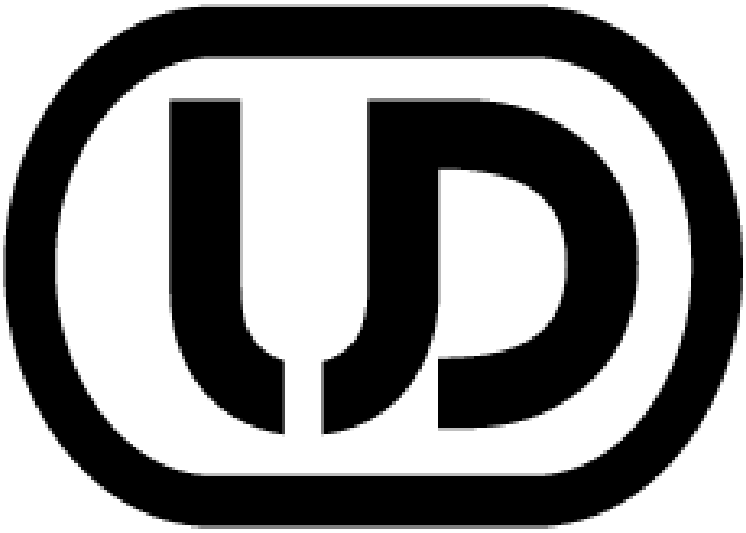
DIAMOND SYSTEMS CORPORATION

Universal Driver

Revision 6.01



Revision	Date	Comment
6.00	4/28/2009	Major update
6.01	6/3/2009	Minor update

FOR TECHNICAL SUPPORT
PLEASE CONTACT:

support@diamondsystems.com

Diamond Systems Corporation
1255 Terra Bella Ave.
Mountain View, CA 94043 USA
Tel 1-650-810-2500
Fax 1-650-810-2525
www.diamondsystems.com

Table of Contents

INDEX

1. Introduction

Universal Driver is a software toolkit that provides a C language interface capability for Diamond Systems' I/O boards. It is provided free with all of Diamond Systems' hardware products and may also be downloaded free from the Diamond Systems website (www.diamondsystems.com). It supports all the data acquisition features of Diamond Systems products, plus support for the watchdog timers on Diamond Systems CPU boards.

The UD 6.01 driver is fully backward compatible with the driver version 5.92 as well as the variants 5.92.1 and 5.92.2. UD 6.01 has added support for the latest SBC boards from DSC.

1.1. Download

You can download a recent version of this complete documentation for your use offline. It will unzip as a directory of web pages which you can view using your web browser. You will not need internet access to view these pages offline.

Download: DSCUD_Manual.zip (347K)

You can also download an older version of the documentation (*http://files.diamondsystems.com/DSCUD/DSCUD_Manual_5.91.zip*) in zip format. The zip file contains documentation in html files.

1.2. Supported Platforms

- Windows NT / 2000 / XP / XP Embedded
- Windows CE 6.0 R2
- Linux 2.4 - 2.6 kernels 2.6.23 and 2.6.24
- MS-DOS 6.22

If you would like the driver ported to another platform contact Diamond Systems technical support. We are always evaluating other systems and your input will help us determine which platforms are in demand. You can also license our driver source code and port the driver yourself.

1.3. Target Audience

Universal Driver software, and this manual, assume a familiarity with programming, the C language, and the PC hardware platform. It is also assumed that the reader is familiar with the target operating system and related C development tools for your platform like Borland, Visual Studio, or the GCC compiler. You also need to understand the features and capabilities of the boards you are using. You should read the hardware manual for each board in your project to get a more complete understanding of it before attempting to write software for it.

1.4. Demo Programs and Utilities

The Universal Driver software package includes example programs to help you understand the use of the driver functions and to use as a starting point for developing your own applications. Additional utility programs are provided for configuring serial port boards, reading out EEPROM data on autocalibrating boards, and demonstrating some boards' operation in Windows. Board functions supported by Universal Driver include the following: Analog input (A/D conversions) Analog output (D/A conversions) Digital I/O Counting, timing, and frequency/period measurement Interrupt-based A/D, D/A, and digital I/O User interrupts Autocalibration Note that Universal Driver does not support any serial port operations. On boards such as EMM-8M-XT that are listed as supported boards, only the digital I/O features are supported. In addition the utilities folder of the release disk contains some board configuration and test programs for the serial port boards.

1.5. Overview of Manual

This manual contains information on the following subjects.

- Installing and using the driver on each supported operating system
- Detailed documentation on all API functions and data structures
- Task oriented guides on subjects such as A/D interrupt operations, digital I/O
- Documentation on using watchdog timers

1.6. Getting Started

To use Universal Driver, first install it according to the instructions in Chapter 2. The easiest way to get running quickly is to first run some example programs to verify that driver installation is correct and your board is installed and running properly. After that you can start to modify the example programs, and then move on to develop your own applications. In many cases you can cut and paste code from the examples into your own application. The examples have been designed to be generic whenever possible and to include as much explanation as possible, so that they can be easily understood and modified to suite your particular application.

2. General Driver Issues

This chapter describes general issues surrounding the use of Universal Driver. These include installing the driver, compiling programs with the driver, and recommended coding conventions.

2.1. Instructions for DOS

In order to install the driver under DOS, perform the following steps: Copy the UD header file dscud.h from the include directory to any location on your hard drive. Copy the UD 16-bit static library file dscudbcl.lib from the lib directory to any location on your hard drive.

If you later wish to remove the driver, you will only need to delete the files that you copied to the hard drive.

2.2. Instructions for Windows 95 and 98

Installation

1. Unzip the UD archive file into its own directory, maintaining file path information.
2. Run **DSCUDInstall.exe**, located at the base of the directory tree.
3. Read the release information and check "I have read this very important information".
4. Click "Next".
5. If you want to copy the library files to a central location, you may enter that location here. Otherwise, select "Do not install developer files".
6. Click "Next".
7. At this point the system files will be installed and the installation will finish.
8. If you are running Windows 9x, you will have to reboot the computer to complete installation.

This will copy and install the system files (**dscudkp.vxd** and **windrvr.vxd**). You will be responsible for placing the library files (**dscud.dll**, **dscud.lib**, **dscud.h**) in their appropriate place for your compiler or program needs.

NOTE: Diamond Systems products are NOT plug and play and can not be managed through the Windows Device Manager. Access and management is done through direct I/O.

Uninstallation

1. Run **DSCUDUninstall.exe** located in the DSCUD archive.
2. Click on the UnInstall button

This will clean all UD system (.**vxd**) files. You will have to manually remove any copies of library files (**dscud.dll**, **dscud.lib**, **dscud.h**) that you may have made.

2.3. Instructions for Windows NT, 2000, XP

Installation

1. Unzip the UD archive file into its own directory, maintaining file path information.
2. Run **DSCUDInstall.exe**, located at the base of the directory tree.
3. Read the release information and check "I have read this very important information".
4. Click "Next".
5. If you want to copy the library files to a central location, you may enter that location here. Otherwise, select "Do not install developer files".
6. Click "Next".
7. At this point the system files will be installed and the installation will finish.
8. If you are running Windows NT/2K/XP, you *do not* have to reboot the computer to complete installation.

