3	•		•	
4	•	•		•
5	•			
6	•	•	•	
7	•			
8	•	•		•
9	•		•	
10	•	•		

• means the indicated channel number was acquired during the burst

Note also by examining the above table that the number of burst numbers separating each acquisition of a particular channel is constant. This means that each sample of any channel occurs at a precise and predictable moment in time.

How Data is Received From an Input Buffer

The following flow chart illustrates how data is received from an input data buffer (all instruments):


Get counter values for channels acquired to input buffer
Decrement all counters
N
Any counter=-1?
Get channel data from input buffer for all counter values of -1
Reset -1 counter values to initial values

Function Reference

Programmer’s SDK Functions

Each function may be classified into one of the following categories:

- Initialization and Information functions
- Buffer functions
- Immediate functions (one-shot, single data value)
- Scanning functions (collecting multiple data values)
- Counter/timer functions
- Miscellaneous functions

Hardware Support for Programmer’s SDK Functions

A checkmark indicates the function is supported by the instrument:

Function	Instrument								
	DI-200, DI-201, DI-210	DI-220 DI-222	DI-221TC	DI-401	DI-400 DI-410	DI-500	DI-510	DI-700	DI-720 DI-730 DI-5001
di_anin	✔	✔	✔	✔	✔	✔	✔		✔
di_anout	✔	✔	✔		✔	✔	✔		✔
di_buffer_alloc	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_buffer_free	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_buffer_size	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_buffer_status	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_burst_rate	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_close	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_copy_array	✔	✔	✔		✔	✔	✔	✔	✔
di_copy_buffer	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_copy_header	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_copy_mux					✔	✔	✔		✔
di_ct_event	✔	✔	✔						
di_ct_one_shot	✔	✔	✔						
di_ct_status	✔	✔	✔						
di_ct_stop	✔	✔	✔						
di_ct_wave	✔	✔	✔						
di_digin	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_digout	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_get_acq_header	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_info	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_inlist	✔	✔	✔	✔	✔	✔	✔	✔*	✔
di_list_length	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_mode	✔	✔		✔	✔	✔	✔		✔
di_open	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_outlist	✔	✔	✔		✔	✔	✔		✔
di_set_data_mode								✔	
di_start_scan	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_status_read	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_stop_scan	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_strerr	✔	✔	✔	✔	✔	✔	✔	✔	✔
di_trigger_status	✔	✔	✔	✔	✔	✔	✔		✔

*only partially supported.

Initialization and Information Functions

Use these functions for initializing communications with your hardware:

- `di_close` Closes communications with your instrument and frees all opened buffers.
- `di_copy_mux` Describes each bank of 16 channels on DI-400, DI-500 Series, DI-720, and DI-730 instruments.
- `di_info` Returns instrument-specific information such as base address, interrupt levels, revision levels, etc.
- `di_open` Opens communications with your instrument.

Buffer Functions

Use these functions for manipulating input or output data buffers:

- `di_buffer_alloc` Allocates buffer memory.
- `di_buffer_free` Frees buffer memory.
- `di_buffer_size` Returns the size of the input buffer to determine the proper index.
- `di_buffer_status` Returns the position of the next entry into the input or output data buffer.
- `di_copy_array` Copies the contents of a BASIC array into the output buffer.
- `di_copy_buffer` Copies the contents of the input buffer into a BASIC array.
- `di_status_read` Reads the status of the input buffer and copies the newest data (collected since a previous call) to a specified destination.

Immediate Functions

Use these functions for one-shot, single data value operations:

- `di_anin` Reads an analog input channel and returns the value.
- `di_anout` Writes a binary value to the DAC.
- `di_digin` Reads a byte from the digital input port and returns the value.
- `di_digout` Writes a byte to the digital output port.

Scanning Functions

Use these functions for collecting multiple data values:

- `di_burst_rate` Sets the burst rate.
- `di_inlist` Initializes the input scan list.
- `di_list_length` Sets input and output scan list lengths.

• `di_mode`	Initializes the instrument for triggering and sets the triggering mode.
• `di_outlist`	Initializes the output scan list.
• `di_start_scan`	Initiates a multiple-channel data acquisition scanning operation.
• `di_stop_scan`	Stops the data acquisition scanning operation.
• `di_trigger_status`	Returns the trigger status.

Counter/Timer Functions

Use these functions to perform timing I/O and counter operations:

• `di_ct_event`	Starts event counting using the digital input bits.
• `di_ct_one_shot`	Generates a one-shot function according to the parameters passed to it.
• `di_ct_status`	Returns the present count of the event counter.
• `di_ct_stop`	Stops event counting.
• `di_ct_wave`	Generates a square wave according to the parameters passed to it.

Miscellaneous Functions

Use these functions to perform miscellaneous operations:

• `di_copy_header`	Copies CODAS header information into a BASIC structure.
• `di_get_acq_header`	Returns either a far pointer to the CODAS header structure or null if the structure is not available.
• `di_set_data_mode`	DI-700 Only. Chooses between 14- and 16-bit measurement resolution.
• `di_strerr`	Maps an error code to an error message.

Equivalent HP VEE Functions

The HP VEE function library contains functions that accomplish the same result as SDK functions but aren't precisely named the same way as the SDK functions and therefore may not be intuitively obvious. The following chart should remove all doubt regarding the SDK functions and their HP VEE equivalents.

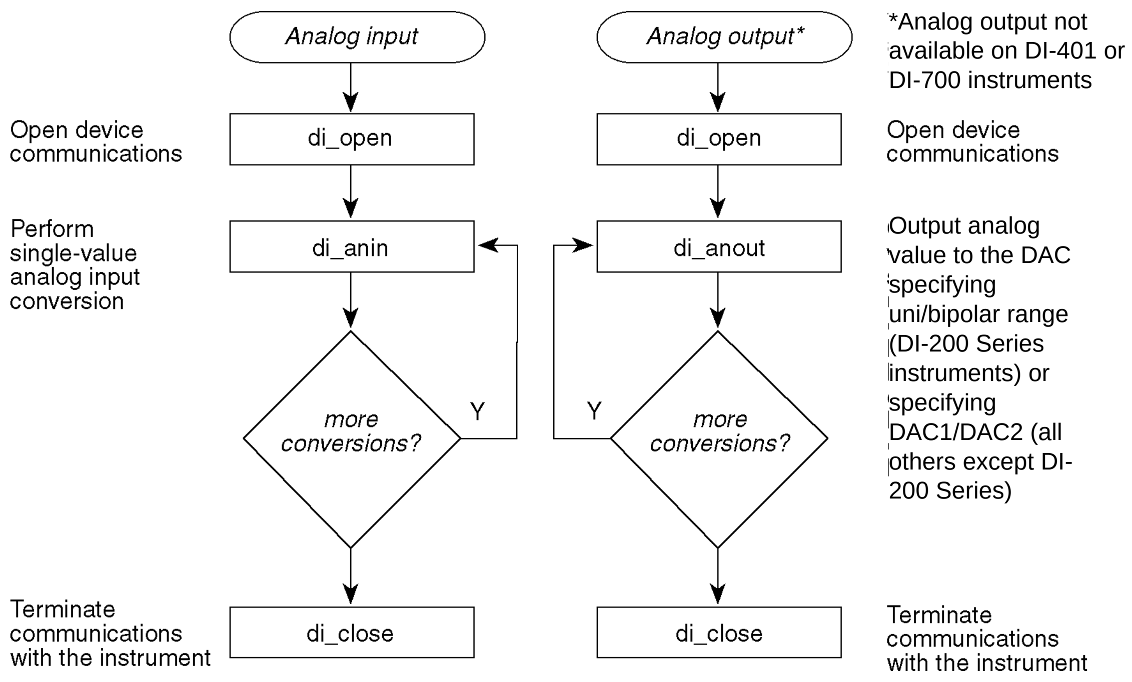
HP VEE Function	SDK Equivalent:
DiAnIn	di_anin
DiAnOut	di_anout
DiBufAlloc	di_buffer_alloc
DiBufFree	di_buffer_free
DiBufSize	di_buffer_size
DiBufStatus	di_buffer_status
DiBurstRate	di_burst_rate
DiClose	di_close
DiCopyArray	di_copy_array
DiCopyBuffer	di_copy_buffer
DiCopyHeader	di_copy_header
DiCopyMux	di_copy_mux
DiCtEvent	di_ct_event
DiCtOneShot	di_ct_one_shot
DiCtStatus	di_ct_status
DiCtStop	di_ct_stop
DiCtWave	di_ct_wave
DiDigIn	di_digin
DiDigOut	di_digout
DiGetAcqHeader	di_get_acq_header
DiInfoBoardID	di_info
DiInfoDspVer	di_info
DiInfoGain	di_info
DiInfoHdrLvl	di_info
DiInfoHrdwrRev	di_info
DiInfoInBufSize	di_info
DiInfoInChan	di_info
DiInfoLastCalDate	di_info
DiInfoOutBufSize	di_info
DiInfoOutChan	di_info
DiInfoPal0Rev	di_info
DiInfoPal1Rev	di_info
DiInfoPort	di_info
DiInfoSDKVer	di_info
DiInfoSerialNo	di_info
DiInfoSftLvl	di_info
DiInfoTsrVer	di_info
DiInList	di_inlist
DiListLength	di_list_length
DiMode	di_mode
DiOpen	di_open
DiOutList	di_outlist
DiSetDataMode	di_set_data_mode
DiStartScan	di_start_scan
DiStatusRead	di_status_read
DiStopScan	di_stop_scan
DiStrErr	di_strerr
DiTriggerStatus	di_trigger_status

Programming Sequences and Flow Charts

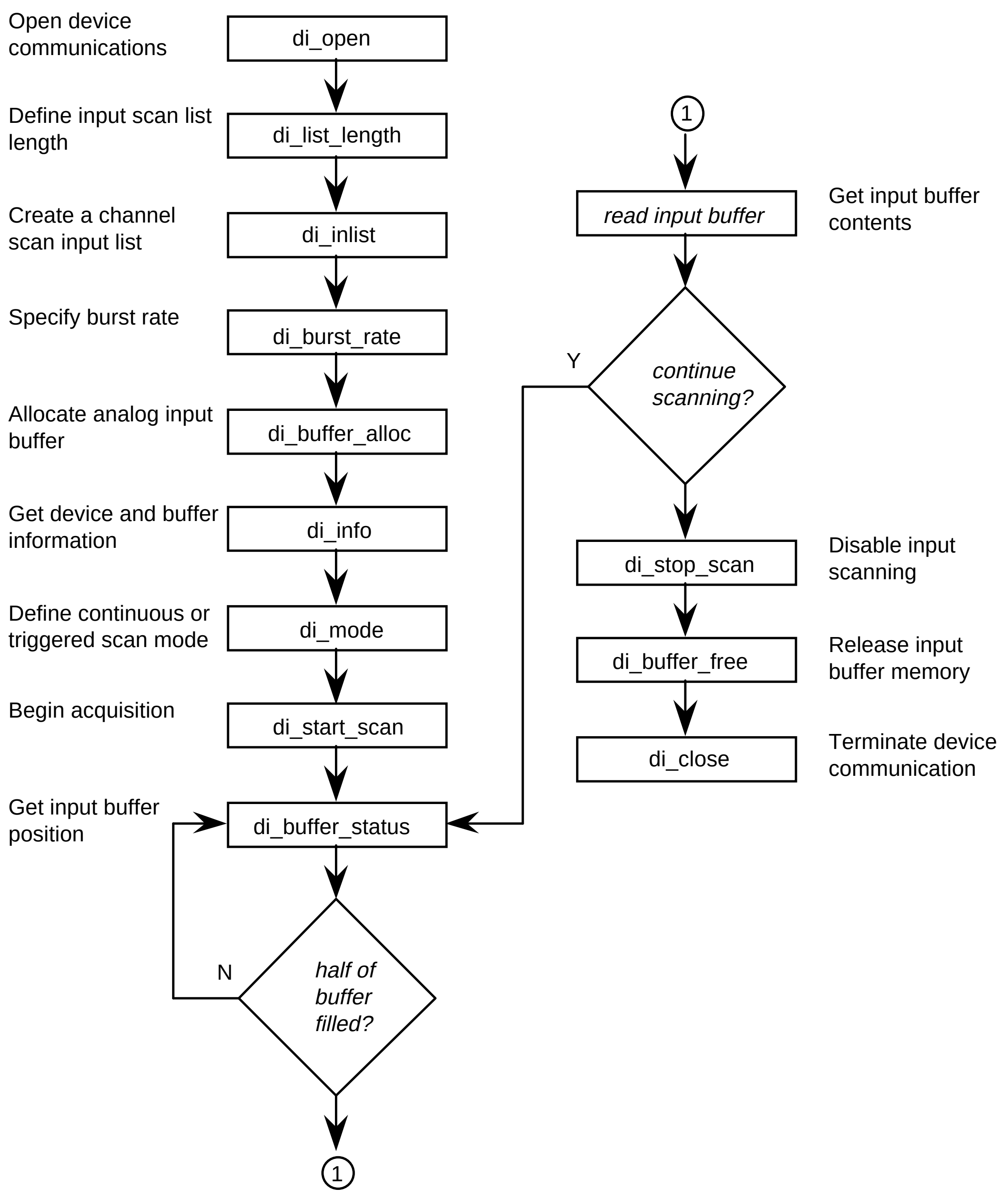
The programming sequences in this manual are provided in flow chart format. Each flow chart illustrates a typical sequence of function calls needed to achieve the following operations:

- Single value analog input and analog output
- Multiple value analog input and/or digital I/O
- Multiple value analog output and/or digital output
- Multiple value simultaneous analog I/O and digital I/O
- Single value digital input and digital output
- Counter/Timer—event counting and one-shot generation
- Counter/Timer—output square wave generation

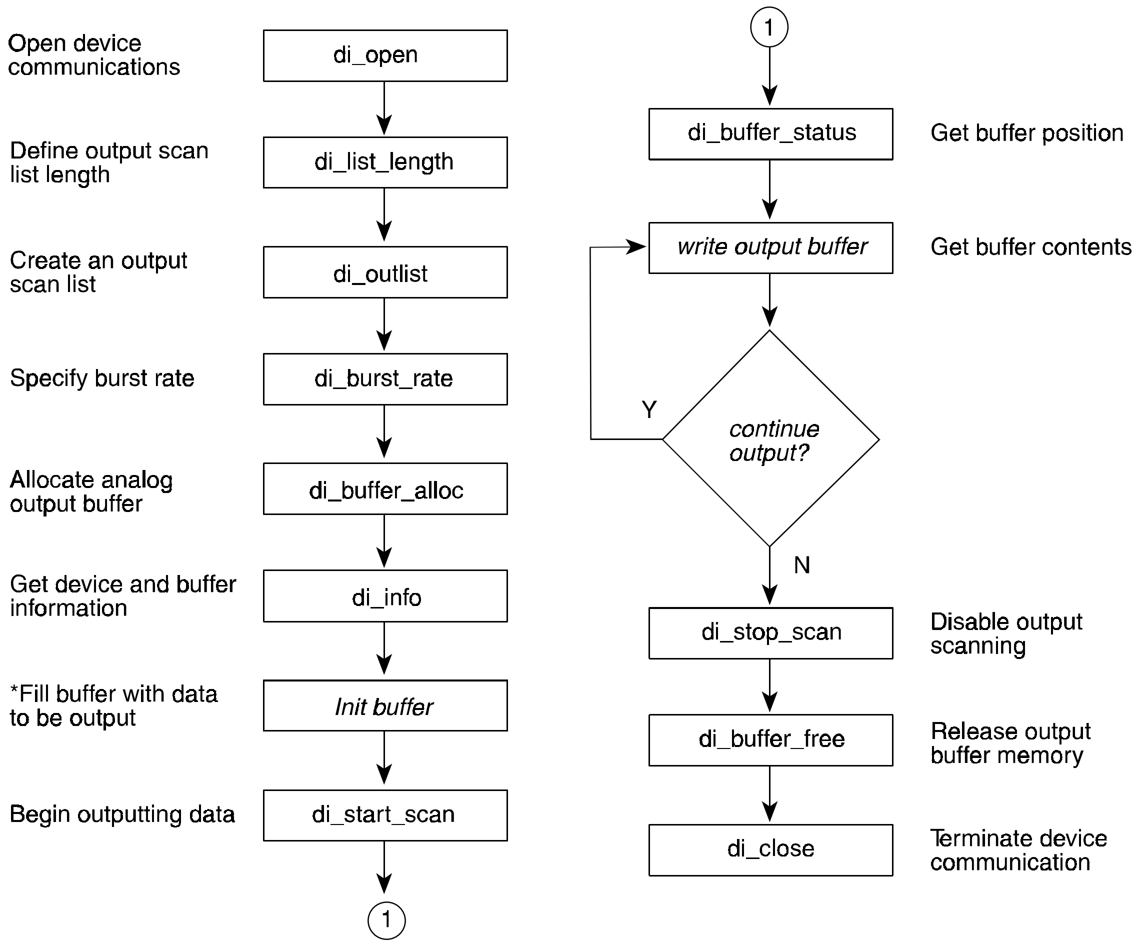
Analog I/O—Single-value Analog Input and Analog Output (Analog Output not available on DI-401 and DI-700 Instruments)


Analog input
Analog output*
*Analog output not available on DI-401 or DI-700 instruments
Open device communications
di_open
di_open
Open device communications
Perform single-value analog input conversion
di_anin
di_anout
Output analog value to the DAC specifying uni/bipolar range (DI-200 Series instruments) or specifying DAC1/DAC2 (all others except DI-200 Series)
more conversions?
Y
Y
more conversions?
Terminate communications with the instrument
di_close
di_close
Terminate communications with the instrument

Analog I/O—Multiple-value Analog Input and/or Digital Input

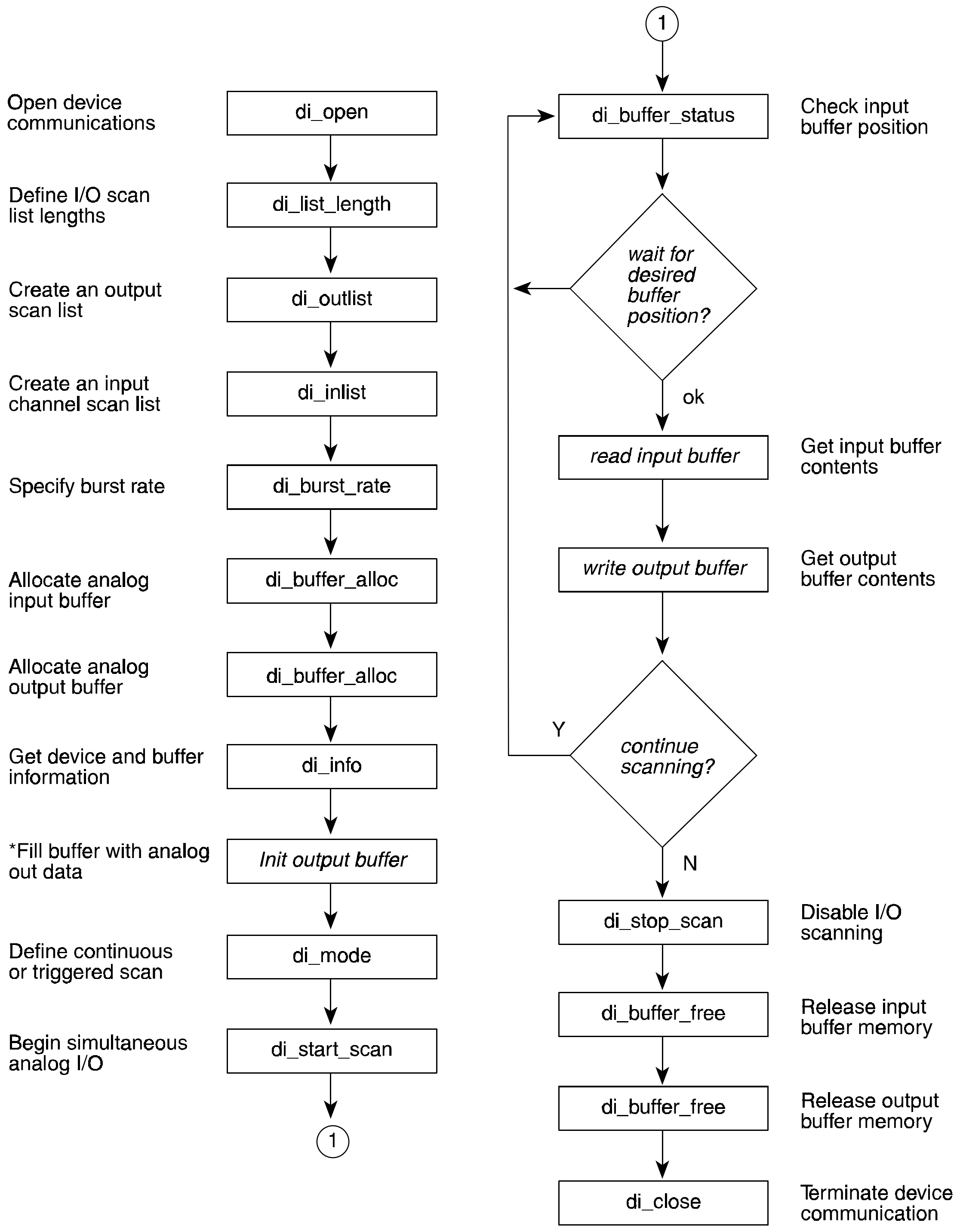

Open device communications
di_open
Define input scan list length
di_list_length
Create a channel scan input list
di_inlist
Specify burst rate
di_burst_rate
Allocate analog input buffer
di_buffer_alloc
Get device and buffer information
di_info
Define continuous or triggered scan mode
di_mode
Begin acquisition
di_start_scan
Get input buffer position
di_buffer_status
N
half of buffer filled?
1
1
read input buffer
Get input buffer contents
Y
continue scanning?
di_stop_scan
Disable input scanning
di_buffer_free
Release input buffer memory
di_close
Terminate device communication

Analog I/O—Multiple-value Analog Output (Analog Output not available on DI-401 and DI-700 Instruments) and/or Digital Output



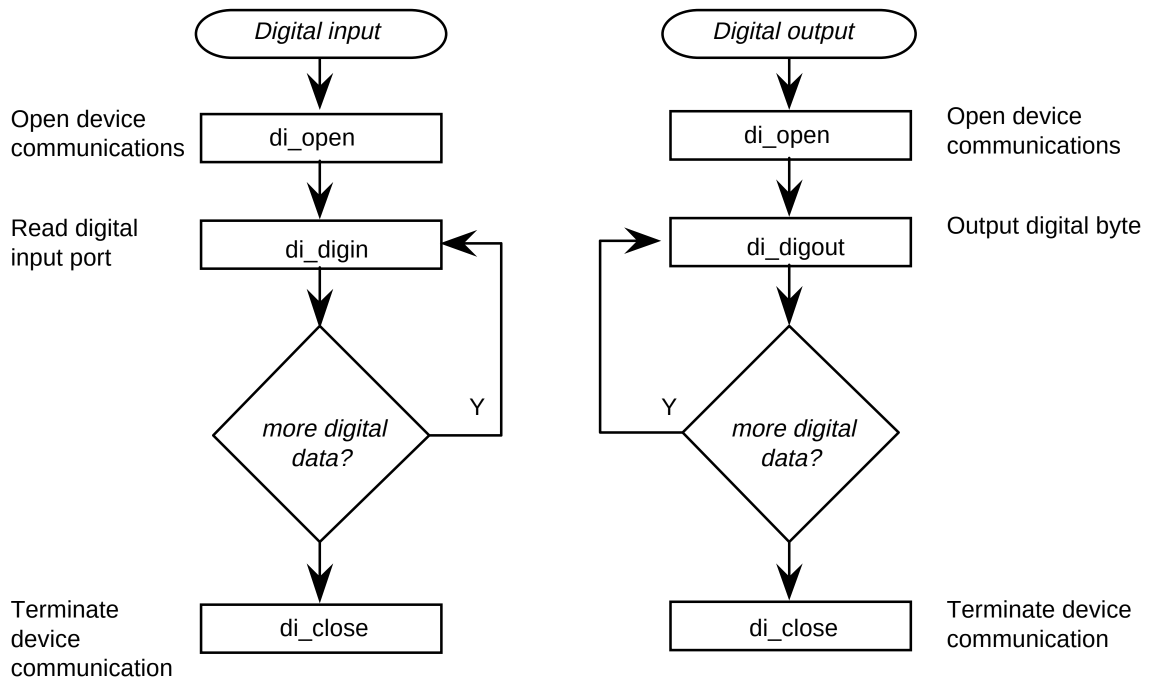
*The two least significant bits of the data word *must* be zero.

Analog I/O—Multiple Value, Simultaneous Analog I/O (Analog Output not available on DI-401 and DI-700 Instruments) and Digital I/O

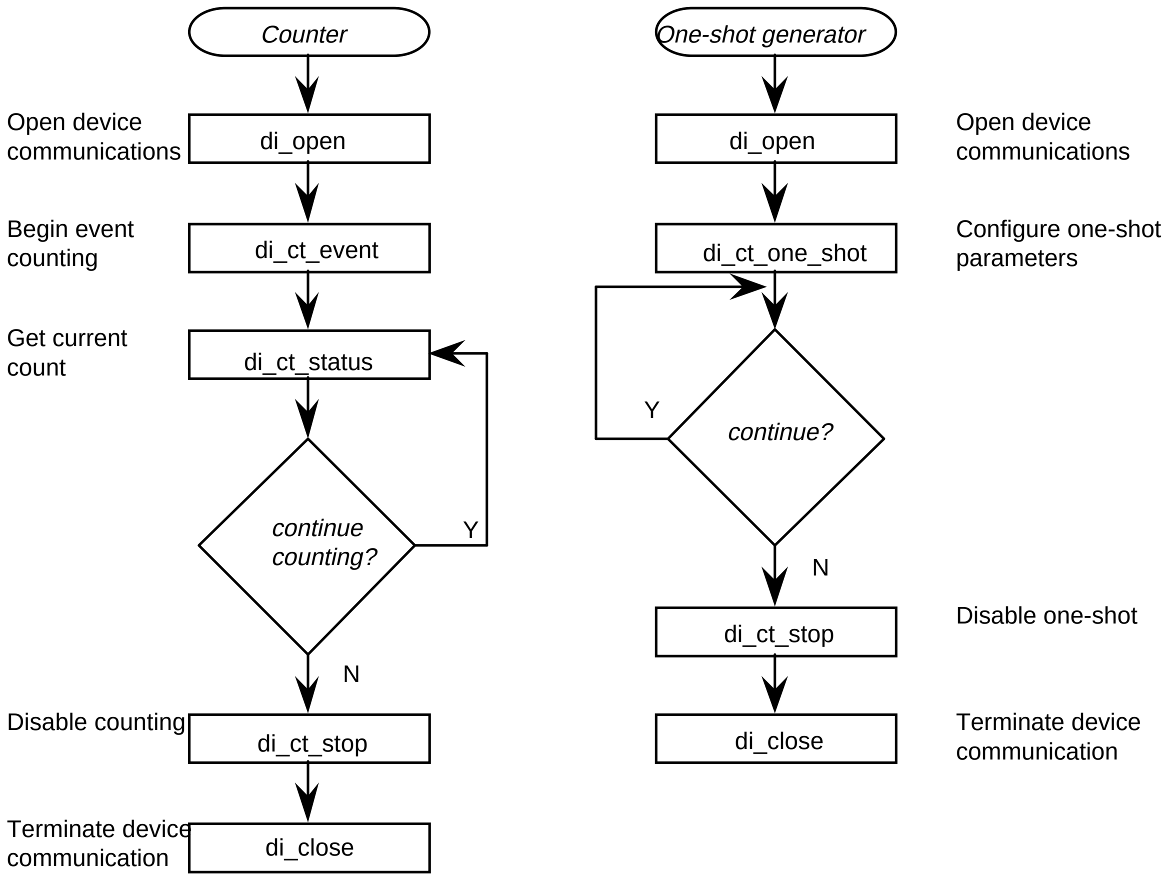

Open device communications
di_open
Define I/O scan list lengths
di_list_length
Create an output scan list
di_outlist
Create an input channel scan list
di_inlist
Specify burst rate
di_burst_rate
Allocate analog input buffer
di_buffer_alloc
Allocate analog output buffer
di_buffer_alloc
Get device and buffer information
di_info
*Fill buffer with analog out data
Init output buffer
Define continuous or triggered scan
di_mode
Begin simultaneous analog I/O
di_start_scan
1
1
di_buffer_status
Check input buffer position
wait for desired buffer position?
ok
read input buffer
Get input buffer contents
write output buffer
Get output buffer contents
continue scanning?
Y
N
di_stop_scan
Disable I/O scanning
di_buffer_free
Release input buffer memory
di_buffer_free
Release output buffer memory
di_close
Terminate device communication

*The two least significant bits of the data word *must* be zero.

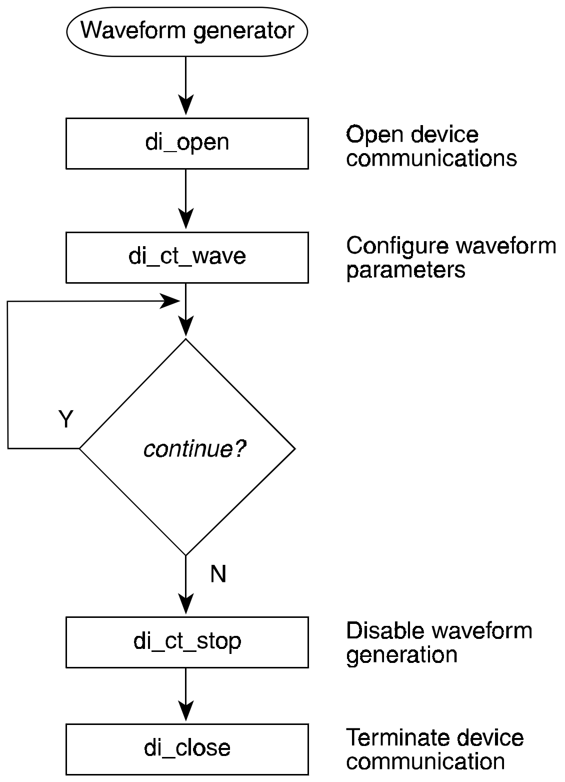
Digital I/O—Single-value Digital Input and Digital Output



Counter/Timer—Event Counting and One-shot Generation (Counter/Timer functions not available on DI-400 Series, DI-500 Series, DI-700, DI-720, DI-730 or DI-5001 Instruments)



Counter/Timer—Output Square Wave Generation (Counter/Timer functions not available on DI-400 Series, DI-500 Series, DI-700, DI-720, DI-730 or DI-5001 Instruments)


Waveform generator
di_open
Open device communications
di_ct_wave
Configure waveform parameters
Y
continue?
N
di_ct_stop
Disable waveform generation
di_close
Terminate device communication

Function Reference

The remainder of this chapter is an alphabetically arranged listing of each function in the SDK.

di_anin

(not available on DI-700 instruments)

• Summary

```
int di_anin(analog_input);
```

```
struct di_anin_struct{
   unsigned chan;       /* input channel. 0 to 255 */
   unsigned diff;       /* input configuration (single-ended or diff) */
   unsigned gain;       /* gain. 0 to 3 */
   unsigned unipolar;   /* unipolar/bipolar; 0=bipolar, 1=unipolar */
   }*analog_input;
```

• Description

di_anin is an immediate function that inputs data from an analog input channel. On DI-200 and DI-210 instruments, this function should not be issued while an input, output, or simultaneous input and output scanning operation is in progress. If it is, the scanning operation will halt. On all other instruments, scanning will not halt when this function is issued, but **di_anin** will only execute successfully while output scanning (the **di_anin** function will be ignored when input scanning or simultaneously input and output scanning). Each element in the structure is defined as follows:

`chan` allows you to specify the input channel you wish to sample. Values for `chan` can range from 0 to 255, according to the following equations:

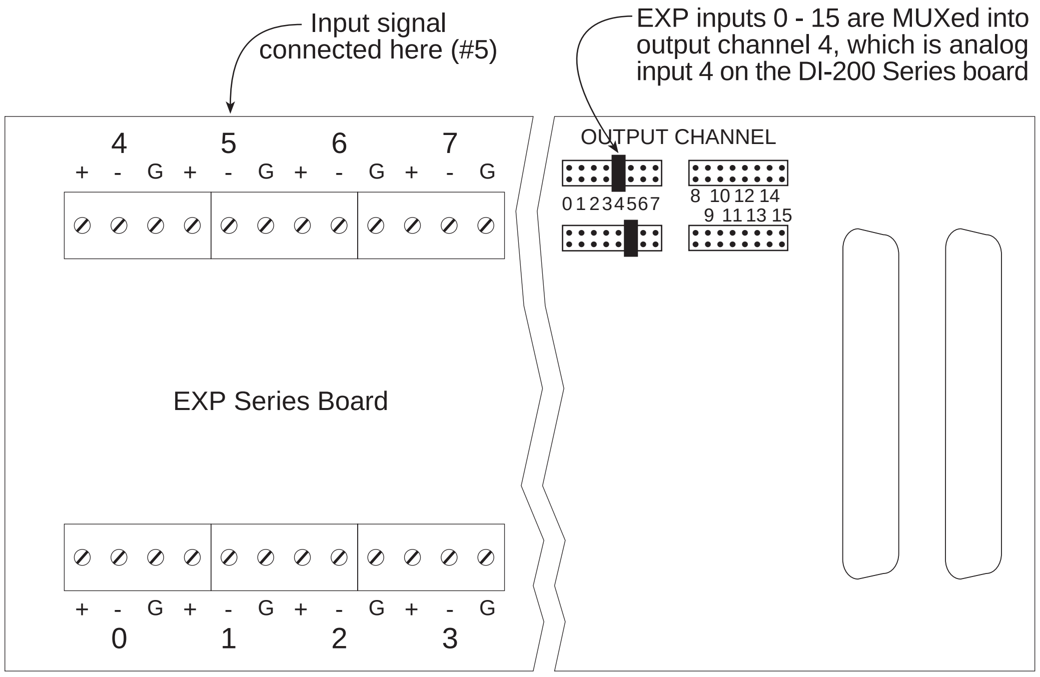
<u>DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222</u>

With 16 channels or less: `chan = channel#`

With more than 16 channels:

```
chan = channel# mod 16 + 16(output channel + 1)
```

`channel#` is the analog input channel on the EXP board you wish to sample. `output channel` is the position of the jumper on the EXP board. On hardware with more than 16 channels, the `mod` operator in the equation above combines two integer expressions using modulo arithmetic. For two integer values, modulo arithmetic returns only the *remainder* from an integer division. That is, 6 mod 4 is 2, the remainder of the integer division of 6 by 4. For example, let's say we have three 32-channel EXP boards multiplexed to a DI-200 Series board and we want to record analog input channel five on the third EXP board. What channel do we specify for `chan` ?


Input signal connected here (#5)
EXP inputs 0 - 15 are MUXed into output channel 4, which is analog input 4 on the DI-200 Series board
4
5
6
7
+ - G + - G + - G + - G
OUTPUT CHANNEL
0 1 2 3 4 5 6 7
8 10 12 14
9 11 13 15
EXP Series Board
+ - G + - G + - G + - G
0
1
2
3

On each EXP board, each bank of 16 analog inputs is multiplexed into one output (specified by the position of the OUTPUT CHANNEL jumper on the EXP board). This output from the EXP board is connected to an *internal* analog input on the DI-200 Series board. Assuming the first EXP board uses output channels 0 and 1 and the second EXP board uses outputs 2 and 3, the third EXP board will multiplex inputs 0 through 15 into output channel 4 and inputs 16 through 31 into output channel 5. From the equation:

```
chan = (channel#) mod 16 + 16(output channel +1)
chan = (5) mod 16 + 16(4 + 1)
chan = 5 + 80
chan = 85
```

Now suppose we want to record channel 23 on the third EXP board. What channel do we specify for `chan` in this case?

```
chan = (channel#) mod 16 + 16(output channel +1)
chan = (23) mod 16 + 16(5 + 1)
chan = 7 + 96
chan = 103
```

Finally, suppose we have just one DI-200 Series board (not multiplexed) and we want to record channel 6. What channel do we specify for `chan`? Since we are using non-multiplexed hardware (only 16 channels), the first equation applies:

```
chan = channel#
chan = 6
```

<u>DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001</u>

With 16 channels or less: `chan = (channel# - 1)`

With more than 16 channels:

```
chan = (channel# - 1) + 32(mux letter - 'A' + 1)
```

`channel#` is the analog input channel you wish to sample. On instruments with more than 16 channels, `mux letter` is the letter you assigned to the instrument (written on the overlay) during installation and initial configuration. This letter is important for keeping track of all the analog input channels on your instrument(s). For example, let's say we have three DI-500-32-P instruments multiplexed together and we want to record analog input channel five on the third instrument. What channel do we specify for `chan`? During installation, you would have labeled the instruments "A", "B", and "C". From the equation:

```
chan = (channel# - 1) + 32(mux letter - 'A' + 1)
chan = (5 - 1) + 32(C - A + 1)
chan = 4 + 32(2 + 1)
chan = 4 + 96
chan = 100
```

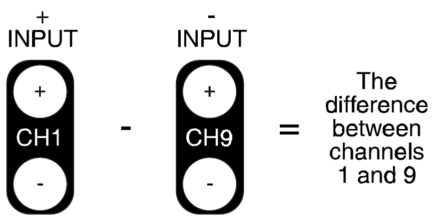
`diff` allows you to specify whether the channel specified by `chan` is single-ended or differential as follows:

<u>DI-200, DI-201, DI-210, DI-220, DI-221TC, DI-222, DI-400, DI-410, DI-720, and DI-5001</u>
`diff = 0` for single ended input configuration.
`diff = 1` for differential input configuration (do not set `diff` = 1 for channels on multiplexers, even though the inputs are differential).

<u>DI-401</u>
reserved for compatibility.

<u>DI-500 and DI-510</u>
`diff = 0` for single ended input configuration.
`diff = 1` for differential input configuration. This configuration allows you to see the difference between 2 differential input channels, but only on `chan` 1 through 8 and 17 through 24 of DI-500-16 and DI-510-48 instruments. When `chan` 1 through 8 or 17 through 24 is specified for differential operation, the other channel that creates the differential pair is automatically selected, eight channels away. For example, if `chan` 1 is configured for differential operation, `chan` 9 becomes the companion channel, similarly with 2 and 10, 8 and 16, 18 and 26, etc. In every case, the lowest channel number becomes the positive (+) differential

input and the automatically selected channel becomes the negative (-) differential input.



DI-730

`diff = 0` always. DI-730 instruments are always differential.

`gain` allows you to specify a gain factor (assigned to a code, from the following table) for the channel specified by `chan`. With the DI-221TC, you can specify a gain factor for linear or non-linear inputs. Note that this structure element is reserved for compatibility on DI-401 instruments (gain is fixed at 1):

Code	DI-200PGH, DI-201PGH, DI-210, DI-220PGH, DI-222PGH, DI-400PGH, DI-410, DI-500PGH, DI-510PGH, DI-720, DI-5001 Gain	DI-200PGL, DI-201PGL, DI-220PGL, DI-222PGL, DI-500PGL, DI-510PGL Gain	DI-400PGL Gain	DI-221TC Only Gain	DI-221TC Only Input Type	DI-730 Gain
0	1	1	1	1		100 (10Vfs)
1	2	10	10	10	Linear	1,000 (1Vfs)
2	4	100	100	100		10,000 (0.1Vfs)
3	8	1,000		1,000		100,000 (0.01Vfs)
4*				1		1 (1,000Vfs)
5*				10	Non-linear 0†	10 (100Vfs)
6‡				100		
7‡				1,000		
8*				1		
9*				10	Non-linear 1†	
10				100		
11				1,000		
12*				1		
13*				10	Non-linear 2†	
14				100		
15				1,000		
16				Reserved	Reserved	
17				CJC	†	

*Gain codes 4, 5, 8, 9, 12, and 13 are undefined when making thermocouple measurements.

Vfs = volts full scale.

†If you are specifying a gain code from non-linear input groups 0, 1, or 2 and you are measuring with a thermocouple, you must add an additional channel to the input scan list. This additional

channel must specify cold junction compensation (code 17) as the gain element and it should be placed last in the input scan list.

Thermocouple type is defined by the value of the `unsigned chan` variable of the last scan list element containing gain code 17:

`unsigned chan` value	TC type	Multiplier for gain of 100	Multiplier for gain of 1000
0	K	1232.3333	121.9772
1	J	760	94.94897
2	T	400	115.2563
3	R	1768	548.1296

‡The A/D converter delivers "counts" as an end result instead of degrees. These counts can be scaled to °C as follows:

$$\left(\frac{x}{32768}\right)\times(multiplier) = ^{\circ}C$$

where x = ADC counts as delivered by the hardware, and *multiplier* is from the table above. Make sure you use the appropriate multiplier for the selected gain.

`unipolar` allows you to specify whether the channel specified by `chan` is a unipolar or bipolar signal. On DI-401 instruments, this structure element is reserved for compatibility. On DI-400, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001 instruments, unipolar configuration is not supported. Therefore on these instruments, `unipolar` must be `0`.

• Return Value

The function returns the analog input value as a left justified, 12-bit number (all instruments except DI-210, DI-410, DI-720, DI-730, and DI-5001), as a left justified, 14-bit number (DI-210, DI-410, DI-720, DI-730, and DI-5001), or as a left justified, 16-bit number (DI-720, DI-730, and DI-5001 only).

• Dependencies

di_open

• Example

```
#include "200sdk.h"

int errcode;
struct di_anin_struct anin;
char errstr[255];

main()
```

```
{
int i;
   if(errcode = di_open()){         /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   anin.chan = 5;                   /* channel 5 */
   anin.diff = 0;                   /* single ended input channel */
   anin.gain = 0;                   /* gain of 1 */
   anin.unipolar = 0;               /* bipolar */
   i = di_anin(&anin))           /* get analog input */
   printf("Channel 5 = %d",i);
   di_close ();
}
```

di_anout

(not available on DI-401 and DI-700 instruments)

- **Summary**

`int di_anout(dac_data,range)`

```
unsigned dac_data;   /* value to output to the DAC */
unsigned range;      /* range (as follows):
```

DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222	DI-400, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001
0 = bipolar 1 = unipolar	0 = DAC1 1 = DAC2 */

- **Description**

di_anout is an immediate function that writes (outputs) a 2's complement, left justified value to the DAC. The value that gets written to the DAC is resolution/instrument dependent as follows:

12-bit — all instruments except DI-210, DI-410, DI-720, DI-730, and DI-5001.
14-bit — DI-210, DI-410, DI-720, DI-730, and DI-5001.
16-bit — DI-720, DI-730, and DI-5001 only.

Only DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222 instruments support unipolar operation. Therefore on all other instruments, `range` specifies which DAC to write to.

DI-200 and DI-222 Only

Value (in Hex)	`range` = 0 (bipolar)	`range` = 1 (unipolar)
8000	-10	0
0000	0	5.00
7FF0	+9.9951	+9.9976

DI-210 Only

Value (in Hex)	`range` = 0 (bipolar)	`range` = 1 (unipolar)
8000	-10	0
0000	0	5.00
7FFC	+9.9988	+9.9988

DI-201, DI-220, and DI-221TC*

Value (in Hex)	`range` = 0 (bipolar)	`range` = 1 (unipolar) *
8000	-5	0
0000	0	2.50
7FF0	+4.9976	+4.9988

*unipolar mode not supported on DI-221TC instruments

DI-400, DI-500 Series, DI-720, DI-730, and DI-5001

Value (in Hex)	Bipolar only
8000	-10
0000	0
7FF0	+9.9976

DI-410 Only

Value (in Hex)	Bipolar only
8000	-10
0000	0
7FFC	+9.9988

On DI-200 and DI-210 instruments, this function should not be issued while an input, output, or simultaneous input and output scanning operation is in progress. If it is, the scanning operation will halt. On all other instruments, scanning will not halt when this function is issued, but **di_anout** will only execute successfully while input scanning (the **di_anout** function will be ignored when output scanning or simultaneously input and output scanning).

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_ANOUT_ERR	Analog output error

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"

int i;
char errstr[255];

main()
{
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   i = 5 * 16;       /* this left justifies the data */
   i = i & 0XFFFC;   /* the lower 2 bits should be zero */
   if(errcode = di_anout(i,0)){        /* output i to DAC */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Analog output successful.....\n");
   di_close ();
}
```

di_buffer_alloc

- **Summary**

`int _huge *di_buffer_alloc(chn,size);`

```
unsigned chn;     /* 0 = input channel or 1 = output channel (output
                     functions not available on DI-401 and DI-700
                     instruments) */
unsigned size;    /* words to allocate for buffer */
```

- **Description**

The **di_buffer_alloc** function allocates a memory block of at least `size` words (1 word = 1 sample). The memory is assigned to input if `chn` = 0 or to output if `chn` = 1 (output functions are not available on DI-401 and DI-700 instruments). The LSB of `chn` is the only bit checked to determine input or output (output functions are not available on DI-401 and DI-700 instruments). Minimum buffer sizes (in words) are as follows:

Instrument:	If allocating only one buffer:	If allocating an input and an output buffer: Input buffer	Output buffer
DI-400 Series	15,000	8,192	6,144
DI-500 Series, DI-700, DI-720, DI-730, and DI-5001	7,500	4,096	3,072
All others	8,182	4,091	4,091

• Return Value

The **di_buffer_alloc** function returns a huge integer pointer if it can allocate memory of `size` and if a buffer has not been previously allocated for `chn`. **NULL** is returned if allocation fails.

• Dependencies

di_open
di_inlist or di_outlist (output functions not available on DI-401 and DI-700 instruments)

• Example

```
#include "200sdk.h"

int *input_buffer;
int *output_buffer;
```
(output functions not available on DI-401 and DI-700 instruments)

```
main()
{
   di_open ();
   if((input_buffer = di_buffer_alloc(0,4096)) == NULL)     /* allocate
                                                            input buffer */
      printf("Insufficient memory or input buffer already allocated...\n");
```
NOTE: output functions not available on DI-401 and DI-700 instruments
```
   if((output_buffer = di_buffer_alloc(1,4096)) == NULL)    /* allocate
                                                            output buffer*/
      printf("Insufficient memory or output buffer already allocated...\n");
   di_close ();

}
```

di_buffer_free

- **Summary**

```
int di_buffer_free(chn);
```

```
unsigned chn;        /* 0 = input channel or 1 = output channel (output
                        functions not available on DI-401 and DI-700
                        instruments) */
```

- **Description**

The **di_buffer_free** function deallocates a memory block. The memory block was previously allocated by **di_buffer_alloc**. The LSB of `chn` is the only bit checked to determine input or output (output functions are not available on DI-401 and DI-700 instruments, therefore no bit is checked on these instruments).

- **Return Value**

DI_NO_ERR

- **Dependencies**

di_open
di_inlist or di_outlist (output functions not available on DI-401 and DI-700 instruments)
di_buffer_alloc
di_start_scan
di_buffer_status
di_stop_scan

- **Example**

```
#include "200sdk.h"

int   *input_buffer;
int   *output_buffer;  (output functions not available on DI-401 and DI-700
                        instruments)

main()
{
   di_open ();
   if((input_buffer = di_buffer_alloc(0,4096)) == NULL)  /* allocate input
                                                            buffer */
      printf("Insufficient memory or input buffer already allocated...\n");
```

NOTE: output functions not available on DI-401 and DI-700 instruments)

```
   if((output_buffer = di_buffer_alloc(1,4096)) == NULL) /* allocate output
                                                            buffer*/
      printf("Insufficient memory or output buffer already allocated...\n");
   di_buffer_free(0);      /* free input buffer */
   di_buffer_free(1);      /* free output buffer */
   di_close ();

}
```

---

di_buffer_size

- **Summary**

`unsigned di_buffer_size(void);`

- **Description**

The **di_buffer_size** function returns the size of the input buffer. This function is used to get the size of the input buffer, when the buffer is allocated by WINDAQ software. The size of the buffer is needed to determine the proper index for use with the **di_copy_buffer** function.

- **Return Value**

The **di_buffer_size** function returns an unsigned integer equal to the size of the input buffer.

- **Dependencies**

none

- **Example (Basic code)**

```
buffer_size = di_buffer_size()               'get size of buffer
buffer_pointer = di_buffer_status(0) – 100   'get the last 100 data points
If buffer_pointer < 0 Then                   'check for wrap around
     buffer_pointer = buffer_size + buffer_pointer       'adjust pointer if
                                                          wrap around
End If
i% = di_copy_buffer(buffer_pointer, inbuffer(0), 100)   'copy buffer to
                                                          user array
```

di_buffer_status

- **Summary**