This will copy and install the system files (**dscudkp.sys** and **windrvr.sys**). You will be responsible for placing the library files (**dscud.dll**, **dscud.lib**, **dscud.h**) in their appropriate place for your compiler or program needs.

NOTE: Diamond Systems products are NOT plug and play and can not be managed through the Windows Device Manager. Access and management is done through direct I/O.

Uninstallation

1. Run **DSCUDUninstall.exe** located in the DSCUD archive.
2. Click on the UnInstall button

This will clean all UD system (**.sys**) files. You will have to manually remove any copies of library files (**dscud.dll**, **dscud.lib**, **dscud.h**) that you may have made.

2.4. Instructions for Windows CE

What is Windows CE?
Windows CE is an embedded operating system created by Microsoft. It targets embedded systems with limited physical space, both RAM and storage space, that require real-time performance.

Customers are allowed to customize the OS to only include features that are needed and write applications using API similar to those found on its desktop counterparts.

Supported Windows CE Version
Diamond Systems provides universal driver version 6.01 for Windows CE version 6.0. For more inquires contact support@diamondsystems.com

UD Files for Windows CE 6.0
The UD for Windows CE 6.0 is avaialble in a DLL format. The UD driver is separated into two different library files.

DSCUD.DLL - This is a kernel level driver file which handles interrupts. DSCUD_API.DLL - This is a user level DLL which needs to be statically linked with the user application. DSCUD.H - Include file which needs to be included with the user application.

Installation of UD in a Platform
The above lilsted files need to be available in a platform that the user has built in order to have UD driver available in the NK.BIN file.

DSCUD.DLL is required to be present in the project.reg registry file. The Project.reg file can be found in the Solutions explorer tab of Platform Builder.

The project.reg file must have the following entries for DSCUD to work.

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\DSCUD]

    "Dll"="dscud.dll"
    "Prefix"="DSC"
    "Order"=dword:1
    "SysIntr"=dword:15   ; IRQ 5
    "Index"=dword:1
    "Flags"=dword:10
        "UserProcGroup"=dword:3
```

The above entries are required so that the kernel can load DSCUD.DLL file automatically when the operating system boots. All the above described entries must be present in the registry. The SysIntr value is the value of the IRQ used by the DAQ circuitry + SYSINTR_FIRMWARE (0x10). If the board is going to use some other IRQ, the SysIntr value in the registry MUST be changed to reflect the change.

If the SysIntr value does not correspond to the IRQ for the DAQ circuit, the driver will not be able to handle interrupt requests.

UD for WinCE Development System Installation and Application Compilation

1. Unzip the UD archive into its own directory maintaining file path information. There should be three subdirectories; dll, include, lib.
2. When compiling an application, include the library dscud_api.lib, found in the "lib" subdirectory, and dscud.h header file, found in the "include" subdirectory.

For running the compiled EXE from the image, make sure that the DSCUD_API.DLL and DSCUD.DLL are both present in the Project.bib file of the platform.

MODULES

```
Name Path Memory Type
------------- ---------------------------------------------- -----------
 dscud.dll        $(_FLATRELEASEDIR)\dscud.dll                    NK SH
 dscud_api.dll    $(_FLATRELEASEDIR)\dscud_api.dll                        NK SH
```

UD Demos

Diamond Systems provides example programs for all our products. Users can download the demos at http://www.diamondsystems.com/support/software. Users may need to modify the path of libraries and include files

Windows CE BSP Installation

1. Unzip the "BSP zip file. The zip file will expand into its own directory, maintaining file path information.
2. Refer to the README and documentation in the directory for how to install and build a WinCE image.

2.5. Instructions for Linux

What is Linux?
Linux is a kernel developed and distributed under an open source model. Distributions such as Redhat, SuSE, and Debian package this kernel with other software to provide a complete operating system. The Universal Driver works directly with the Linux kernel, so any distribution should work fine as long as they used a supported Linux kernel version.

Supported Linux Versions
Linux kernel versions 2.4, and 2.6.x are supported.

Instructions If You Don't Need Interrupt Features
Many of the instructions that follow relate to building and installing a Linux kernel module provided by Diamond Systems. This kernel module is required to support high speed processing of interrupts from the data acquisition boards. However, if you do not intend to use interrupts from the board, there is no need for you to install this kernel module. Examples of interrupt functions are DscADSampleInt and dscUserInt. Simple functions like DscDAConvert, DscADSample, and digital I/O operations do not require interrupts.

If you don't need interrupts, you can simply extract the driver files, link to the driver library with your program, and copy your executable to your embedded system. You don't need to run the install.sh or load.sh shell scripts, or create the /dev/dscud device, do anything with kernel source code, or create a kernel module.

Things To Do Before You Install
The driver uses a Linux kernel module developed by Diamond Systems to handle interrupt processing. If you intend to use any interrupt related functions of the driver you will need this module. It is compiled during the installation procedures described below.

To build a Linux kernel module you must have the source code to the Linux kernel available on your system. Additionally, kernel modules are built for a specific version of the kernel, so if you build the module for one version and then attempt to load the module under another version, errors or unexpected results may occur.

The kernel version you select should be the version you will use on your embedded system. If you build a module for the version of Linux you are running on your desktop PC, this module will not run correctly on your embedded system unless it is running the exact same kernel version. Often this means that the driver will only load properly on one platform and not the other.

Download the Kernel Source
If your embedded system is using a standard version of the Linux kernel you can download the source from the www.kernel.org website. If it is a patched or custom kernel you will have to get the kernel source from the organization that provided the custom kernel.

Extract the kernel source on your development system so that it is available in /usr/src/linux. Below is an example of how this is normally done.

```
$ tar xvzf kernel-source-2.6.x.tar.gz
$ mv linux-2.6.x /usr/src/linux-2.6.x
$ ln -s /usr/src/linux-2.6.x /usr/src/linux
```

Prepare the Kernel Source For Use

The kernel source is not ready to be used for building kernel modules until it is configured. In some kernel configurations you must also begin the kernel compile process for it to include all the code required.

The first step is to configure the kernel. Run this command.

```
$ cd /usr/src/linux
$ make menuconfig
```

You will be presented with a menu that lists many options. You should select the correct CPU type and disable Symmetric multi-processing support under "Processor Type and Features". Then go to "Loadable Module Support" and make sure that "Enable loadable module support" is enabled. Exit the menu making sure it saves the kernel configuration.

Run this command to finish setting up the kernel source.