```
unsigned di_buffer_status(chn);

unsigned chn;        /* 0 = input channel or 1 = output channel (output
                        functions not available on DI-401 and DI-700
                        instruments) */
```

- **Description**

The **di_buffer_status** function gets the position of the next entry into the buffer. The position is for an input if `chn` = 0 or an output if `chn` = 1 (output functions are not available on DI-401 and DI-700 instruments). The LSB of `chn` is the only bit checked to determine input or output (output functions are not available on DI-401 and DI-700 instruments, therefore no bit is checked on these instruments). The size of the input or output buffer should be evenly divisible by the length of the scan list in order to use the pointer as a buffer index (or starting position of the next scan).

- **Return Value**

The **di_buffer_status** function returns an unsigned integer.

- **Dependencies**

di_open
di_buffer_alloc
di_list_length
di_start_scan
di_inlist or di_outlist (output functions not available on DI-401 and DI-700 instruments)

- **Example**

```
#include "200sdk.h"

int   *input_buffer,errcode;
struct di_mode_struct mode = {0};
struct di_inlist_struct inlist[256] = {0};
char errstr[255];
```

```
main()
{
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   /* allocate 4096 words for input */
   if((input_buffer = di_buffer_alloc(0,4096) = = NULL)
      printf("Insufficient memory or input buffer already allocated...\n");
   if(errcode = di_list_length(1,0))   /* Set the input list length */
      printf("Input list length error...\n");
   if(errcode = di_inlist(inlist))     /* Set up the input list */
      printf("Input list error...\n");
   if(errcode = di_scan_mode(mode))    /* Set mode */
      printf("Mode error...\n");
   if(errcode = di_start_scan())       /* Start scanning */
      printf("Start scan error...\n");
   while(!kbhit())                     /* Main loop executes until key hit.
                                          Prints last values in buffer. */
      printf("Current value in buffer = %04X\r",*(input_buffer +
              di_buffer_status(0)));
   if(errcode = di_close()){           /* close the device */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
}
```

---

di_burst_rate

- **Summary**

`int di_burst_rate(count);`

```
unsigned count;
```

- **Description**

The **di_burst_rate** function sets the burst rate. All of Dataq Instruments hardware products continuously sample data using a burst sampling technique. With the burst sampling technique, the board or hardware device samples data at one rate (referred to as the maximum sampling rate or *burst rate*) while your computer reports (i.e., displays and stores) this data at another rate (called the sample rate or *throughput rate*). Again, the burst rate determines how fast the board or hardware device samples the data and the sample rate determines how fast the sampled data is reported.

DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222

In most instances, the default value of 80,000Hz is sufficient. However the burst rate can be modified to accommodate a specific sampling rate. The equation that governs burst rate is as follows:

Rev. A or Rev B units:*

$$\text{Burst rate} = \frac{16{,}000{,}000}{\text{Count}}$$

Rev. C and higher units:*

$$\text{Burst rate} = \frac{24{,}000{,}000}{\text{Count}}$$

*Revision levels are returned by the `pal1_rev` structure element in the **di_info** function.

Where: *count* is an even integer that allows the burst rate to be modified. The integer range that determines burst rate is: 200 ≤ count ≤ 32,767 (for Rev. A or B) or 300 ≤ count ≤ 43,690 (for Rev. C and higher). Note that *count* must be an even number. If an odd value is entered for *count*, it will automatically be converted to an even value, rounded down.

DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730 and DI-5001

In most instances, the default value of 40,000Hz is sufficient. However the burst rate can be modified to accommodate a specific sampling rate. The equation that governs burst rate is as follows:

$$\text{DI}-400\text{,DI-}401\text{,DI-}410\text{,DI}-500\text{,andDI-}510\text{Burstrate} = \frac{16{,}000{,}000}{Count}$$

Where: *count* is an integer that allows the burst rate to be modified. The integer range that determines burst rate is: 32 ≤ count ≤ 32,767 for DI-400 Series instruments or 64 ≤ count ≤ 32,767 for DI-500 Series, DI-720, DI-730, and DI-5001 instruments.

DI-700

The equation that governs sample rate is as follows:

$$\text{DI-}700\,\text{Samplerate} = \frac{976.5625}{\text{Count}}$$

Where: *count* is an integer that allows the sample rate to be modified. The integer range that determines sample rate is: 1 ≤ count ≤ 32,767.

Additional consideration must be given to burst rate when performing multiple tasks. The following charts show the fastest sampling speeds that can be expected (and the count required to deliver that speed) when performing the indicated tasks simultaneously.

DI-200, DI-201, and DI-210 I/O Throughput Rates*

Minimum Burst Count**	Maximum Continuous Rates	Simultaneous Data Acquisition Tasks: ADC Input	DAC Output	Signal Averaging	Pre & Post Triggering
300	80kHz	✔			
300	80kHz		✔		
300	80kHz	✔		✔	
300	80kHz	✔			✔
337	71.1kHz	✔	✔		
427	56.1kHz	✔	✔	✔	
427	56.1kHz	✔		✔	✔
450	53.3kHz	✔	✔		✔
480	50.0kHz	✔	✔	✔	✔

*based on a 33MHz '386. **for Rev. C and higher units.

DI-220, DI-221TC, and DI-222 I/O Throughput Rates*

Maximum Continuous Rates†		Simultaneous Data Acquisition Tasks			
Standard Parallel Port	Bi-directional Parallel Port	ADC Input	DAC Output	Signal Averaging	Pre & Post Triggering
30kHz	37.5kHz	✔			
30kHz	37.5kHz		✔		
30kHz	37.5kHz	✔		✔	
80kHz‡	80kHz‡	✔			✔
30kHz	37.5kHz	✔	✔		
30kHz	37.5kHz	✔	✔	✔	
80kHz‡	80kHz‡	✔		✔	✔
60kHz‡	60kHz‡	✔	✔		✔
50.0kHz‡	50.0kHz‡	✔	✔	✔	✔

*based on a 33MHz '486.
†All non-triggered modes indicate continuous throughput to disk.
‡All triggered modes indicate throughput to on-board 8kb FIFO memory only.

DI-400, DI-401, and DI-410 Throughput Rates*

Minimum Burst Count	Maximum Continuous Rates†	Simultaneous Data Acquisition Tasks: ADC Input[1]	ADC Input[2]	DAC Output	Trig≤ FIFO‡	Trig> FIFO‡
32	500kHz	✔				
64	250kHz		✔			
64	250kHz		✔			✔
64	250kHz		✔		✔	
64	250kHz			✔**		
128	125kHz		✔	✔**		

*based on a 75MHz Pentium.
**The DI-401 is not capable of DAC Output.
†All non-triggered modes indicate continuous throughput to disk. All triggered modes indicate throughput to on-board 15kb FIFO memory only.
‡Trig = (pre-trigger samples + post-trigger samples) × number of channels. FIFO = (7500 / number of channels) × number of channels.
[1]Without signal averaging or min/max calculations.
[2]With signal averaging and min/max calculations.

DI-500, DI-510, DI-720, DI-730, and DI-5001 I/O Throughput Rates*

Maximum Continuous Rates†			Simultaneous Data Acquisition Tasks			
Standard Parallel Port	Bi-directional Parallel Port	Enhanced Parallel Port	ADC Input	DAC Output	Trig≤ FIFO‡	Trig> FIFO‡
40kHz	80kHz	250kHz	✔			
40kHz	80kHz	250kHz	✔			✔
250kHz	250kHz	250kHz	✔		✔	
40kHz	80kHz	250kHz		✔		
20kHz	40kHz	125kHz	✔	✔		

*based on a 75MHz Pentium.
†All non-triggered modes indicate continuous throughput to disk. All triggered modes indicate throughput to on-board 8kb FIFO memory only.
‡Trig = (pre-trigger samples + post-trigger samples) × number of channels.
FIFO = (7500 / number of channels) × number of channels.

For example, with a DI-220, the fastest sampling rate you could program while simultaneously inputting data, outputting data, averaging data, and triggering is 50.0kHz.

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_BURST_ERR	Burst rate error

• Dependencies

di_open

• Example

```
#include "200sdk.h"

int errcode;
unsigned rate;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){                /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   count = 193;                            /* init rate */
   if(errcode = di_burst_rate(count)){     /* stop scanning */
      di_strerr(errcode,errstr);
```

```
         printf("%s",errstr);
      }
      else
         printf("Burst rate set.....\n");
      di_close();
   }
```

---

di_close

- **Summary**

  `int di_close(void);`

- **Description**

  The **di_close** function stops scanning, restores the device to its initial state, frees all opened buffers, and closes the device for communications.

- **Return Values**

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error

- **Dependencies**

  di_open

- **Example**

```
#include "200sdk.h"

int   errcode;
char errstr[255];

main()
{
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   if(errcode = di_close()){           /* close the device */
      di_strerr(errcode, errstr);
      printf("%s",errstr);
   }
   else
      printf("Closed successfully.....\n");
}
```

---

di_copy_array

(not available on DI-401 and DI-700 instruments)

• Summary

`int di_copy_array (buf_index, source, number);`

```
unsigned buf_index;  /* specifies the offset from the first element in the
                        buffer */
int _huge *source;   /* specifies source BASIC array name */
unsigned number;     /* specifies how many array elements to copy to the
                        output buffer */
```

• Description

The **di_copy_array** function is used to copy the contents of a BASIC array into the output buffer. This function is necessary since Quick BASIC and Visual BASIC do not support pointers. If an offset of zero is specified (by `buf_index`), a number of elements (specified by `number`) from a BASIC array (specified by `source`) are copied to the output buffer starting with the very first element in the array. Similarly, specifying an offset of 10 (by `buff_index`) will start the copy process with the tenth value of the output buffer.

The _huge pointer allows C programmers to specify a source array larger than 64k bytes.

This function also supports buffer wrapping (i.e., if `buf_index` and `number` exceed the buffer size in words, the copy process continues to the start of the buffer). The **di_buffer_status** function can be used to compute `buf_index`, which must be between 0 and (buffer size - 1). Although `number` may be as large as the buffer size, active scanning may concurrently output values starting at the index returned by **di_buffer_status**.

• Return Value

DI_NO_ERR

- **Dependencies**

di_buffer_alloc

- **Example (BASIC code)**

```
Declare Sub di_copy_array Lib "200SDK.DLL" (ByVal buf_index%, source%(),
ByVal number%)

Dim out_buff%                ' offset in the output buffer
Dim analog_out%(1000)        ' array used to store analog out
Dim number%                  ' number of samples to copy

Sub Form_Load ()
   ' This copies the first 500 elements of analog_out
   ' to the output buffer at offset 0
   buf_index = 0
   number = 500
   di_copy_array buf_index,analog_out(0),number
End Sub
```

---

di_copy_buffer

- **Summary**

`int di_copy_buffer (buf_index, dest (0), number);`

```
unsigned buf_index;  /* specifies the offset from the first element in the
                        buffer */
int _huge *dest (0); /* specifies the destination array name */
unsigned number;     /* specifies how many buffer elements to copy to dest
                        (0) */
```

- **Description**

The **di_copy_buffer** function is used to copy the input buffer into a BASIC array, as defined by `dest`. This function is necessary since Quick BASIC and Visual BASIC do not support pointers. If an offset of zero is specified (by `buf_index`), a number of buffer elements (specified by `number`) are copied to an array (specified by `dest`) starting with the very first element in the buffer. Similarly, specifying an offset of 10 (by `buff_index`) will start the copy process with the tenth element from the top of the buffer.

The _huge pointer allows C programmers to specify a destination array larger than 64k bytes.

This function also supports buffer wrapping (i.e., if `buf_index` and `number` exceed the buffer size in words, the copy process continues from the start of the buffer). The **di_buffer_status** function can be used to compute `buf_index`, which must be between 0 and (buffer size - 1). Although `number` may be as large as the buffer size, active scanning may concurrently modify values starting at the index returned by **di_buffer_status**.

• Return Value

DI_NO_ERR

• Dependencies

di_buffer_alloc

• Example (BASIC code)

```
Declare Sub di_copy_buf Lib "200SDK.DLL" (ByVal buf_index%, dest%(), ByVal
number%)

Dim in_buff%         ' offset in the input buffer Dim
analog_in%(1000)     ' array used to store analog in
Dim number%          ' number of samples to copy

Sub Form_Load ()
   ' This copies the first 500 values of the input buffer
   ' to the array 'analog_in' starting with element 0
   buf_index = 0
   number = 500
   di_copy_buffer buf_index,analog_in(0),number
End Sub
```

di_copy_header

• Summary

```
unsigned FARC PASCAL di_copy_header(unsigned hdr_index, void
   FAR *dest, unsigned byte_count, unsigned clear_bits);
```

```
unsigned hdr_index         /* offset from beginning of header */
```

```
void FAR *dest              /* destination structure name */
unsigned byte_count         /* number of bytes from header copied to dest */
unsigned clear_bits         /* mask for clearing flag bits */
```

• Description

The **di_copy_header** function is used to copy CODAS header information into a BASIC structure, as defined by `dest`. This function is necessary since Visual BASIC does not support pointers. If an offset of zero is specified (by `hdr_index`), a number of bytes (specified by `byte_count`) are copied to a structure (specified by `dest`) starting with the very first byte in the header. Similarly, specifying an offset of 10 (by `hdr_index`) will start the copy process with the tenth byte from the beginning of the header.

• Return Value

The **di_copy_header** function returns flags that indicate whether or not WINDAQ software has updated the header. When the header is updated, all flag bits are set to one (returned value is FFFF).

• Dependencies

A running WINDAQ application.

• Example (BASIC code)

```
Declare Function di_copy_header Lib "200SDK.DLL" (ByVal
hdr_index%,dest%,ByVal byte_count%,ByVal clear_bits%)

Dim flags%                                  'flag bits
Dim hdr_index%                              'offset in header
Dim CODAS_header as CODAS_header_struct     'structure to store header
                                            information
Dim byte_count%                             'number of bytes to copy
Dim clear_bits%                             'mask to indicate which flag bits
                                            to clear

Sub Timer ()

'This program checks if the header was updated and copies the first 10
bytes 'of the header to the structure CODAS_header if it was, starting with
the 'first byte, then clears the lsb of the flag bits.

hdr_index = 0
byte_count = 10
clear_bits = 1

flags = di_copy_header (hdr_index,CODAS_header,0,clear_bits)
if (flags and clear_bits) <> 0 then 'using only the lsb of flag bits
   flags = di_copy_header (hdr_index,CODAS_header,byte_count,clear_bits)
```

```
end if

End Sub
```

di_copy_mux

(available only on DI-400, DI-410, DI-500 Series, DI-720, DI-730, and DI-5001 instruments)

• Summary

```
int di_copy_mux(dest);

int *dest;           /* pointer to 16-word buffer */
```

• Description

Each byte describes a bank of 16 channels. The first two bytes are reserved, the third byte is for channels A1 through A16, the fourth is for A17 through A32, etc. In each byte, the bit assignments are as follows:

bit 7	bit 6	bits 5, 4, 3	bit 2	bit 1	bit 0
PGH/PGL	Input Type	Channel	MUX	Cable	High Voltage Option

Bit 7: PGH/PGL Status
This bit describes the MUX gain status. When set, the MUX is low gain PGH (programmable gain factors 1, 2, 4, and 8). When clear, the MUX is high gain PGL (programmable gain factors 1, 10, 100, and 1000 on DI-500 Series instruments or 1, 10, and 100 on DI-400 Series instruments).

Bit 6: Reserved
This bit describes the input type. When set, the inputs are signal conditioned (DI-5B amplified). When clear, the inputs are high-level.

Bits 5, 4, and 3: Channel status
These bits contain bits one through three of the primary channel to which the MUX connects (bit 4 of the primary channel is always 1, and bit 0 is 1 for odd channels or 0 for even channels).

Bit 2: MUX status
This bit describes the MUX status. When set, MUX is not present. When clear, MUX is present.

Bit 1: Cable status
This bit describes the cable status. When set, the cable is not defective. When clear, the cable is defective.

Bit 0: High voltage option status
This bit describes the high voltage option status of the bank of 16 channels. When clear, this bank of 16 channels has high voltage measurement capability. When set, this bank of 16 channels does not have high voltage measurement capability.

- **Return Value**

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_INFO_ERR	Device is not a DI-400 Series, DI-500 Series, DI-720, DI-730, or DI-5001 instrument

- **Dependencies**

none.

- **Example**