```
$ make depend
```

This kernel source is now ready to be used to build kernel modules.

Installing the Driver on your Development System

The driver is available as a simple tar gzip'd archive. To install the driver in this format you just extract the archive and move the new driver folder to the correct location on your system. Here is an example:

```
$ tar zxvf dscud-6.01.x.tar.gz
$ mv dscud-6.01.x /usr/local/
```

Once you have moved the directory, you must run an install script which will prepare the driver for use on your system. This will build a Linux kernel module which is required to use the interrupt related features of the driver. Here is an example:

```
$ cd /usr/local/dscud-6.01.x
$ chmod +x install.sh load.sh
$ ./install.sh
$ ./load.sh
```

See the next section if you experience any problems with the install script.

Compiling the Linux Kernel Module

You should run the install script to build the kernel module. Here is an example:

```
$ cd /usr/local/dscud-6.01.x
$ ./install.sh
```

This will compile the dscudkp.c file using the Linux kernel source located in /usr/src/linux and build a loadable kernel module called dscudkp.ko. This file will then be copied to /lib/modules/misc. When you run *load.sh* this module will be loaded into the kernel so that it can be used.

If you see an error complaining about "unresolved symbols" it means that the kernel you are running does not match the kernel source in /usr/src/linux, or the kernel has been configured differently, for example with or without SMP support.

Compiling With the Driver

The driver is a library which you link with your application. Here is an example:

```
$ gcc myapp.c -o myapp -I/usr/local/dscud-6.01.1 -L/usr/local/dscud-6.01.1 -ldscud-6.01 -lm -lpthread
```

Installing the Driver on your Embedded System

If you are not using any interrupt related functions of the driver, you can simply copy your compiled program to the embedded system and run it without problems. However, if you need interrupt functions you'll need to setup the Linux kernel module on your embedded system. Follow these steps.

1. Copy the dscudkp.ko file to /lib/modules/misc on your embedded system
2. Copy the load.sh script to /root/load.sh on your embedded system
3. Modify your system startup script to run /root/load.sh every time Linux boots.

On many systems there is an /etc/rc.sysinit , /etc/profile or /etc/rc.local file which you can use to add custom bootup commands.

Troubleshooting Linux Driver Problems

Here are some tips on troubleshooting problems on Linux.

- The driver must be run as the root user. If you would rather not do this to avoid potential security risks, you can use the setuid() call to switch to a different user after calling dscInit(). You should also change the file permissions for /dev/dscud so that your alternate user has read/write access to it.

- Check to make sure the /dev/dscud device exists. This device file is created by the load.sh script. It is required for communication with the dscudkp.ko kernel module. If it does not exist, run the load.sh script provided with the driver to create it.

- Run the lsmod command to make sure the dscudkp kernel module is loaded. If it is not listed, either the load.sh script was not run, or an error occured. Run the load.sh script manually to check for errors.

- If the module loader complains of "unresolved symbols" this indicates that the module was built for a different version of Linux than you are running. Make sure that /usr/src/linux on the system where the driver was built matches the version of the kernel you are running on the target system.

- If the module loader complaints of "kernel-module version mismatch" this also indicates that the kernel source that was downloaded in /usr/src/linux on your development system does not match the version on the target system.

- If the install.sh script complains of "missing or wrong headers" this indicates that the full kernel source was not found in /usr/src/linux. Make sure you followed the full process of preparing the kernel source for use, including running "make depend."

- If the module loader complains "dscudkp.ko will taint the kernel" you can disregard this message. This is the Linux community making you aware that you are loading a module provided by a 3rd party (in this case, Diamond Systems) and that they will want you to come to us instead of them for support with problems that come from this.

2.6. Calling Universal Driver API Functions

When using Universal Driver, you must always place four specific function calls in your program. Any application using the UD API should always start with calls to dscInit and dscInitBoard and end with calls to dscFreeBoard and DscFree. These calls are important in initializing and freeing resources used by the driver. Here is an example of the framework for an application using the driver:

```
#include "dscud.h"


DSCB dscb;
DSCCB dsccb;

int main()
{
	if ((result = dscInit(DSC_VERSION)) != DE_NONE)
		return result;

	dsccb.base_address = 0x300;
	dsccb.int_level = 5;
	dsccb.dma_level = 1;
	dsccb.clock_freq = 10000000L;

	if ((result = dscInitBoard(DSC_DMM32, &dsccb, &dscb)) != DE_NONE)
		return result;

	/* Application code goes here */

	if ((result = dscFreeBoard(dscb)) != DE_NONE)
		return result;

	if ((result = dscFree()) != DE_NONE)
		return result;

	return 0;
}
```

In the above example, DSC_VERSION, DSC_DMM32, and DE_NONE are macros defined in the included header file, *dscud.h*. Calls to any UD API function are always of the form, dsc[...], where [...] is a specific function (i.e., *ADSample* as in dscADSample). Most function calls using the driver require the initialization of a UD-specific structure (i.e. DSCCB, or DSCAIOINT) that is defined in *dscud.h* and described in the Data Type Reference.

Each board must be initialized separately, and each board must also have its own unique base address and I/O address range that is different from all other boards in the system and does not overlap with any other board or system resource (such as COM port or parallel port).

Universal Driver can control up to 10 boards simultaneously. Each board has its own unique handle generated by dscInitBoard. Most PC/104 systems will not support 10 boards simultaneously, so this limit should be satisfactory for all applications.

2.7. Error Trapping

All driver functions provide a basic error trapping mechanism that stores the last reported error in the driver. If your application is not behaving properly, you can check for the last error by calling the function dscGetLastError. This function takes an ERRPARAMS structure pointer as its argument.

Nearly all of the available functions in the UD API return a BYTE value upon completion. This value represents an error code that will inform you as to whether or not the function call was successful. You should always check if the result returns a DE_NONE value (signifying that no errors were reported), as the code below illustrates:

```
BYTE result;
ERRPARAMS errparams;

if ((result = dscInit(DSC_VERSION)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscInit failed: %s (%s)\n", dscGetErrorString(result), errparams.errstring);
   return result;
}
```

In this code snippet, the BYTE result of executing a particular driver function (dscInit in this case) is stored and checked against the expected return value (DE_NONE). Anytime a function does not complete successfully, an error code other than DE_NONE will be generated, and the current API function will terminate. The function dscGetErrorString provides a description of the error that occurred.

3. Common Task Reference

Universal Driver provides a set of functions that cover most of the features on Diamond Systems' I/O boards. Each board contains a different set of features with different valid parameter ranges. Not all driver functions apply to all boards, so not all of the tasks described here apply to all boards. The board reference information in Chapter 10 lists each function applicable to each board, along with the relevant features and valid parameter ranges for the various functions. The function reference in Chapter 7.3 lists the boards associated with each function. Below is a list of common tasks that are explained in this chapter. User interrupts are explained in Chapter 6.

Task Summary

- Performing an A/D.
- Performs A/D conversions one at a time on one channel or a range of channels.
- Performing an A/D Scan.
- Performs A/D conversions in groups on a range of channels.
- Interrupt-Based A/D Sample or Scan.
- Performs A/D conversions using interrupts for faster sampling rates.
- Performing a D/A Conversion.
- Performs a D/A conversion on a specific channel.
- Performing a D/A Conversion Scan.
- Performs a D/A conversion on multiple, specified channels.
- Interrupt-Based D/A Conversion Scan.
- Performs D/A conversions using interrupt-based I/O.
- Performing Digital I/O Operations.
- Performs bit and byte read and write operations.
- Checking Interrupt Operation Status.
- Checks the status of the currently-running interrupt operation.
- Performing an A/D Autocalibration.
- Performs an A/D auto-calibration on a selected A/D mode or all A/D modes.
- Performing a D/A Autocalibration.
- Performs a D/A auto-calibration.
- A/D Calibration Verification.
- Checks the error in A/D LSB counts after calibration.
- D/A Calibration Verification.
- Checks the error in D/A LSB counts after calibration.

NOTE: In the step-by-step instructions for performing each task, it is assumed that you will already be making the proper calls to dscInit(), dscInitBoard(), dscFreeBoard(), and dscFree() in your application.

3.1. Performing an AD Conversion

Description

When you perform an A/D conversion, you are getting a digital reading of an analog voltage signal applied to one of the A/D board's analog input channels. The A/D board uses a device called an analog-to-digital (A/D) converter to convert the real-world analog signal (temperature, pressure, tank level, speed, etc.) into a digital value that the digital computer electronics can handle. This digital value can be translated back to the input voltage using the conversion formulas provided in the board's user manual.

Typically the digital value, or the underlying voltage, is then converted to engineering units, such as temperature or speed, using the appropriate conversion formula for the device or sensor that generated the original voltage signal.

These conversion formulas are provided by the manufacturer of the device or sensor. Both conversions (digital to volts and volts to engineering units) can be combined into a single formula if desired.

The Universal Driver function name for A/D conversions is dscADSample.

Step-By-Step Instructions
Create and initialize an A/D settings structure (DSCADSETTINGS).

Call DscADSetSettings and pass it a pointer to this structure in order to setup the driver for A/D operations.

Call dscADSample and pass it a pointer to your sample variable (DSCSAMPLE) this will generate an A/D conversion and the result will be stored in your sample variable.

Example of Usage for dscADSample

```
...

DSCB dscb;
DSCADSETTINGS dscadsettings;
DSCSAMPLE dscsample;

...

/* Step 1 */

// These are example values; your application will vary.
// See DSCADSETTINGS struct and hardware manuals for details.

dscadsettings.current_channel = 0;
dscadsettings.gain = GAIN_1;
dscadsettings.range = RANGE_5;
dscadsettings.polarity = UNIPOLAR;
dscadsettings.load_cal = 0;
//scan_interval of 20 microseconds (default value), For DMM32, Hercules, DMM48
dscadsettings.scan_interval = 0;

/* Step 2 */

if ((result = dscADSetSettings(dscb, &dscadsettings)) != DE_NONE)
   return result;

/* Step 3 */

if ((result = dscADSample(dscb, &dscsample)) != DE_NONE)
   return result;

...
```

3.2. Performing an AD Scan

Description
An A/D scan is similar to an A/D sample except that it performs several conversions on a specified range of input channels with each function call. The Universal Driver function for performing an A/D scan is dscADScan.

Step-By-Step Instructions
Create and initialize an A/D settings structure (DSCADSETTINGS).

Call DscADSetSettings and pass it a pointer to this structure in order to setup the driver for A/D operations.

Create and initialize an A/D scan structure (DSCADSCAN.

Create and initialize a sample buffer of type DSCSAMPLE*.

Call dscADScan() and pass it a pointer to your scan structure - this will generate an A/D conversion for each channel in your scan range and the results will be stored in the sample_values element of your DSCADSCAN structure.

NOTE: You must always remember to allocate enough memory for your sample buffer (in this case, the WORD* sample_values). The driver assumes that your application has allocated at least the amount of memory given in the following formula:

Memory = sizeof(WORD) * (high_channel - low_channel + 1)

Example of Usage for A/D Scan

```
...
DSCB dscb;
DSCADSETTINGS dscadsettings;
DSCADSCAN dscadscan;

/* Step 1 */
dscadsettings.current_channel = 0;
dscadsettings.gain = 0;
dscadsettings.range = 0;
dscadsettings.polarity = 0;
dscadsettings.load_cal = 0;

/* Step 2 */
if ((result = dscADSetSettings(dscb, &dscadsettings)) != DE_NONE)
        return result;

/* Step 3 */
dscadscan.low_channel = 0;
dscadscan.high_channel = 3;

/* Step 4 */
dscadscan.sample_values = (DSCSAMPLE*)malloc(sizeof(DSCSAMPLE) *
(dscadscan.high_channel - dscadscan.low_channel + 1));
memset(dscadscan.sample_values, 0, sizeof(DSCSAMPLE) *
(dscadscan.high_channel - dscadscan.low_channel + 1));
```

```
/* Step 5 */
if ((result = dscADScan(dscb, &dscadscan, dscadscan.sample_values)) != DE_NONE)
        return result;
...
```

3.3. Interrupt-Based AD Sample or Scan

Description
Interrupt-based analog input operations are necessary when conversions must be performed at high sampling rates. They differ in operation from standard A/D samples and scans in that they perform a preset number of conversions per function call, rather than taking a single sample or scan. There are a number of different options that can be set for interrupt-based operations; these are explained in the interrupt-based operations chapter.

The Universal Driver functions for A/D sample and scan operations are dscADSampleInt and DscADScanInt.

Step-By-Step Instructions
Create and initialize an A/D settings structure (DSCADSETTINGS).

Call DscADSetSettings and pass it a pointer to this structure in order to setup the driver for A/D operations.

Create and initialize an analog I/O interrupts structure (DSCAIOINT).

Initialize the sample_values buffer in the DSCAIOINT structure by allocating sufficient memory to store the results of the interrupt operations. The memory required is:

Memory = sizeof(WORD) * num_conversions

Call dscADSampleInt or dscADScanInt and pass it a pointer to the above-mentioned DSCAIOINT structure to begin interrupt operation. The interrupt operations will now run in a separate system thread until either they reach the maximum number of conversions specified (one-shot mode only) or they are cancelled by a call to dscCancelOp.

Example of Usage for Interrupt-Based A/D Sample or Scan

```
...

DSCB dscb;
DSCADSETTINGS dscadsettings;
DSCAIOINT dscaioint;

...

/* Step 1 */

// These are example values; your application will vary.
// See DSCADSETTINGS struct and hardware manuals for details.

dscadsettings.current_channel = 0;
dscadsettings.gain = GAIN_1;
dscadsettings.range = RANGE_5;
dscadsettings.polarity = UNIPOLAR;
dscadsettings.load_cal = 0;
```

```
/* Step 2 */

if ((result = dscADSetSettings(dscb, &dscadsettings)) != DE_NONE)
        return result;

/* Step 3 */

dscaioint.num_conversions = 1024;
dscaioint.conversion_rate = 10000;
dscaioint.cycle = FALSE;
dscaioint.internal_clock = TRUE;
dscaioint.low_channel = 0;
dscaioint.high_channel = 3;
dscaioint.external_gate_enable = FALSE;
dscaioint.internal_clock_gate = FALSE;
dscaioint.fifo_enab = TRUE;
dscaioint.fifo_depth = 256;
dscaioint.dump_threshold = 512;

/* Step 4 */

dscaioint.sample_values = (WORD*) malloc(sizeof(WORD) *
        dscaioint.num_conversions);
memset(dscaioint.sample_values, 0, sizeof(WORD) * dscaioint.num_conversions);

/* Step 5 */

if ((result = dscADScanInt(dscb, &dscaioint)) != DE_NONE)
                return result;
...
```

NOTE: Boards can only achieve interrupt rates that divide evenly into their onboard counters, so the driver will attempt to find the rate closest to that which you specified.

3.4. Performing a DA Conversion

Description

When you perform a D/A conversion, you are essentially taking a digital value and converting it to a voltage value that you send out to the specified analog output. This output code can be translated to an output voltage using one of the equations described in the hardware manual.

The Universal Driver function for performing a D/A conversion is dscDAConvert.

Step-By-Step Instructions

Call dscDAConvert and pass it BYTE and DSCDACODE values for the channel and output code - this will generate a D/A conversion on the selected channel that will output the new voltage that you set with the output code.

NOTE: Once you generate a D/A conversion on a specific output channel, that channel will continue to maintain the specified voltage until another conversion is done on the same channel or the board is reset or powered down. For a 12 bit DAC, the range of output code is from 0 to 4095. For a 16-bit DAC, the rnage of output code is from 0 to 65535.

Example of Usage for D/A Conversion

```
...