```
main()
{
   int mux_infor[16];
   di_open();
   di_copy_mux(mux_info);
   di_close();
}
```

---

di_ct_event

(not available on DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, or DI-5001 instruments)

- **Summary**

`int di_ct_event(unsigned bit_and,unsigned bit_xor);`

```
unsigned bit_and;          /* specifies which input bit(s) to count */
unsigned bit_xor;          /* compare value */
```

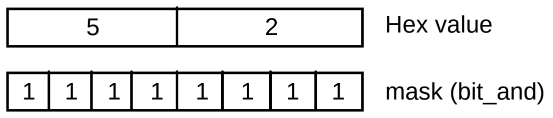
• Description

The **di_ct_event** function starts event counting using the digital input bits. Before event counting can begin, you must specify the bit mask (`bit_and`) and the way an event will be counted (`bit_xor`). Use the **di_ct_status** function to return the event count (maximum count is 64K).

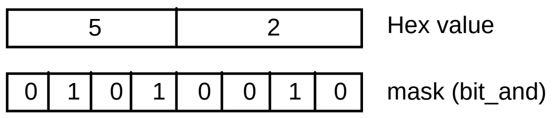
`bit_and` is used to specify which digital input bit(s) you wish to use for event counting. `bit_and` creates a bit mask that ANDs the actual digital input port with the corresponding bit in the mask. A 1 in the bit mask unmasks the corresponding input port, allowing the signal from this port to be compared with `bit_xor`. Similarly, 0's in the bit mask prevent the input port(s) from being compared.

`bit_xor` is used as a compare value to define the trigger condition. The signal from the input bit(s) that passes through the bit mask is XOR'd with the corresponding bit in `bit_xor`. If the compared values are the same, the event is counted. If the compared values are not the same, the event is not counted.

For example, suppose you had an eight-bit counter connected to the input ports and you wanted to count every occurrence of 52. Start by specifying `bit_and`. In this case, all the bits of the bit mask should be unmasked (set to 1 or allowed to pass through) as follows:



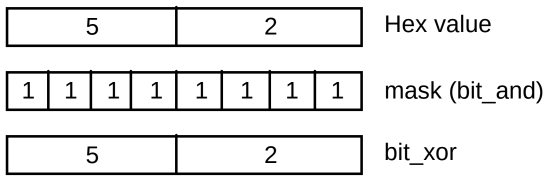
If only the input ports that correspond to 52 were unmasked, several additional values would be allowed to pass, resulting in false event counts as follows:



5	3
5	6
5	7
5	A
5	B
5	E
5	F
7	3
7	6
7	7
7	A
7	B
7	E
7	F
D	3
D	6
D	7
D	A
D	B
D	E
D	F
F	3
F	6
F	7
F	A
F	B
F	E
F	F

Hex values that would also pass through the above mask

As the above illustrates, selecting an appropriate `bit_and` value is important. In our example, the hexadecimal value for `bit_and` would be FF. The next step is to determine `bit_xor`. Since you are interested in counting every occurrence of 52, `bit_xor` is set to 52.



The di_ct_event function call for this example would be:

```
di_ct_event(0xFF,0x52)
```

The minimum pulse width for this function is 500 nS. The maximum frequency is 500 kHz

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error

DI_CT_EVENT_ERR	Counter timer start error

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"

int errcode;
unsigned bit_in;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   bit_in = 0;                         /* use digital input bit 0 */
   if(errcode = di_ct_event(2,2)){     /* start event counting */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Event counting started.....\n");
   di_close();
}
```

di_ct_one_shot

(not available on DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, or DI-5001 instruments)

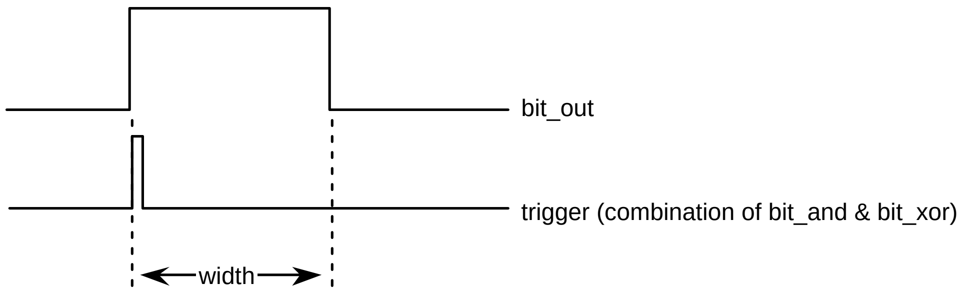
- **Summary**

`int di_ct_one_shot(bit_and, bit_xor, bit_out, width);`

```
unsigned bit_and; /* specifies input bit(s) to be used as trigger */
unsigned bit_xor; /* trigger compare value */
unsigned bit_out; /* digital output bit(s) to be used as one shot output */
unsigned width;   /* pulse width = (125 nS + 1000*width) */
```

• Description

The **di_ct_one_shot** function performs a digital one-shot operation according to the parameters passed to it. A digital one-shot occurs when one or more digital output lines (`bit_out`) change state for a specified amount of time (`width`) after a specified trigger occurs (`bit_and`) and (`bit_xor`). After `width` expires, `bit_out` reverts back to its original state. The following diagram illustrates a typical one-shot operation:



`bit_and` is used to specify which digital input bit(s) you will be using for the trigger. `bit_and` creates a bit mask that ANDs the actual digital input port (used as the trigger) with the corresponding bit in the mask. A 1 in the bit mask unmasks the corresponding input port(s), allowing the signal from this port to be compared with `bit_xor`. A 0 in the bit mask prevents the other input port(s) from being compared.

`bit_xor` is used as a compare value to define the trigger condition. The signal from the input bit(s) that passes through the bit mask is XOR'd with the corresponding bit in `bit_xor`. If the compared values are the same, a one-shot is generated. If the compared values are not the same, the one-shot is not generated.

The maximum pulse width is 65.536 mS.

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_ONE_SHOT_ERR	Counter timer one shot error

• Dependencies

di_open

- **Example**

```
#include "200sdk.h"

int errcode;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");

   if(errcode = di_ct_one_shot(1,1,1,128)){      /* one shot function */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }

   di_close();

}
```

---

di_ct_status

(not available on DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, or DI-5001 instruments)

- **Summary**

```
unsigned     di_ct_status(void);
```

- **Description**

The **di_ct_status** function returns the present count of the event counter.

- **Return Value**

Present event counter value.

- **Dependencies**

di_open

di_ct_event

- **Example**

```
#include "200sdk.h"

int errcode;
char errstr[255];
struct di_mode_struct mode;

main()
{
int i;
   if(errcode = di_open()){         /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   di_ct_event(1,1); /* start event counting */
   while(!kbhit())   /* count events while waiting for key to be pressed */
      ;
   i = di_ct_status();
   printf("Count = %d\n",i);
   di_close():
}
```

---

di_ct_stop

(not available on DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, or DI-5001 instruments)

- **Summary**

`int di_ct_stop(void);`

- **Description**

The **di_ct_stop** function stops previously started event counting.

- **Return Value**

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_CT_STOP_ERR	Counter timer stop error

- **Dependencies**

di_open
di_ct_event

- **Example**

```
#include "200sdk.h"

int errcode;
unsigned bit_in;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   bit_in = 0;                         /* use digital input bit 0 */
   if(errcode = di_ct_event(bit_in)){  /* start event counting */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Event counting started.....\n");
   if(errcode = di_ct_stop()){         /* stop event counting */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Event counting stopped.....\n");
   di_close();
}
```

---

di_ct_wave

(not available on DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, or DI-5001 instruments)

- **Summary**

`int di_ct_wave(bit_out, hi_factor, lo_factor);`

```
unsigned bit_out;  /* digital output bit(s) to be used as wave output */
unsigned hi_factor;  /* waveform high width = (1µs * hi_factor) */
unsigned lo_factor;  /* waveform low width = (1µs * lo_factor) */
```

• Description

The **di_ct_wave** function generates a square wave that goes high for `(1µs * hi_factor)` and goes low for `(1µs * lo_factor)`. The values entered for `hi_factor` and `lo_factor` in the above equations determine the high and low pulse widths.

For example, suppose you wanted to generate a square wave output on digital output bits 6 and 0 that goes high for 1µS and then low for 4µS. Start by determining the hi and lo factors. Plug 1µS into the equation and solve for the `hi_factor`:

$$\begin{aligned} 1\mu S &= (1\mu S \times \text{hi\_factor}) \\ 1 &= \text{hi\_factor} \end{aligned}$$

Similarly, plug 4µS into the equation and solve for the `lo_factor`:

$$\begin{aligned} 4\mu S &= (1\mu S \times \text{lo\_factor}) \\ 4 &= \text{lo\_factor} \end{aligned}$$

Next, determine the hex value for `bit_out`. The hex value is dictated by the digital output bits you want to output the square wave. Place a one in the bit position of the desired digital output as follows:

Digital output bits (8 total):	7	6	5	4	3	2	1	0
Desired waveform output (to generate a wave output on bits 6 and 0):	0	1	0	0	0	0	0	1
Hex value:	4				1			

To generate a wave output on digital output bits 6 and 0 that goes high for 1µS and then low for 4µS, the **di_ct_wave** function call would look like this:

di_ct_wave (0x41,1,4)

The hi factor and lo factor equations are valid when the hi and lo factors are non-zero. Using zero for either factor is equivalent to setting it to 2^{16}. The hi and lo factors also must be integers.

The highest waveform frequency that can be generated with this function is 500 kHz (using hi and lo factors of 1) and the lowest frequency that can be generated is 7.629 Hz (using hi and lo factors of 2^{16}).

- **Return Value**

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_CT_WAVE_ERR	Counter timer waveform error

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"

int errcode;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){                  /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   if(errcode = di_ct_wave(1,1000,128)){     /* waveform function */
      di_strerr(errcode, errstr);
      printf("%s",errstr);
   }
   di_close();
}
```

---

di_digin

- **Summary**

`unsigned di_digin(void);`

- **Description**

di_digin is an immediate function that reads a word from the digital input port.

On all instruments except the DI-500 Series, the DI-720, and the DI-5001, the digital data is in the lower byte.

On DI-500 Series, DI-720, and DI-5001 instruments, the digital data is in the higher byte. Only DI-500-16 and DI-510-48 instruments in the DI-500 Series provide access to digital data (it is accessed from the AUXILIARY PORT).

The value read from the digital input port can range from 0 to 255. On DI-200 and DI-210 instruments, this function should not be issued while an input, output, or simultaneous input and output scanning operation is in progress. If it is, the scanning operation will halt. On all other instruments, scanning will continue unimpeded when this function is issued, and **di_digin** will execute successfully.

• Return Value

The function returns a byte from the digital input port.

• Dependencies

di_open

• Example

```
#include "200sdk.h"

int i;
char errstr[255];

main()
{
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");

   i = di_digin();                     /* read digital input port */

   printf("Digital input port = %04X\n",i);
   di_close();
}
```

di_digout

- **Summary**

`unsigned di_digout(i);`

```
unsigned i;    /* value to output to the digital output port */
```

- **Description**

di_digout is an immediate function that writes (outputs) a byte to the digital output port. The output value can range from 0 to 255. On DI-200 and DI-210 instruments, this function should not be issued while an input, output, or simultaneous input and output scanning operation is in progress. If it is, the scanning operation will halt. On all other instruments, scanning will not halt when this function is issued, but the **di_digout** function will be ignored unless a special DSP code has been installed. Contact Dataq Instruments for complete details.

- **Return Value**

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_DIGOUT_ERR	Digital output error

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"
char errstr[255];

int i;

main()
{
   if(errcode = di_open()){             /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
      i = 5;
      if(errcode = di_digout(i)){   /* output i to digital output port */
         di_strerr(errcode,errstr);
         printf("%s",errstr);
      }
      else
         printf("Digital output successful.....\n");
      di_close();
   }
```

---

di_get_acq_header

- **Summary**

`CODASHDR far * di_get_acq_header(void);`

- **Description**

The **di_get_acq_header** function returns either a far pointer to the CODAS header structure (allocated by WINDAQ waveform recording software) or NULL if the structure is not available. This function is used in conjunction with WINDAQ waveform recording software to access header file information (e.g., sample rate, number of channels, etc. Complete header file details can be found in the Data Storage Format section of the *WINDAQ/Pro and WINDAQ/Pro+ User's Manual*). For example, suppose you wanted to write an application that would intercept the data from WINDAQ's data buffer and apply a moving average to it. You would call **di_get_acq_header** to find out the sampling rate, number of enabled channels, etc.

If the programming language you are using accepts Windows™ messages (such as C or C++), you can register the messages "WindaqUpdate" and "WindaqExit", which will notify you whenever WINDAQ waveform recording settings (sample rate, channels, etc.) change. The example segment of code shows how this can be implemented.

- **Return Value**

The **di_get_acq_header** function returns a far pointer to the CODAS header structure (allocated by WINDAQ waveform recording software), or NULL if the structure is not available.

- **Dependencies**

A running WINDAQ application.

- **Example**

```
#include "200sdk.h"

.
.
.
UINT wm_WindaqUpdate;
UINT wm_WindaqExit;

int PASCAL WinMain(HANDLE hInstance, HANDLE hPrevInstance, LPSTR lpszCmdLine,
int nCmdShow)
{
CODASHDR far * lpCODASHDR;
   .
   .
   .
   wm_WindaqUpdate=RegisterWindowMessage("WindaqUpdate");
   wm_WindaqExit=RegisterWindowMessage("WindaqExit");
   .
   .
   .
   lpCODASHDR=di_get_acq_header();
   /* Now you can set up your app according to WINDAQ */
}

LONG FAR PASCAL WndProc(HWND hWnd, WORD Message, WORD wParam, LONG lParam)
{
CODASHDR far * lpCODASHDR;
   .
   .
   .
   if (Message==wm_WindaqUpdate){
      /* WINDAQ/x00 changes its setting */
      lpCODASHDR=di_get_acq_header();
      /* Now you can change settings of your app according to WINDAQ */
      .
      .
      .
   };
   if (Message==wm_WindaqExit){
      /* WINDAQ quits */

   };

}
```

di_info

- **Summary**

int di_info(info);

```
struct di_info_struct{
   unsigned port;            /* device port address */
   unsigned buf_in_chn;      /* device input channel */
   unsigned buf_out_chn;     /* device output channel (all instruments
                                except DI-401 and DI-700) or reserved for
                                compatibility (DI-401 and DI-700
                                instruments) */
   unsigned sft_lvl;         /* software interrupt level */
   unsigned hrd_lvl;         /* hardware interrupt level */
   int huge *buf_in_ptr;     /* input buffer pointer */
   unsigned buf_in_size;     /* input buffer size (in words) */
   int huge *buf_out_ptr;    /* output buffer pointer (all instruments
                                except DI-401 and DI-700) or reserved for
                                compatibility (DI-401 and DI-700
                                instruments) */
   unsigned buf_out_size;    /* output buffer size,in words (all
                                instruments except DI-401 and DI-700) or
                                reserved for compatibility (DI-401 and DI-
                                700 instruments) */
   char tsr_version[20];     /* TSR version */
   char dsp_version[20];     /* DSP program version */
   char sdk_version[20];     /* SDK library version */
   unsigned long serial_no;  /* PCB serial no. */
   unsigned long last_cal;   /* last calibration time in sec since 1/1/1970 */
   char board_id[10];        /* PCB model name */
   char pgh_pgl;             /* type of PGA; 0 = PGH, 1 = PGL (DI-200, DI-
                                220, DI-222, DI-400, DI-500, and DI-510
                                instruments) or reserved for compatibility
                                (all other instruments) */
   char hrdwr_rev;           /* ASCII char REV letter */
   char pal0_rev;            /* ASCII char REV of PAL0 */
   char pal1_rev;            /* ASCII char REV of PAL1 */
}*info;
```

• Description

The **di_info** function loads a structure with information about the device. Elements `buf_in_chn` and `buf_out_chn` equal the device's hardware DMA channel or software FIFO number, depending on the instrument.