DSCB dscb;
BYTE channel;
DSCDACODE output_code;
...

/* Step 1 */

channel = 0;
output_code = 4095;

if ((result = dscDAConvert(dscb, channel, output_code)) != DE_NONE)
   return result;
...
```

3.5. Performing a DA Conversion Scan

Description

A D/A scan conversion is similar to a D/A conversion except that it performs several conversions on a multiple, specified output channels with each function call.

The Universal Driver function for performing a D/A conversion scan is dscDAConvertScan.

Step-By-Step Instructions

Create and initialize an array of D/A scan conversion settings structures (DSCDACS).

Call dscDAConvertScan() and pass it a pointer to your scan structure array - this will generate a D/A conversion for each channel you set to enable.

NOTE: You must remember to initialize enough entries in your DSCDACS array to cover the number of conversions to be performed. If you do not, you will experience segmentation faults and invalid page faults.

Example of Usage for D/A Conversion Scan

```
...
#define NUM_DA_CHANNELS 4

DSCB dscb;
DSCDACS dscdacs;
int i;
...

/* Step 1 */

dscdacs.output_codes = (DSCDACODE*)malloc(sizeof(DSCDACODE) * NUM_DA_CHANNELS);

for (i = 0; i < NUM_DA_CHANNELS; i++)
{
        dscdacs.channel_enable[i] = TRUE;
```

```
        dscdacs.output_codes[i] = 4095;
}

/* Step 2 */

if ((result = dscDAConvertScan(dscb, &dscdacs)) != DE_NONE)
        return result;
...
```

3.6. Interrupt-Based DA Conversion Scan

Description
Interrupt-based analog output is useful for generating analog output values automatically at a specified rate, such as generating an analog waveform. It differs in operation from a single D/A conversion scan in that it generates output data continuously or for a specified number of samples each time the function is called. There are a number of different options that can be set for interrupt-based operations; these are explained in the interrupt-based operations chapter.

The Universal Driver function for performing interrupt-based D/A conversion scans is dscDAConvertScanInt.

Step-By-Step Instructions
Create and initialize an analog I/O interrupts structure (DSCAIOINT).

Set the entries of the sample_values buffer in the DSCAIOINT structure to the desired output codes. You must have num_conversion entries of size, WORD, in the buffer to ensure proper operation.

Call dscDAConvertScanInt() and pass it a pointer to the above-mentioned DSCAIOINT structure to begin interrupt operation. The interrupt operations will now run in a separate system thread until either they reach the maximum number of conversions specified (one-shot mode only) or they are cancelled by a call to dscCancelOp.

Example of Usage for Interrupt-Based D/A Scan

```
...

DSCB dscb;
DSCAIOINT dscaioint;
int i;

/* Step 1 */
dscaioint.num_conversions = 1024;
dscaioint.conversion_rate = 10000;
dscaioint.cycle = FALSE;
dscaioint.internal_clock = TRUE;
dscaioint.low_channel = 0;
dscaioint.high_channel = 3;
dscaioint.external_gate_enable = FALSE;

/* Step 2 */
dscaioint.sample_values = (WORD*)malloc(sizeof(WORD) * dscaioint.num_conversions);

for (i = 0; i < dscaioint.num_conversions; i++)
dscaioint.sample_values[i] = (WORD)4095;
```

```
/* Step 3 */
if ((result = dscDAConvertScanInt(dscb, &dscaioint)) != DE_NONE)
                        return result;
...
```

3.7. Performing Digital IO Operations

Description

The driver supports four types of direct digital I/O operations: input bit, input byte, output bit, and output byte. Digital I/O is fairly straightforward - to perform digital input, you provide a pointer to the storage variable and indicate the port number and bit number if relevant. To perform digital output, you provide the output value and the output port and bit number, if relevant.

The four Universal Driver functions described here are dscDIOInputByte, dscDIOInputBit, dscDIOOutputByte, dscDIOOutputBit, dscDIOSetBit, and dscDIOClearBit.

Step-By-Step Instructions

If you are performing digital input, create and initialize a pointer to hold the returned value of type BYTE*.

Some boards have digital I/O ports with fixed directions, while others have ports with programmable directions. Make sure the port is set to the required direction, input or output. Use function dscDIOSetConfig to set the direction if necessary.

Call the selected digital I/O function. Pass it a BYTE port value, and either a pointer to or a constant BYTE digital value. If you are performing bit operations, then you will also need to pass in a BYTE value telling the driver which particular bit (0-7) of the DIO port you wish to operate on.

Example of Usage for Digital I/O Operations

```
...
DSCB dscb;
BYTE port, bit;
BYTE digital_value;   // the value ranges from 0 to 255

/* 1. input bit - read bit 3 of port 0 (port 0 is in input mode) */
port = 0;
bit = 3;
if ((result = dscDIOInputBit(dscb, port, bit, &digital_value)) != DE_NONE)
   return result;

/* 2. input byte - read all 8 bits of port 0 (port 0 is in input mode) */
port = 0;
if ((result = dscDIOInputByte(dscb, port, &digital_value)) != DE_NONE)
   return result;

/* 3. output bit - 3 examples (assumes port 2 is in output mode) */
port = 2;
bit = 7;

/* 3a. dscDIOOutputBit(), set bit 7 of port 2 to 1, leave other bits alone */
if ((result = dscDIOOutputBit(dscb, port, bit, 1)) != DE_NONE)
```

```
   return result;

/* 3b. dscDIOSetBit(), set bit 7 of port 2 to 1, leave other bits alone */
if ((result = dscDIOSetBit(dscb, port, bit)) != DE_NONE)
   return result;

/* 3c. dscDIOClearBit(), set bit 7 of port 2 to 0, leave other bits alone */
if ((result = dscDIOClearBit(dscb, port, bit)) != DE_NONE)
   return result;

/* 4. output byte - set port 2 to "01010101" (port 2 is in output mode) */
port = 2;
if ((result = dscDIOOutputByte(dscb, port, 0x55)) != DE_NONE)
   return result;
...
```

3.8. Checking Interrupt Operation Status

Description
The function dscGetStatus is used to indicate the current status of any interrupt operation, including A/D sampling, D/A conversions, digital input, digital output, and user interrupts. It provides several status indicators (see the definition for the data structure DSCS on page 142 for complete details):
Operation status

Either OP_TYPE_NONE (no operation or operation completed) or any bit-wise combination of INT_TYPE_* (interrupt operations in progress)

Number of transfers

In one-shot mode, this is the number of input or output transfers have been completed. This is not necessarily the number of interrupts that have occurred, since if a FIFO is being used each interrupt will account for more than one transfer.

In recycle mode, this is the current position in the input/output buffer. When the buffer pointer reaches the end of the buffer, it will be reset to 0 and start counting up again.

Each individual A/D conversion in a scan counts as one transfer.

Total transfers

The total number of transfers that have occurred during this operation. In one-shot mode, total transfers is the same as number of transfers. In recycle mode, this number is not reset and should be used instead of the transfers variable for tracking the total number of conversions taken.

Overflow

During A/D sampling operations on boards with a FIFO, this indicates how many times the board's FIFO has overflowed during the current operation. During correct operation this value should always be zero. If this value is non-zero, data has been missed and the sampling rate is too high for the current system configuration.

The common convention for using this function is to start a particular interrupt function, and then go into a while loop that terminates when a specified condition has occurred, such as when a specified timeout has been reached or when the interrupt operation completes. Completion is indicated by the status indicator OP_TYPE_NONE.

Step-By-Step Instructions
Create and initialize a driver status structure (DSCS). Call dscGetStatus and pass it a pointer to this structure in order to return the current interrupt operation status. The two important elements to note in the DSCS structure are op_type and transfers. After calling dscGetStatus, these elements will contain the interrupt operation status and current number of transfers respectively.

Example of Usage for Checking Interrupt Status

```
/* run the interrupt operation for 30 seconds, then cancel */
#define TIMEOUT_MS 30000
DSCS dscs;
DWORD time_last, time_now;

/* Start interrupt function - dscADSampleInt() etc. */

dscGetTime(&time_last);
do
{
   dscGetTime(&time_now);
   if (time_now - time_last > TIMEOUT_MS)
   {
      dscCancelOp(dscb);
      break;
   }
   time_last = time_now;

   dscGetStatus(dscb, &dscs);
} while (dscs.op_type != OP_TYPE_NONE);
```

3.9. Performing an AD Autocalibration

Description
On supported boards (-AT boards), the user may auto-calibrate the board's A/D circuit in software, rather than undergoing a tedious process of manual calibration. Each A/D range has its own set of calibration settings for maximum accuracy. This function enables you to calibrate a single range or all ranges. To calibrate a single range, specify the range in the function call. To calibrate all ranges, specify 255 for the range.

Step-By-Step Instructions
Create and initialize an A/D auto-calibration settings structure (DSCADCALPARAMS).

Select the A/D range to calibrate. For an individual range, select 0-15. For all ranges, select 255.

Select the booot range. This is the range whose calibration settings will be recalled upon power-up.

Call dscADAutoCal and pass it a pointer to this structure in order to calibrate the A/D circuit on the board for the selected range(s).

NOTE: Performing an A/D auto-calibration will take about 1-3 seconds per range, depending on the board type and the range.

Example of Usage for A/D Autocalibration

```
...
DSCB dscb;
DSCADCALPARAMS dscadcalparams;
BYTE result;

...

/* Step 1 */
dscadcalparams.adrange = 255;                    // calibrate all A/D modes
dscadcalparams.boot_adrange = 8;                 // set power-up A/D mode to +/-10V

/* Step 2 */
if ((result = dscADAutoCal(dscb, &dscadcalparams)) != DE_NONE)
                        return result;
...
```

3.10. Performing a DA Autocalibration

Description

On supported boards (-AT boards), the user may auto-calibrate the board's D/A circuit in software, rather than undergoing a tedious process of manual calibration.

Autocalibration is used for two purposes:

- Calibrate the fixed D/A ranges, such as 0-5V or 0-10V
- Calibrate the programmable range, e.g. 0-1V or 0-4.096V

Autocalibration utilizes an autodetect feature to determine the jumper configuration of the D/A circuit, and then calibrates the circuit for the detected range.

NOTE: Performing a D/A autocalibration causes significant fluctuations in the D/A outputs. The fluctuations will either be from mid-scale to full-scale or from zero-scale to full-scale, depending on the board and the range. Be sure these fluctuations will not have adverse effects on any devices connected to the board before performing D/A autocalibration. Otherwise, disconnect the devices from the D/A outputs before performing D/A autocalibration.

Step-By-Step Instructions

Create and initialize a D/A auto-calibration settings structure (DSCDACALPARAMS).

Call dscDAAutoCal and pass it a pointer to this structure in order to calibrate the D/A circuit on the board.

NOTE: Performing a D/A auto-calibration takes about 2-5 seconds depending on the board and the mode.

Example of Usage for D/A Autocalibration

```
...
DSCB dscb;
DSCDACALPARAMS dscdacalparams;
```

```
BYTE result;

...

/* Step 1 - set the programmable D/A range to 4.096V.
Calibrate voltage point at 0.0.
(jumpers on board are set for programmable range) */

dscdacalparams.da_range = 4.096;
dscdacalparams.cal_point = 0.0;

/* Step 2 */
if ((result = dscDAAutoCal(dscb, &dscdacalparams)) != DE_NONE)
                        return result;
...
```

3.11. AD Calibration Verification

Description
On supported boards (-AT boards), after performing an A/D autocalibration, you can verify the accuracy of the calibration. This function will return the offset and gain error of each range in LSBs.

The default A/D calibration tolerance for all boards is +/-2LSBs. In general autocalibration will result in errors of +/-1LSB or less.

Step-By-Step Instructions
Create and initialize an A/D auto-calibration settings structure (DSCADCALPARAMS) .

Select the A/D range to verify. For an individual range, select the range number 0-15. For all ranges, select 255.

Call dscADCalVerify and pass it a pointer to this structure in order to verify the A/D calibration of the board. The elements ad_offset and ad_gain will contain the error in LSB counts for A/D operations. Values of less than 2 are within tolerance.

Example of Usage

```
...
DSCB dscb;
DSCDACALPARAMS dscdacalparams;
BYTE result;
...

/* Step 1 */
dscadcalparams.adrange = 8;   // select the range to verify; 255 = all ranges

/* Step 2 */
if ((result = dscADCalVerify(dscb, &dscadcalparams)) != DE_NONE)
                        return result;
...
```

3.12. DA Calibration Verification

Description
On supported boards (-AT boards), after performing a D/A autocalibration, you can verify the accuracy of the calibration. This function will return the offset and gain error of each range in LSBs.

The default D/A calibration tolerance for all boards is +/-2LSBs. In general autocalibration will result in errors of +/-1LSB or less.

Step-By-Step Instructions
Create and initialize a D/A auto-calibration settings structure (DSCDACALPARAMS).

Call dscDACalVerify and pass it a pointer to this structure in order to verify the D/A calibration of the board. The elements offset and gain of DSCDACALPARAMS will contain the error in LSB counts for D/A operations. Values of less than 2 are within tolerance.

Example of Usage for D/A Calibration Verification

```
...
DSCB dscb;
DSCDACALPARAMS dscdacalparams;
BYTE result;
...