The eighth character of the `char dsp_version [20]` string is a 14-bit or 16-bit resolution flag. For instruments capable of recording with either 14 or 16 bits of resolution (i.e., DI-700, DI-720, DI-730, and DI-5001), the eighth character of this string shows "A" for 14-bit and "B" for 16-bit. For all other instruments (i.e., the other instruments that are not capable of recording with either 14- or 16-bits of measurement resolution), "A" means the instrument is using the non-mux DSP program and "B" means the instrument is using the mux version of the DSP program.

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_INFO_ERR	Information error

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"

int errcode;
struct di_info_struct info;
char errstr[255];

main()
{
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   if(errcode = di_info(&info)){       /* Get info about the device*/
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else{
      printf("\nPort:               %04X\n",info.port);
      printf("Input chn:          %04X\n",info.buf_in_chn);
      printf("Output chn:         %04X\n",info.buf_out_chn);
      printf("Sft lvl:            %04X\n",info.sft_lvl);
      printf("Hrd lvl:            %04X\n",info.hrd_lvl);
      printf("Input ptr:          %08lX\n",info.buf_in_ptr);
      printf("Output ptr:         %08lX\n",info.buf_out_ptr);
      printf("Input size:         %04X\n",info.buf_in_size);
      printf("Output size:        %04X\n",info.buf_out_size);
      printf("TSR Ver:            %s\n",info.tsr_version);
      printf("DSP Ver:            %s\n",info.dsp_version);
      printf("SDK Ver:            %s\n",info.sdk_version);
      printf("Serial #:           %08lX\n",info.serial_no);
      printf("Last cal:           %s",ctime(&info.last_cal));
      printf('Board ID:           %s\n",info.board_id);
      printf("PGH/PGL(1/0):       %0x1\n",info.pgh_pgl);
      printf("Hard Rev:           %c\n",info.hrdwr_rev);
      printf("PAL0 Rev:           %c\n",info.pal0_rev);
      printf("PAL1 Rev:           %c\n",info.pal1_rev);

   }
   di_close();
}
```

Note

This example will return zeros for the BUF pointer and the size information because the buffers have not been allocated.

---

di_inlist

- **Summary**

```
   int di_inlist(input_list);

/* To get dig in, set chan = 8 and diff = 1 */

   struct di_inlist_struct{
      unsigned chan;            /* input channel, 0 to 255 */
      unsigned diff;            /* input configuration (single-ended or diff) */
      unsigned gain;            /* gain. 0 to 17 */
      unsigned unipolar;        /* unipolar/bipolar; 0=bipolar, 1=unipolar
                                   (reserved for compatibility on DI-700)*/
      unsigned dig_out_enable;  /* 1=enables dig out, 0=disables dig out
                                   (reserved for compatibility on DI-700)*/
      unsigned dig_out;         /* digital data (reserved for compatibility on
                                   DI-700)*/
      unsigned ave;             /* sample averaging. 0 = off, 1 = on (always
                                   disabled on DI-700)*/
      unsigned counter;         /* sample rate counter (reserved for
                                   compatibility on DI-700)*/
      }*input_list;
```

- **Description**

The **di_inlist** function initializes the input list. This function must be called before **di_start_scan** and after **di_list_length**. Each element in the structure is defined as follows:

`chan` allows you to specify the input channel you wish to sample. Note that the first channel listed on the input scan list is used as the trigger channel by the **di_mode** function (for those instruments that support the **di_mode** function). For example if channel 3 is specified by `chan` as the first element in the input scan list, triggering will occur on channel 3. Values for `chan` can range from 0 to 255, according to the following equations:

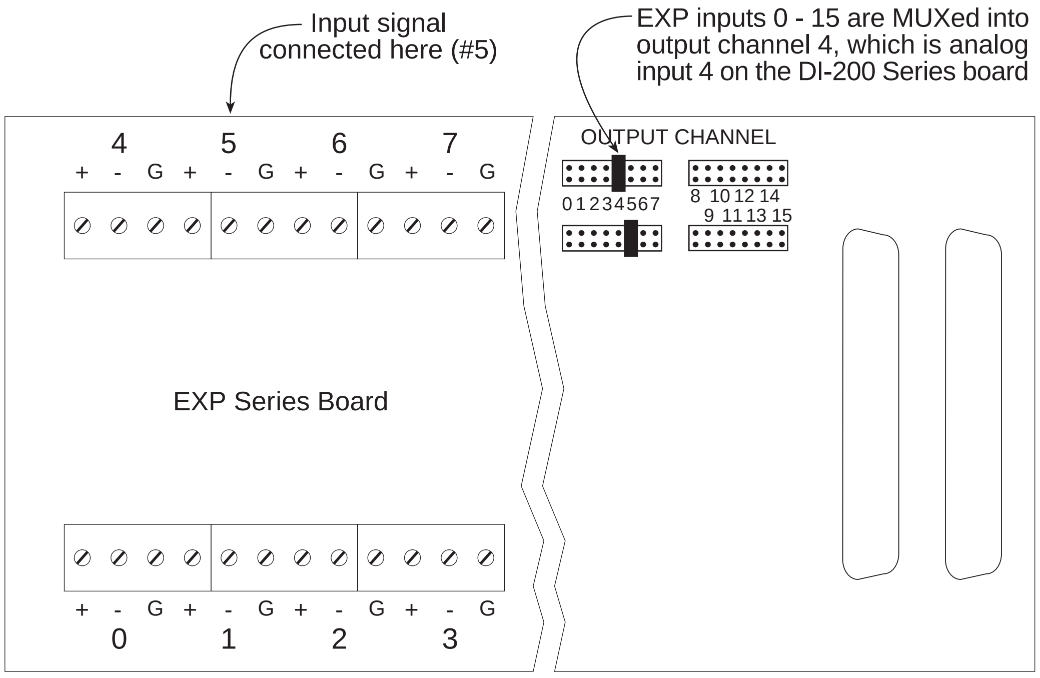
<u>All instruments except DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, and DI-5001</u>

With 16 channels or less: `chan = channel#`

With more than 16 channels:

```
chan = channel# mod 16 + 16(output channel + 1)
```

`channel#` is the analog input channel on the EXP board you wish to sample. `output channel` is the position of the jumper on the EXP board. On hardware with more than 16 channels, the `mod` operator in the equation above combines two integer expressions using modulo arithmetic. For two integer values, modulo arithmetic returns only the *remainder* from an integer division. That is, 6 mod 4 is 2, the remainder of the integer division of 6 by 4. For example, let's say we have three 32-channel EXP boards multiplexed to a DI-200 Series board and we want to record analog input channel five on the third EXP board. What channel do we specify on the input list ?


Input signal connected here (#5)
EXP inputs 0 - 15 are MUXed into output channel 4, which is analog input 4 on the DI-200 Series board
OUTPUT CHANNEL
0 1 2 3 4 5 6 7
8 10 12 14
9 11 13 15
4 5 6 7
+ - G + - G + - G + - G
EXP Series Board
+ - G + - G + - G + - G
0 1 2 3

On each EXP board, each bank of 16 analog inputs is multiplexed into one output (specified by the position of the OUTPUT CHANNEL jumper on the EXP board). This output from the EXP board is connected to an *internal* analog input on the DI-200 Series board. Assuming the first EXP board uses output channels 0 and 1 and the second EXP board uses outputs 2 and 3, the third EXP board will multiplex inputs 0 through 15 into output channel 4 and inputs 16 through 31 into output channel 5. From the equation:

```
chan = (channel#) mod 16 + 16(output channel +1)
chan = (5) mod 16 + 16(4 + 1)
chan = 5 + 80
chan = 85
```

Now suppose we want to record channel 23 on the third EXP board. What channel do we specify on the input list in this case?

```
chan = (channel#) mod 16 + 16(output channel +1)
chan = (23) mod 16 + 16(5 + 1)
chan = 7 + 96
chan = 103
```

Finally, suppose we have just one DI-200 Series board (not multiplexed) and we want to record channel 6. What channel do we specify on the input list? Since we are using non-multiplexed hardware (only 16 channels), the first equation applies:

```
chan = channel#
chan = 6
```

<u>DI-400, DI-401, DI-410, DI-500, DI-510, DI-700, DI-720, DI-730, and DI-5001</u>

With 16 channels or less: `chan = (channel# - 1)`

With more than 16 channels:

```
chan = (channel# - 1) + 32(mux letter - 'A' + 1)
```

`channel#` is the analog input channel you wish to sample. On instruments with more than 16 channels, `mux letter` is the letter you assigned to the instrument (written on the overlay) during installation and initial configuration. This letter is important for keeping track of all the analog input channels on your instrument(s). For example, let's say we have three DI-500-32-P instruments multiplexed together and we want to record analog input channel five on the third instrument. What channel do we specify on the input list? During installation, you would have labeled the instruments "A", "B", and "C". From the equation:

```
chan = (channel# - 1) + 32(mux letter - 'A' + 1)
chan = (5 - 1) + 32(C - A + 1)
chan = 4 + 32(2 + 1)
chan = 4 + 96
chan = 100
```

`diff` allows you to specify whether the channel specified by `chan` is single-ended or differential as follows:

<u>DI-200, DI-201, DI-210, DI-220, DI-222, DI-221TC, DI-400, DI-410, DI-700, DI-720, and DI-5001 Instruments</u>

`diff = 0` for single ended input configuration.
`diff = 1` for differential input configuration (do not set `diff` = 1 for channels on multiplexers, even though the inputs are differential).

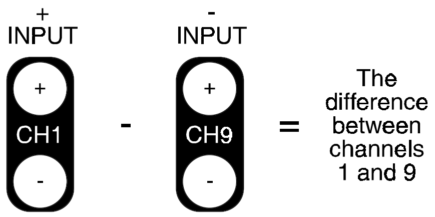
<u>DI-401 Instruments</u>

reserved for compatibility.

<u>DI-500 and DI-510 Instruments</u>

`diff = 0` for single ended input configuration.

`diff = 1` for differential input configuration. This configuration allows you to see the difference between 2 differential input channels, but only on `chan` 1 through 8 and 17 through 24 of DI-500-16 and DI-510-48 instruments. When `chan` 1 through 8 or 17 through 24 is specified for differential operation, the other channel that creates the differential pair is automatically selected, eight channels away. For example, if `chan` 1 is configured for differential operation, `chan` 9 becomes the companion channel, similarly with 2 and 10, 8 and 16, 18 and 26, etc. In every case, the lowest channel number becomes the positive (+) differential input and the automatically selected channel becomes the negative (-) differential input.



<u>DI-730 Instruments</u>

`diff =  0` always. DI-730 instruments are always differential.

`gain` allows you to specify a gain factor (assigned to a code, from the following table) for the channel specified by `chan`. With the DI-221TC, you can specify a gain factor for linear or non-linear inputs. Note that this structure element is reserved for compatibility on DI-401 instruments (gain is fixed at 1):

Code	DI-200PGH, DI-201PGH, DI-210, DI-220PGH, DI-222PGH, DI-400PGH, DI-410, DI-500PGH, DI-510PGH, DI-720, DI-5001	DI-200PGL, DI-201PGL, DI-220PGL, DI-222PGL, DI-500PGL, DI-510PGL, DI-700	DI-400PGL	DI-221TC Only		DI-730
	Gain	Gain	Gain	Gain	Input Type	Gain
0	1	1	1	1		100 (10Vfs)
1	2	10	10	10	Linear	1,000 (1Vfs)
2	4	100	100	100		10,000 (0.1Vfs)
3	8	1,000		1,000		100,000 (0.01Vfs)
4*				1		1 (1,000Vfs)
5*				10	Non-linear 0†	10 (100Vfs)
6‡				100		
7‡				1,000		
8*				1		
9*				10	Non-linear 1†	
10				100		
11				1,000		
12*				1		
13*				10	Non-linear 2†	
14				100		
15				1,000		
16				Reserved	Reserved	
17				CJC	†	

*Gain codes 4, 5, 8, 9, 12, and 13 are undefined when making thermocouple measurements.

Vfs = volts full scale.

†If you are specifying a gain code from non-linear input groups 0, 1, or 2 and you are measuring with a thermocouple, you must add an additional channel to the input scan list. This additional channel must specify cold junction compensation (code 17) as the gain element and it should be placed last in the input scan list.

Thermocouple type is defined by the value of the `unsigned chan` variable of the last scan list element containing gain code 17:

`unsigned chan` value	TC type	Multiplier for gain of 100	Multiplier for gain of 1000
0	K	1232.3333	121.9772
1	J	760	94.94897
2	T	400	115.2563
3	R	1768	548.1296

‡The A/D converter delivers "counts" as an end result instead of degrees. These counts can be scaled to °C as follows:

$$\left(\frac{x}{32768}\right)\times(multiplier) = ^{\circ}C$$

where x = ADC counts as delivered by the hardware, and *multiplier* is from the table above. Make sure you use the appropriate multiplier for the selected gain.

`unipolar` allows you to specify whether the channel specified by `chan` is a unipolar or bipolar signal. On DI-401 and DI-700 instruments, this structure element is reserved for compatibility. On DI-400, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001 instruments, unipolar configuration is not supported. Therefore on these instruments, `unipolar` must be `0`.

`dig_out_enable` allows you to enable or disable the digital output bits. This structure element is reserved for compatibility on DI-700 instruments.

DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222 Instruments

When `dig_out_enable` is 1, digital output is enabled and the data appearing on the digital output bits is written to lines D0 through D4 of the digital output port.

When `dig_out_enable` is 0, digital output is disabled. When digital output is disabled and:

`chan` = 0 to 15, digital output bits D2 through D0 are used to specify an acquisition method (either average, minimum, or maximum. Refer to the description of `dig_out` for full details). In other words, you cannot use the three least significant bits of digital output, they are used to specify acquisition method.

`chan` is on a multiplexer, lines D3 through D0 of the digital output port are set to the least significant four bits of the EXP board input being multiplexed, and `dig_out_enable` automatically becomes enabled (`chan` setting of 16 or greater automatically overrides a `dig_out_enable` = 0 setting). In other words, with a MUX channel, you cannot use the four least significant bits of digital output (they are used) and you cannot specify a maximum or minimum acquisition method.

DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001

When `dig_out_enable` is 1, digital output is enabled and the data appearing on the digital output bits is written to lines D0 through D3 of the EXPANSION port.

When `dig_out_enable` is 0, digital output is disabled. When digital output is disabled and:

`chan` is on a multiplexer, lines D3 through D0 of the EXPANSION port are set to the least significant four bits of the channel to which the multiplexer is connected, and `dig_out_enable` automatically becomes enabled (when `chan` is on a multiplexer, a `dig_out_enable` = 0 setting is automatically overridden). In other words, with a MUX channel you cannot use the four least significant bits of digital output, they are taken.

`dig_out` is a multipurpose structure element. It's function depends on the instrument being used and the status of the `dig_out_enable` element. This structure element is reserved for compatibility on DI-700 instruments.

<u>**DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222 Instruments**</u>

When digital output is enabled (`dig_out_enable` = 1), the `dig_out` bits output a digital value to lines D4 through D0 of the digital output port.

When digital output is disabled (`dig_out_enable` = 0) and `chan` = 0 to 15, `dig_out` bits D2 through D0 are used to specify an acquisition method as follows:

D2	D1	D0	
0	0	0	reports average value
0	0	1	reports maximum value
0	1	0	reports minimum value

All Dataq Instruments hardware products continuously sample data using a burst sampling technique. With the burst sampling technique, the board or hardware device samples data at one rate (referred to as the maximum sampling rate or burst rate) while your computer reports (i.e., displays and stores) this data at another rate (called the sample rate or throughput rate). Again, the burst rate determines how fast the board or hardware device samples your data and the sample rate determines how fast the sampled data is reported. The board or hardware device can sample data much faster than it can report it. For example, let's say we want to record one channel of data at 100 Hz and our burst rate is set at 50kHz. In this example, we will be sampling the data at 50kHz and reporting it at 100Hz, a 500 to 1 ratio. This means that for every 500 data points sampled, only 1 will be reported. The dilemma becomes: which data point out of the 500 gets reported? Fortunately, you have a choice of methods for reporting this single data point.

Average - This method averages all of the data points in the burst sample and returns this average as the single value for storage and display. This is the most universal method and should be used in all cases unless you wish to report peak or valley values (in which case you would use the maximum or minimum method respectively).

Maximum - This method returns the highest value data point in the burst sample for storage and display. The rest of the data points in the burst sample are ignored.

Minimum - This method returns the lowest value data point in the burst sample for storage and display. The rest of the data points in the burst sample are ignored.

Last Point - This method simply returns the last input data point in the burst sample for storage and display. The rest of the data points in the burst sample are ignored. This is the only data reporting method used by DI-700 instruments.

When digital output is disabled (`dig_out_enable` = 0) and `chan` is on a multiplexer, `dig_out` bits D3 through D0 are set to the least significant four bits of the EXP board input being multiplexed. In other words, with a MUX channel you cannot use the four least significant bits of digital output, they are taken.

<u>DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730 and DI-5001</u>
When digital output is enabled (`dig_out_enable` = 1), the `dig_out` bits output a digital value to lines D3 through D0 of the EXPANSION port.

When digital output is disabled (`dig_out_enable` = 0), there is no digital output (digital output is disabled). However, when digital output is disabled and `chan` is on a multiplexer, lines D3 through D0 of the EXPANSION port are set to the least significant four bits of the channel to which the multiplexer is connected, and `dig_out_enable` automatically becomes enabled (when `chan` is on a multiplexer, a `dig_out_enable` = 0 setting is automatically overridden). In other words, with a MUX channel you cannot use digital output, it is disabled.

`ave` allows you to enable or disable sample averaging (always disabled on DI-700 instruments).

When sample averaging is enabled (`ave` = 1), up to 32,767 consecutive samples for each entry on the input list can be averaged. All Dataq Instruments hardware products continuously sample and report data using a burst sampling method. When averaging is enabled, data is temporarily stored in an accumulator until the sample interval (specified by `counter` on the input list) elapses. When the sample interval elapses, the value reported is not a single instantaneous sample but the average of all samples since the last interval. On DI-200 Series instruments, averaging works only for the first 32 analog input channels appearing on the input scan list. It is not possible to average all 256 entries on the list. However on DI-400 Series, DI-500 Series, DI-720, DI-730, and DI-5001 instruments, it is possible to average all 256 input scan list entries.

When sample averaging is disabled (`ave` = 0) on:

<u>DI-200, DI-201, DI-210, DI-220, DI-221TC, and DI-222 Instruments</u>
The last point acquisition method is enabled (refer to the description of `dig_out` for full acquisition method details).

<u>DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001</u>
Digital output bit D4 and `ave` work together to specify the acquisition method as follows (refer to the description of `dig_out` for full acquisition method details):

D4	ave	acquisition method
0	0	reports last point
0	1	reports average value
1	0	reports minimum value
1	1	reports maximum value

`counter` allows you to adjust the sample rate counter. This structure element is reserved for compatibility on DI-700 instruments. Since the input scan list is capable of holding 256 entries, it is possible to program each channel in the input list for a different sampling rate. The equation for determining the value required for a specific sampling rate is as follows:

$$S = \frac{B}{L(C+1)}$$

Where: S = desired sampling rate of the input list entry, B = burst rate of the instrument, L = length of the input **or** output list **(whichever is greater)**, and C = "count weight" or `counter`.

Since simultaneous input and output operations are possible, some consideration must be given to input and output synchronization. The following table illustrates the order of each input/output operation, with respect to the other operations. In this example, there are 10 elements each in the input and output scan lists. Each input or output is referenced to its position in the input or output scan list.

Sample Number	Analog Input	Analog Output	Digital In DMA	Digital Out DMA	Digital Out Inlist	Digital Out Outlist
1	-	-	-	-	0	-
2	-	-	-	0	1	0
3	0	-	0	1	2	1
4	1	0	1	2	3	2
5	2	1	2	3	4	3
6	3	2	3	4	5	4
7	4	3	4	5	6	5
8	5	4	5	6	7	6
9	6	5	6	7	8	7
10	7	6	7	8	9	8
11	8	7	8	9	0	9
12	9	8	9	0	1	0
13	0	9	0	1	2	1
14	1	0	1	2	3	2
15	2	1	2	3	4	3
16	3	2	3	4	5	4
17	4	3	4	5	6	5
18	5	4	5	6	7	6
19	6	5	6	7	8	7
20	7	6	7	8	9	8

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_INLIST_ERR	Value in inlist structure out of range error

• Dependencies

di_open
di_list_length

• Example

```
#include "200sdk.h"

int errcode;
#define IDIM 256  /* largest value allowed */
struct di_inlist_struct inlist[IDIM] = {0};  /* input list cleared */
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
```

```
         di_strerr(errcode,errstr);
         printf("%s",errstr);
      }
      else
         printf("Device installed successfully.....\n");

/* The following initializes scan list position 0 and must be repeated for all
   positions intended to be scanned */
      inlist[0].chan = 5;                 /* channel 5 */
      inlist[0].diff = 0;                 /* single ended */
      inlist[0].gain = 0;                 /* gain of 1 */
      inlist[0].unipolar = 0;             /* bipolar */
      inlist[0] dig_out_enable = 1        /* enable digital output */
      inlist[0].dig_out = 3;              /* output digital value 3 */
      inlist[0].ave = 0;                  /* averaging off */
      inlist[0].counter = 100;            /* init counter */

      di_list_length (IDIM, 0);
      if(errcode = di_inlist(inlist)){ /* initialize the input list */
         di_strerr(errcode,errstr);
         printf("%s",errstr);
      }
      else
         printf("Input list initialized.....\n");
      di_close();
   }
```

---

di_list_length

• Summary

```
int     di_list_length(in_length,out_length);

unsigned in_length;        /* sets input list length */
unsigned out_length;       /* sets output list length */
```

• Description

The **di_list_length** function sets input and output scan list lengths. On DI-401 and DI-700 instruments, output functions are not available. Therefore `unsigned out_length` is reserved for compatibility.

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_LENGTH_ERR	List length error

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"

int errcode;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){                  /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   if(errcode = di_list_length(1,0)){  /* scan one input element (no
                                          outputs) */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Length error.....\n");
   di_close();
}
```

---

di_mode

(not available on DI-221TC and DI-700 instruments)

- **Summary**

`int di_mode(mode);`

```
struct di_mode_struct{
   unsigned mode:4;    /* specifies the trigger mode as follows:
                                0 = trigger off
                                1 = analog
                                2 = digital
                                5 = software trigger */
   unsigned hystx:4;   /* on DI-400, DI-401, DI-410, DI-500, DI-510, DI-
                          720, DI-730, and DI-5001 instruments, specifies
                          the hysteresis index as follows (on all other
                          instruments, use 0):
```

hysteresis index	hysteresis + and - from level in LSB of 12-bit ADC
0	0
1	1
2	2
3	3
4	4
5	6
6	9
7	13
8	19
9	28
10	40
11	58
12	84
13	122
14	176
15	255

```
                          */
unsigned scnx:8;    /* input scan list index of trigger channel, 0 for
                       first channel and 0 for all instruments except
                       DI-400, DI-401, DI-410, DI-500, DI-510, DI-720,
                       DI-730, and DI-5001 */
unsigned trig_level;/* for an analog trigger; specify a 12 or 14-bit,
                       2’s complement, left justified value. For a
                       digital trigger; specify 1 of the 8 digital
                       input bits. */
unsigned trig_slope;/* 0 triggers on rising slope, 1 triggers on
                       falling slope */
unsigned trig_pre;  /* specifies the number of scans through the input
                       list for pre-trigger data */
unsigned trig_post; /* specifies the number of scans through the input
                       list for post-trigger data */
}*mode;
```

• Description

The **di_mode** function initializes the hardware for triggering on all instruments except the DI-221TC. The **di_start_scan** function is then called to start scanning. The **di_trig_status** function is called to check on trigger status.

`mode` allows you to specify no triggering, single scan triggering, or software triggering on all instruments except the DI-221TC.

NOTE

If you have a DI-221TC and you wish to perform triggering operations, you can reconfigure your DI-221TC to act like a DI-220, thus allowing triggering operations. This is done by running file DI220.BAT (for hardware with 16 channels or less) or file DI221M.BAT (for hardware with more than 16 channels) during startup instead of DI221.BAT.

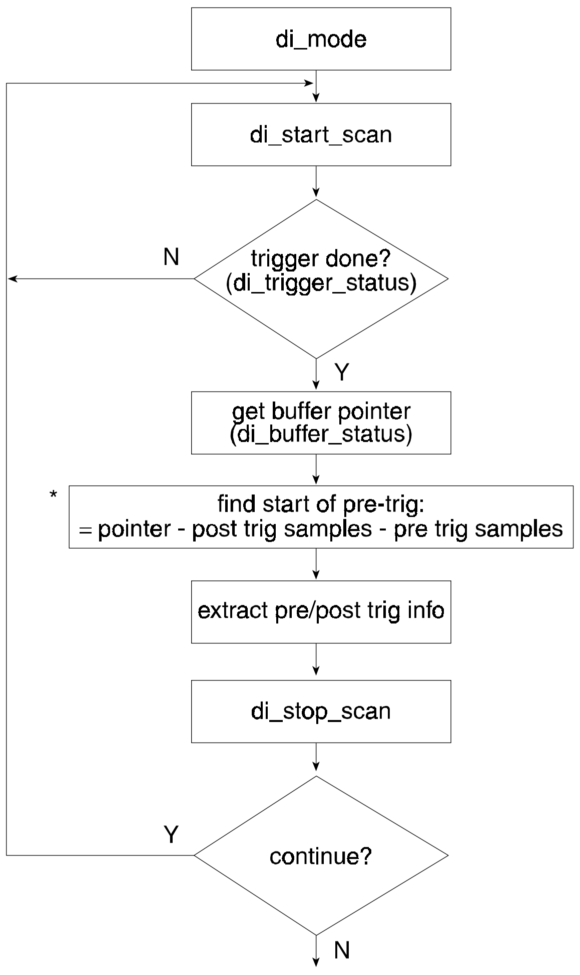
Triggering occurs on the first channel in the input scan list. For example, if channel 3 is listed as the first element in the input scan list, triggering will occur on channel 3.

Mode 0 (no triggering) allows data to be continuously acquired to your buffer the instant **di_start_scan** is called. Data acquisition does not stop when the buffer is full, the buffer simply gets overwritten.

Modes 1 (for analog trigger signals) and 2 (for digital trigger signals) are single scan trigger modes. The single scan mode is used when it is only necessary to monitor one trigger occurrence. For example; a buffer is allocated, scanning is started, the trigger condition occurs, *n* pre-trigger data samples and *m* post-trigger data samples are collected in the buffer (if pre- and/or post-trigger data is desired), and scanning is stopped.

Mode 5 (software triggering) is similar to mode 0 in that data is acquired to your buffer the instant **di_start_scan** is called, but differs from mode 0 in that you can specify the number of data points that will be acquired. When the buffer contains the desired number of data points, scanning is stopped. This mode is normally used without pre-trigger data. The value entered for `trig_post` specifies how many data points will be acquired.

The following flow chart illustrates a typical sequence of function calls required to extract data from the input buffer with pre-trigger data requested:



*2 LSB's = 0 and 1 at trigger point

`hystx` allows you to specify a hysteresis index. This index corresponds to a value shown in a table in the structure.

`scnx` is an input scan list index (pointer) to the trigger channel (0 for the first channel).

`trig_level` allows you to specify the point at which the trigger occurs. If you wish to trigger on an analog signal, specify a 12-bit, 14-bit, or 16-bit (depending on your instrument's capability), 2's complement, left-justified value. If you are triggering on a digital signal, specify one of the eight digital input bits as the trigger according to the following illustration:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	trig_level value
7	6	5	4	3	2	1	0	Digital input bits

For example, suppose you wanted to trigger on digital input bit 4. You would specify `mode.trig_level = 16.`

`trig_slope` allows you to trigger on the rising slope of the analog or digital signal (if `trig_slope` = 0) or on the falling slope of the analog or digital signal (if `trig_slope` = 1).

The digital signal is 1 if any of the specified digital input bits is high. A transition in the result is required to generate a digital trigger.

`trig_pre` allows you to specify how many data samples to acquire before the trigger condition occurs. If any amount of pre-trigger information is requested, data is automatically acquired to your buffer when `di_start_scan` is issued in anticipation of the trigger. If no pre-trigger information is requested (`trig_pre = 0`), no data is acquired in the buffer until the trigger condition occurs.

`trig_post` allows you to specify how many data samples to acquire after the trigger condition occurs.

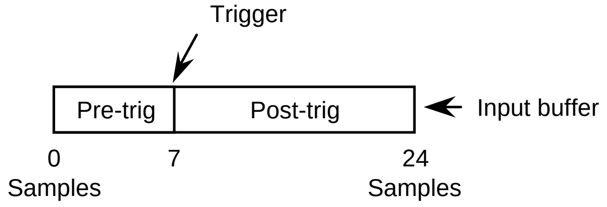
• Special Triggering Considerations

DI-200, DI-201, and DI-210 Only

The size of the input buffer you allocate for acquiring the data depends on several factors. At a *minimum*, this buffer must be large enough to hold the pre- and post-trigger data (if requested) for each element in the input list. For example, suppose we have 2 elements in the input list and we would like to acquire 4 scans of pre-trigger information and 8 scans of post-trigger information. The minimum input buffer size is calculated as follows:

$$2(4 + 8) = \textit{Input buffer size}$$

Using the **di_buffer_alloc** function, you would specify 24 samples of memory for your input buffer (1 sample = 1 word). Refer to the following illustration for a graphical representation of the input buffer:



The **di_trigger_status** function can be used to check the status of the sampling progress. When the specified amount of post-trigger samples have been acquired, the **di_trigger_status** function returns a "trigger done" condition. To extract the pre- and post-trigger data, you need to find the beginning of the pre-trigger data then write out *n* + *m* samples to a storage buffer. This is done by counting back *m* + *n* samples from the "trigger done" point to find the beginning of the pre-trigger data.

DI-220, DI-221TC, DI-222, DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001 Only

When in the triggering mode on DI-220, DI-221TC, and DI-222 instruments, the maximum number of samples that can be acquired is 8180 if doing an input operation only or 4090 if doing a simultaneous input/output operation.

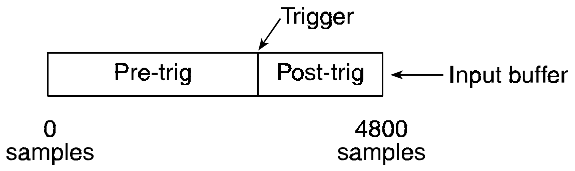
When in the triggering mode on DI-400, DI-401, and DI-410 instruments, the maximum number of samples that can be acquired is 15,000 samples, or 8192 if a simultaneous input/output operation is being done.

When in the triggering mode on DI-500, DI-510, DI-720, DI-730, and DI-5001 instruments, the maximum number of samples that can be acquired is 7,500 samples (or 4096 if a simultaneous input/output operation is being done), provided that the sample rate is less than or equal to the maximum communications rate, which is defined by the type of parallel port the instrument is connected to (refer to the **di_burst_rate** function for complete communications rate details).

The size of the input buffer you allocate for acquiring the data depends on several factors. At a *minimum*, this buffer must be large enough to hold the pre- and post-trigger data (if requested) for each element in the input list. For example, suppose we have 16 channels on the input list and we would like to acquire 200 samples of pre-trigger information and 100 samples of post-trigger information. The minimum input buffer size is calculated as follows:

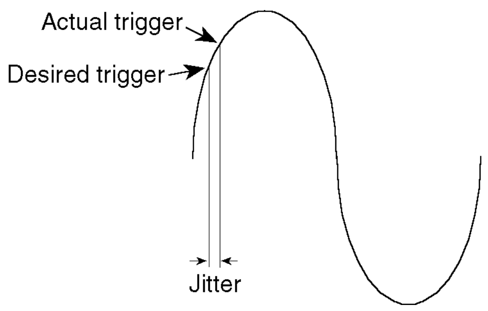
$$16(200 + 100) = \text{Input buffer size}$$

Using the **di_buffer_alloc** function, you would specify 4800 samples of memory for your input buffer (1 sample = 1 word). Note that the **di_buffer_alloc** function requires a minimum size of 4096 words for DI-220, DI-221TC, DI-222, DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001 instruments. Refer to the following illustration for a graphical representation of the input buffer:



Jitter (Trigger Uncertainty)

Another special triggering consideration that is common to all instruments is something called jitter. Jitter can be defined as trigger uncertainty and is illustrated as follows:



The maximum amount of jitter can be calculated as follows:

$$\text{max jitter} = \left[\frac{1}{\text{burstrate}}(L+1)\right](C_0+1)$$

where: L = length of the input or output list (whichever is greater).