/* Step 1 - verify the programmable range calibration from previous example */
dscdacalparams.da_range = 4.096;

/* Step 2 */
if ((result = dscDACalVerify(dscb, &dscdacalparams)) != DE_NONE)
  return result;
...
```

Example of Usage for D/A Calibration Verification in Helios

```
DSCB dscb; DSCDACALPARAMS dscdacalparams; BYTE result; ... dscInit( DSC_VERSION ) ;
dsccb.base_address = 0x300;
dsccb.int_level = 5;
dscInitBoard(DSC_HELIOS, &dsccb, &board);
dacalparams.darange_calibrate = i; result = dscDACalVerify( board, &dacalparams) ; ...
```

4. Interrupt-Based Operations

Interrupts are a means of running a piece of specialized code automatically in response to a hardware event. The hardware generates an interrupt request (IRQ) on the bus, which causes the processor to suspend its current task and run the special task. The special task is usually a routine to collect data from the board or to output data to the board. It can also include code to process the data. Interrupts offload timing and data transfer operations from the main program so it can concentrate on other tasks. The main program or programs can perform other tasks at the same time as the interrupts are occurring.

Interrupt-based sampling enables faster and more precise timing of I/O operations than is possible with standard single-operation function calls such as dscADSample. It is essential when operations must be performed at high sample rates, since usually the main program cannot manage the operation itself with reliability or without taking up too much processor time.

The Universal Driver contains several built-in functions for collecting (A/D) or outputting (D/A) data with interrupts. It also provides a "user interrupt" mechanism that enables you to run your own code each time an interrupt occurs. When run in conjunction with the built-in interrupt functions, your own code can run after the standard interrupt routine runs or it can run instead of the standard interrupt routine. Many boards also support user interrupts that are unrelated to A/D or D/A operations. User interrupts are described fully in the next chapter.

Every built-in A/D and D/A interrupt operation follows the same basic procedure. First, a structure containing the various options and parameters for the interrupt operation (i.e. DSCAIOINT) must be created and initialized. Then, a buffer that will either hold resulting samples taken (A/D) or contains a set of output values (D/A) must be created. Finally, the interrupt function is called. The board will then start to generate interrupts, which are caught and handled by a separate system thread. Depending on the mode of operation, the interrupt operation will either terminate when the current number of transfers reaches the maximum number specified (one-shot mode) or else will reset the counter and continue to generate interrupts (recycle mode). Any interrupt operation may be terminated at any time by calling dscCancelOp.

NOTE: UD supports multiple simultaneous interrupt operations. This allows you to:

A)Perform multiple different interrupt types on a single board, as long as this is physically supported by the board. For example, you can run A/D interrupts and D/A interrupts simultaneously on a single Diamond-MM-32-AT.

B)Perform simultaneous interrupt operations on different boards as long as they are on different IRQ levels. For example, you can run simultaneous A/D interrupts on two Diamond-MM-32-AT boards where board 1 is set to IRQ 5 and board 2 is set to IRQ 7.

C)Any combination of (A) and (B). For example you may run, all simultaneously, A/D and D/A interrupt operations on two Diamond-MM-32-AT boards, only if they each reside on different IRQs.

You cannot perform simultaneous interrupt operations on two different boards that occupy the same IRQ level.

4.1. AD Interrupt Operations

The behavior of A/D interrupt operations depends on three parameters. Some affect the behavior of the hardware, and some affect the behavior of the interrupt routine software. Together they determine the overall characteristics of the interrupt operation. A series of tables below provides a complete explanation of each possible configuration of options.

Single Conversions vs. Scan Conversions

In single-conversion mode, the board will take one sample from one channel each time it receives a clock pulse. The total sample rate is equal to the clock rate. If more than one channel is being sampled, the channels will be sampled on a rotating basis, with all samples evenly spaced apart in time. The sample rate for each channel is therefore the clock rate divided by the number of channels.

In scan mode, the board will take one sample from each channel in the scan list each time it receives a clock pulse. The samples are spaced closely together in time (5-20 microseconds depending on the board and the selected intersample period). The total sample rate is equal to the clock rate times the number of channels in the scan list. The sample rate for each channel is equal to the clock rate.

For both modes, the low and high channels in the sample range are determined by the low_channel and high_channel elements of the DSCAIOINT structure passed to the interrupt function.

Single-conversion mode vs. scan mode is determined by the name of the function called. For single-conversion mode, use dscADSampleInt, and for scan mode use dscADScanInt.

FIFO vs. No FIFO

A FIFO is a First In First Out buffer that is used to hold A/D data on the board. The FIFO serves two purposes: It can be used to enable the board to store data while waiting for the interrupt routine to respond to interrupt requests, so samples are not missed; and it also reduces the interrupt rate relative to the sample rate. The interrupt rate is determined by the formula

Interrupt rate = ((A/D clock rate) * (no. of samples per A/D clock)) / (FIFO threshold)

In No FIFO mode, the board will generate an interrupt each time the current A/D sample or A/D scan completes, depending on whether a sample or a scan operation is being performed. The interrupt routine will retrieve the sample or the set of samples for the scan. Note that if the board is in sample mode (not scan mode), and the number of channels being sampled is greater than one, each individual sample will still cause an interrupt request to be generated, and the interrupt routine will still read only one sample. The board's channel register is then incremented to the next channel in the list for the next sample.

In FIFO mode, the board will store A/D samples in its FIFO buffer until the FIFO threshold is reached. Once the threshold is reached, the board generates an interrupt request. The interrupt routine reads out the number of samples equal to the threshold value. This allows a dramatic reduction in interrupt rates, which causes more efficient use of processing power. For example, with a sample rate of 100,000 per second and a FIFO threshold of 256, the interrupt rate is about 391 per second, a manageable rate compared to 100,000 per second without the FIFO, an impossible rate.

FIFO vs. No FIFO is determined with two parameter in the DSCAIOINT structure passed to the interrupt function, fifo_enab and fifo_depth. Not all boards support FIFO mode, and of the boards that do, not all boards have programmable FIFO depth. These parameters are ignored by the software in cases where they do not apply.

One-Shot vs. Recycle

In one-shot mode, the interrupt routine will fill up the buffer with data one time and then terminate automatically. The parameter num_conversions indicates the number of samples to be taken. The data in the buffer is protected until the application program modifies it or uses it for another interrupt operation. The value returned by the function dscGetStatus indicates the total number of samples taken during the current interrupt operation.

In recycle mode, when the buffer is filled, the interrupt routine resets to the beginning of the buffer and continues, overwriting data already in the buffer. Thus the parameter num_conversions indicates the buffer size and not the number of samples to be taken. The buffer always contains the most recent n samples, where n is the size of the buffer.

You should use the function dscGetStatus to determine how many samples have been stored in the buffer. This function saves these values in the DSCS structure. The two structure members of interest are DSCS.transfers and DSCS.total_transfers. DSCS.transfers indicates how many samples have been placed in the buffer for the current cycle only. This value is reset to 0 at beginning of each cycle. DSCS.total_transfers indicates how many samples have been taken over the course of the entire operation. This value is not reset.

For operating systems which utilize the DSCAIOINT structure dump_threshold variable (Windows, Linux) you will see these DSCS.transfers and DSCS.total_transfers variables go up in dump_threshold sized increments. For other supported operating systems, you will see them go up by FIFO depth for dscADSampleInt operatings, or the scan size for dscADScanInt operations.

The larger the buffer, the more current data will be available at any particular time, and the longer that data will be available before being overwritten when the buffer pointer is reset to the beginning.

One-shot vs. recycle mode is determined by the cycle element in the DSCAIOINT structure passed to the interrupt function.

NOTE for Windows users: In order to terminate interrupts properly in one-shot mode, a call to dscGetStatus must be made after the maximum number of conversions is reached. The common procedure for running interrupt operations in one-shot mode entails calling dscGetStatus within a while loop until the process completes. This procedure will satisfy this condition.

Example Code for Recycle Mode
We have included some short example code below which may help to understand how recycle mode works. This is not meant to be used as a stand alone program. Instead it demonstrates the logic used to monitor the progress of an ongoing cycle mode A/D interrupt operation. In this example program a circular buffer of 10240 samples is used, and the program stops after taking 102400 samples. The program simply prints the A/D sample AD codes as they are collected.