C_0 = The counter value of the first element (element 0) of the input counter list.

- **Return Value**

This function returns error code.

- **Dependencies**

di_open
di_inlist

- **Example**

```
#include "200sdk.h"

int errcode;
struct di_mode_struct mode;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){          /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");

   mode.mode = 1;                    /* analog triggering enabled */
   mode.hystx = 0;                   /* reserved for compatibility */
   mode.scnx = 0;                    /* reserved for compatibility */
   mode.trig_level = 0;              /* trigger on zero crossing */
   mode.trig_slope = 0;              /* trigger on positive slope */
   mode.trig_pre = 1000;             /* 1000 pre trigger samples */
   mode.trig_post = 2000;           /* 2000 post trigger samples */
   if(errcode = di_mode(&mode)){    /* initialize mode */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   di_close();
}
```

---

di_open

- **Summary**

`int di_open(void);`

- **Description**

The **di_open** function opens the device for communication. This function has to be called before any of the other functions can be called.

- **Return Values**

DI_NO_ERR	No error
DI_DRIVER_ERR	Device driver not found
DI_COMM_ERR	Communication error

- **Dependencies**

none

- **Example**

```
#include "200sdk.h"

int errcode;
char errstr[255];

main()
{
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
```

```
      printf("Device installed successfully.....\n");
   di_close();
}
```

di_outlist

(not available on DI-401 and DI-700 instruments)

• Summary

`int di_outlist(output_list);`

```
struct di_outlist_struct{
  unsigned unipolar;     /* unipolar/bipolar. 0 = BI, 1 = UN (for all
                            instruments except DI-400, DI-410, DI-500, DI-
                            510, DI-720, DI-730, and DI-5001)

                                                 or

                            DAC1/DAC2. 0 = DAC1, 1 = DAC2 (for DI-400, DI-
                            410, DI-500, DI-510, DI-720, DI-730, and DI-
                            5001 instruments) */
  unsigned digital;       /* 1=buffer data is digital; 0=buffer data is analog */
  unsigned dig_out_enable /* 1=enables dig out, 0=disables dig out */
  unsigned dig_data;      /* digital data is D0 thru D7 */
  unsigned counter;       /* scan position counter */
  }*output_list;
```

• Description

The **di_outlist** function initializes the output list (output functions are not available on DI-401 instruments). This function has to be called after **di_list_length** and before **di_start_scan**. Note that the two LSB's of data in each location of the output buffer *must* be zero. Refer to the paragraph titled Input and Output Data Buffer Architecture at the beginning of this chapter for more details.

In the structure, when the buffer data is specified as digital, the digital output should be disabled. DMA data format is left justified.

Since the output scan list is capable of holding 16 entries, it is possible to program each channel in the output list for a different output rate. The equation for determining the value required for a specific output rate is as follows:

$$O = \frac{B}{L(C+1)}$$

Where: O = desired output rate of the output list entry, B = burst rate of the instrument, L = length of the input **or** output list **(whichever is greater)**, and C = "count weight" or output counter list entry.

Since simultaneous input and output operations are possible, some consideration must be given to input and output synchronization. The following table illustrates the order of each input/output operation, with respect to the other operations. In this example, there are 10 elements each in the input and output scan lists. Each input or output is referenced to its position in the input or output scan list.

Sample Number	Analog Input	Analog Output	Digital In DMA	Digital Out DMA	Digital Out Inlist	Digital Out Outlist
1	-	-	-	-	0	-
2	-	-	-	0	1	0
3	0	-	0	1	2	1
4	1	0	1	2	3	2
5	2	1	2	3	4	3
6	3	2	3	4	5	4
7	4	3	4	5	6	5
8	5	4	5	6	7	6
9	6	5	6	7	8	7
10	7	6	7	8	9	8
11	8	7	8	9	0	9
12	9	8	9	0	1	0
13	0	9	0	1	2	1
14	1	0	1	2	3	2
15	2	1	2	3	4	3
16	3	2	3	4	5	4
17	4	3	4	5	6	5
18	5	4	5	6	7	6
19	6	5	6	7	8	7
20	7	6	7	8	9	8

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_OUTLIST_ERR	Value in outlist structure out of range error

• Dependencies

di_open
di_list_length

- **Example**

```
   #include "200sdk.h"

   int errcode;
   #define ODIM 16;  /* largest value allowed */
   struct di_outlist_struct outlist[ODIM] = {0};      /* input list cleared */
   char errstr[255];

   main()
   {
   int i;
      if(errcode = di_open()){            /* open the device for comm */
         di_strerr(errcode,errstr);
         printf("%s",errstr);
      }
      else
         printf("Device installed successfully.....\n");

/* The following initializes scan list position 0 and must be repeated for all
   positions intended to be scanned */
      outlist[0].unipolar = 0;      /* bipolar */
      outlist[0].digital = 1;       /* buffer data is digital */
      outlist[0] dig_out_enable = 0 /* disable digital output */
      outlist[0].dig_data = 0x20;   /* digital data to be output */
      outlist[0].counter = 10;      /* init counter */

      di_list_length (ODIM, 0);
      if(errcode = di_outlist(outlist)){  /* initialize output list */
         di_strerr(errcode,errstr);
         printf("%s",errstr);
      }
      else
         printf("Output list initialized.....\n");
      di_close();
   }
```

---

di_set_data_mode

(DI-700 instruments only)

- **Summary**

`void di_set_data_mode(datamode);`

```
unsigned datamode;   /* sets DI-700 data mode: 1=16-bit, 0=14-bit */
```

- **Description**

di_set_data_mode can be passed an argument of 1 to put the DI-700 in 16-bit data mode, in which the least significant 2 bits of each data word are used for higher resolution instead of passing the inverse of the DI1 and DI0 digital inputs with the first channel data (note that special versions of WINDAQ recording and playback software are required to support 16-bit data files). Passing an argument of 0 returns the DI-700 to 14-bit data mode.

This function must be called before scanning starts.

- **Return Value**

void

- **Dependencies**

di_open
di_inlist or di_outlist (di_outlist not available on DI-401 and DI-700 instruments)
di_list_length
di_mode
di_buffer_alloc

---

di_start_scan

- **Summary**

```
int di_start_scan(void);
```

- **Description**

The **di_start_scan** function starts scanning.

On all instruments except the DI-500 Series, scanning will stop if data is sent from the output buffer to the instrument with either of the two least significant data bits set (the two LSB's of data *must* be zero to start the scanning process).

While scanning, you should not issue any other functions. If you call other functions while actively scanning, unexpected results may occur. If you wish to use any other functions, you must first stop scanning, call the desired function, then resume scanning.

- **Return Value**

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_START_SCN_ERR	Start scan error

- **Dependencies**

di_open
di_inlist or di_outlist (di_outlist not available on DI-401 and DI-700 instruments)
di_list_length
di_mode
di_buffer_alloc

- **Example**

```
#include "200sdk.h"

int errcode;
#define IDIM 256; /* largest value allowed */
struct di_inlist_struct inlist[IDIM] = {0};  /* input list cleared */
struct di_mode_struct mode = {0};            /* mode structure */
char errstr[255];
int * input_buffer;

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");

   inlist[0].chan = 5;           /* channel 5 */
   inlist[0].diff = 0;           /* single ended */
   inlist[0].gain = 0;           /* gain of 1 */
   inlist[0].unipolar = 0;       /* bipolar */
   inlist[0].ave = 0;            /* averaging off */
   inlist[0].counter = 100;      /* init counter */

   di_list_length (IDIM, 0);
   if(errcode = di_inlist(inlist)){ /* initialize the input list */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Input list initialized.....\n");

   mode.mode = 3;                /* continuous triggering enabled */
   mode.trig_level = 0;          /* trigger on zero crossing */
   mode trig_slope = 0;          /* trigger on positive slope */
```

```
    mode.trig_pre = 0;            /* no pre-trigger samples */
    mode.trig_post = 0;           /* no post-trigger samples */

    if(errcode = di_mode(mode)){  /* set mode */
       di_strerr(errcode,errstr);
       printf("%s",errstr);
    }
    else
       printf("Mode set.....\n");
    input_buffer = di_buffer_alloc (0, 40%);
    if (input_buffer = = NULL)
       printf("Failed to allocate buffer");
    if(errcode = di_start_scan()){   /* start scanning */
       di_strerr(errcode,errstr);
       printf("%s",errstr);
    }
    else
       printf("Scanning started.....\n");

    di_close();
 }
```

---

di_status_read

- **Summary**

`unsigned di_status_read(dest,num_scans);`

```
int far * dest;      /* destination for input data buffer */
unsigned num_scans;  /* number of data points to be copied */
```

- **Description**

The **di_status_read** function reads the status of the input buffer (opened by **di_buffer_alloc**) and copies the newest data points (`num_scans`) collected in the buffer from the previous call of this function to your specified destination (`dest`).

If there are not enough data points available in the input buffer (as specified by `num_scans`), this function will wait until there are enough available.

If `num_scans` is set to 0, this function will return the number of data points available without waiting.

- **Return Value**

The **di_status_read** function returns the number of data points in the buffer which still have not been copied.

• Dependencies

di_open	di_buffer_alloc
di_inlist	di_start_scan

• Example

```
#include "220sdk.h"

int *input_buffer, errcode, list_length, samples;
struct di_mode_struct mode = {0};
struct di_inlist_struct inlist[256] = {0};
char errstr[255];
int analog_in[1024];

main()
{
   list_length = 5;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   /* allocate 4096 words for input */
   if((input_buffer = di_buffer_alloc(0,4096) == NULL)
      printf("Insufficient memory or input buffer already allocated...\n");
   if(errcode = di_inlist(inlist))     /* Set up the input list */
      printf("Input list error...\n");
   if(errcode = di_mode(mode))         /* Set mode */
      printf("Mode error...\n");
   if(errcode = di_start_scan())       /* Start scanning */
      printf("Start scan error...\n");
   while(di_status_read (analog_in, 0) < 100);  /* Main loop executes until
                                                   100 data points are
                                                   accumulated. */
   di_status_read (analog_in, 100);       /* Copy 100 data points to the
                                             analog_in buffer. */
   if(errcode = di_close()){              /* close the device */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
}
```

di_stop_scan

• Summary

`int di_stop_scan(void);`

• Description

The **di_stop_scan** function stops scanning.

• Return Value

DI_NO_ERR	No error
DI_OPENED_ERR	Device not opened
DI_COMM_ERR	Communication error
DI_STOP_SCN_ERR	Stop scan error

• Dependencies

di_open
di_list_length
di_mode
di_start_scan

• Example

```
#include "200sdk.h"

int errcode;
#define IDIM 256; /* largest value allowed */
struct di_inlist_struct inlist[IDIM] = {0};  /* input list cleared */
struct di_mode_struct mode = {0};            /* mode structure */
char errstr[255];
int * input_buffer;

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");

   inlist[0].chan = 5;        /* channel 5 */
   inlist[0].diff = 0;        /* single ended */
   inlist[0].gain = 0;        /* gain of 1 */
   inlist[0].unipolar = 0;    /* bipolar */
   inlist[0].ave = 0;         /* averaging off */
   inlist[0].counter = 100;   /* init counter */

   di_list_length (IDIM, 0);
```

```
    if(errcode = di_inlist(inlist)){ /* initialize the input list */
       di_strerr(errcode,errstr);
       printf("%s",errstr);
    }
    else
       printf("Input list initialized.....\n");

    mode.mode = 1;             /* analog triggering enabled */
    mode.trig_level = 0;       /* trigger on zero crossing */
    mode.trig_slope = 0;       /* trigger on positive slope */
    mode.trig_pre = 0;         /* no pre-trigger samples desired */
    mode.trig_post = 0;        /* no post-trigger samples desired */

    if(errcode = di_mode(mode)){     /* set mode */
       di_strerr(errcode,errstr);
       printf("%s",errstr);
    }
    else
       printf("Mode set.....\n");
    input_buffer = di_buffer_alloc (0, 4096);
    if(errcode = di_start_scan()){      /* start scanning */
       di_strerr(errcode,errstr);
       printf("%s",errstr);
    }
    else
       printf("Scanning started.....\n");

    if(errcode = di_stop_scan()){    /* stop scanning */
       di_strerr(errcode,errstr);
       printf("%s",errstr);
    }
    else
       printf("Scanning stopped.....\n");

    di_close();
 }
```

---

di_strerr

- **Summary**

`int di_strerr(err_code,err_str);`

```
unsigned err_code;         /* error code returned by function call */
char     *err_str;         /* pointer to string to return error */
```

- **Description**

The **di_strerr** function maps `err_code` to an error-message string. The pointer to the string is passed to the function and the string is modified by the function. The maximum length of the error message string is 255 characters.

- **Return Value**

The **di_strerr** function returns a number of characters in `err_str`.

- **Dependencies**

di_open

- **Example**

```
#include "200sdk.h"

int errcode;
char errstr[255];

main()
{
int i;
   if(errcode = di_open()){            /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");

   di_close();

}
```

---

di_trigger_status

(not available on DI-700 instruments)

- **Summary**

`int di_trigger_status(void);`

- **Description**

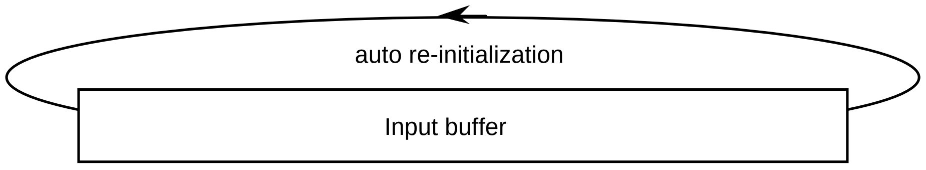
The **di_trigger_status** function returns the trigger status in the form of a 16-bit integer.

• Return Value

DI-200, DI-201, and DI-210 Only

Bits 15 thru 0
Number of times triggered

Bits 15 through 0 return the number of times triggered. Data is being written to the buffer constantly. Auto re-initialization occurs when a full buffer “wraps around” to the beginning, overwriting the existing data in the buffer.



For example, suppose your buffer is 1000 samples in size and you are sampling at 1 kHz. With these parameters, the buffer would auto re-initialize every 1 second.

DI-220, DI-221TC, and DI-222 Only
The two LSB’s return the trigger status as follows:

0	collecting pre-trigger data
1	collecting post-trigger data

To tell when the trigger is done, wait until the value returned by di_buffer_status has advanced by the number of pre- and post-trigger points.

DI-400, DI-401, DI-410, DI-500, DI-510, DI-720, DI-730, and DI-5001 Only
The two LSB’s return the trigger status as follows:

0	collecting pre- or post-trigger data (no distinction is made between pre- and post-trigger data)
2	trigger done (all pre- and post-trigger data has been collected)

• Dependencies

di_open
di_mode

• Example

```
#include "200sdk.h"
```

```
struct di_inlist_struct inlist(255) = {0};
int errcode;
char errstr[255];
struct di_mode_struct mode;
int * input_buffer;

main()
{
int i;
   if(errcode = di_open()){         /* open the device for comm */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   else
      printf("Device installed successfully.....\n");
   di_list_length (1,0);
   di_inlist (inlist);
   mode.mode = 1;                   /* analog triggering enabled */
   mode.trig_level = 0;             /* trigger on zero crossing */
   mode.trig_slope = 0;             /* trigger on positive slope */
   mode.trig_pre = 1000;            /* 1000 pre trigger samples */
   mode.trig_post = 2000;           /* 2000 post trigger samples */
   if(errcode = di_mode(&mode)){    /*initialize mode */
      di_strerr(errcode,errstr);
      printf("%s",errstr);
   }
   input_buffer = di_buffer_alloc (0, 4096);
   di_start_scan();                 /* start scanning */
   while(di_trigger_status() !=1)   /* wait for trigger to finish */
      ;
   printf("Trigger buffers filled.\n");
   di_close();
}
```

DATAQ Instruments, Inc.
241 Springside Drive
Akron, Ohio 44333
Telephone: 330-668-1444
Fax: 330-666-5434
E-mail: support@dataq.com

Vertrieb: / Distributed by:

ALTHEN GmbH Mess- und Sensortechnik
Frankfurter Str. 150-152 Tel.: +49 (0)6195 7006-0 http://www.althen.de
65779 Kelkheim / Deutschland Fax: +49 (0)6195 700666 E-Mail: info@althen.de