```
ERRPARAMS errparams;
DSCB dscb;
BYTE result;
DSCAIOINT aio;
DSCS dscs;
int i;
DWORD last_total_transfers;
DWORD last_transfers;
int num_conversions = 10240;
int new_sample_count;
DWORD stop_after_transfers = 102400;
DWORD sleep_ms = 1000;

/* Board initialization code omitted */

/* The values used below are just for example and are not the
 * best choices for all boards.  In particular the fifo depth
 * and dump threshold should be changed to fit your needs.
 */
aio.num_conversions = num_conversions;
aio.conversion_rate = 1000;
aio.cycle = TRUE;
aio.internal_clock = TRUE;
```

```
aio.internal_clock_gate = FALSE;
aio.external_gate_enable = FALSE;
aio.fifo_enab = TRUE;
aio.fifo_depth = 32;
aio.high_channel = 0;
aio.low_channel = 0;
aio.dump_threshold = 32;

aio.sample_values = malloc(num_conversions * sizeof(DSCSAMPLE));
if ( aio.sample_values == NULL ) {
   printf("Operation failed: unable to allocate memory\n");
   exit(1);
}

if ((result = dscADSampleInt(dscb, &aio)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscADSampleInt failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}

last_total_transfers = 0;
last_transfers = 0;

do {
   dscSleep(sleep_ms);
   dscGetStatus(dscb, &dscs);

   if ( dscs.overflows ) {
      printf("Operation failed: FIFO overflowed\n");
      break;
   }

   if ( dscs.total_transfers == last_total_transfers ) {
      printf("Operation failed: no new samples taken in %d ms\n", sleep_ms);
      break;
   }

   new_sample_count = dscs.total_transfers - last_total_transfers;

   /* Number of new samples should never exceed the size of the circular buffer.  If
    * it does it means that either "sleep_ms" should be smaller so you check status
    * more often, or "num_conversions" should be bigger so the circular buffer is bigger
    */
   if ( new_sample_count > num_conversions ) {
      printf("Operation failed: not processing data fast enough.  %d samples lost\n",
         new_sample_count - num_conversions);
      break;
   }
```

```
    /* The code below uses the "transfers" counter to determine where in the circular
     * buffer the new sample data is located.  It could be only later in the buffer
     * than we already reported.  Or it could have looped back around to the start of
     * the circular buffer in which case the data to the end of the buffer, plus data
     * at the start of the buffer must be reported.
     */

    /* Case one: more data has been placed in the circular buffer after the
     * "last_transfers" position we checked previously.
     */
    if ( dscs.transfers > last_transfers ) {
       for ( i = last_transfers; i < dscs.transfers; i++ )
          printf("A/D sample: %d ADCODE\n", aio.sample_values[i]);

    /* Case two: more data has been placed in the circular buffer both after
     * the "last_transfers" and before it at the start of the buffer.
     */
    } else if ( dscs.transfers <= last_transfers ) {
       for ( i = last_transfers; i < num_conversions; i++ )
          printf("A/D sample: %d ADCODE\n", aio.sample_values[i]);
       for ( i = 0; i < dscs.transfers; i++ )
          printf("A/D sample: %d ADCODE\n", aio.sample_values[i]);
    }

    last_transfers = dscs.transfers;
    last_total_transfers = dscs.total_transfers;

    if ( dscs.total_transfers >= stop_after_transfers )
       break;

  } while ( dscs.op_type != OP_TYPE_NONE );

  dscCancelOp(dscb);

  /* Board and driver deallocation code ommitted */
```

4.1.1. Dump Threshold

On Windows 9x/NT/2000 and Linux platforms, the A/D interrupt handlers reside within the operating system's kernel space. For reasons pertaining to this fact, they have their own sample buffers separate and distinct from the buffer passed to the functions as a part of the DSCAIOINT structure. Typically, the buffer in the kernel is copied to the user-space buffer whenever the maximum number of transfers has been reached (i.e., at the end of one-shot mode, or with each cycle in recycle mode). However, in some instances it is useful to have access to a current sample buffer with greater frequency, particular at slow sample rates. To allow for this behavior, the parameter dump_threshold is provided in the DSCAIOINT structure.

When this parameter is set to 0, the dump threshold mechanism is disabled. When this parameter is non-zero, then anytime the current number of transfers reaches a multiple of the dump threshold, the buffer in kernel space will be

copied to the user's buffer. When operating in FIFO mode, the dump threshold must be a multiple of the FIFO depth. When operating in non-FIFO scan mode, the dump threshold must be a multiple of the scan range.

4.2. BDA Interrupt Operations

D/A interrupt operations are affected by only one parameter: recycle vs. one-shot. No Diamond Systems boards currently contain a FIFO or other hardware buffer for D/A operations. Each interrupt will output a single point to each specified channel. The application program simply builds a buffer containing the output waveform and then passes it to the function. In one-shot mode, the buffer is output a single time and then the function terminates. In recycle mode, the buffer is output repeatedly until the user calls dscCancelOp.

5. AD Interrupt Mode Reference

The tables on the following pages describe the behavior of the board and the driver during A/D interrupt operations with various combinations of settings. Each board has slightly different options and characteristics that affect which options apply. These differences are described here.

Note that even when FIFO is disabled, all boards with FIFO still use the FIFO to hold A/D conversions during scan operations. In these cases, FIFO disabled means the interrupt will occur after each scan, while FIFO enabled means the interrupt will occur after enough scans occur to reach or exceed the FIFO threshold.

Prometheus
The FIFO is always enabled. To "disable" the FIFO during interrupt operations, set FIFO depth = scan size (or 1 for single conversions). In this case the descriptions for Modes 0-3 apply. Otherwise modes 4-7 apply.

The FIFO depth is 48. The FIFO threshold is programmable. The power-on default FIFO threshold is 16.

Hercules-EBX
All tables below apply as described.

The FIFO depth is 2048. The FIFO threshold is programmable. The power-on default FIFO threshold is 1024.

Diamond-MM-48-AT
All tables below apply as described.

The FIFO depth is 2048. The FIFO threshold is programmable to 1024 or 256. The power-on default FIFO threshold is 1024.

Diamond-MM-32-AT
All tables below apply as described.

The FIFO depth is 512. The FIFO threshold is programmable. The power-on default FIFO threshold is 256.

Diamond-MM-16-AT, Diamond-MM-AT
All tables below apply as described.

The FIFO depth is 512. The FIFO threshold is fixed at 256.

Diamond-MM
This board has no FIFO. Modes 4-7 do not apply. For A/D scans, the interrupt occurs after the first A/D conversion is complete, and the interrupt routine completes the scan operation for the rest of the channels in the scan range. The interrupt rate is still equal to the scan rate.

Zircon-MM-DX
This board has no FIFO. Modes 4-7 do not apply. For A/D scans, the interrupt occurs after the first A/D conversion is complete, and the interrupt routine completes the scan operation for the rest of the channels in the scan range. The interrupt rate is still equal to the scan rate. Note that Zircon-MM-LC has no counter/timer and is therefore unable to generate A/D interrupts.

Mode 0: FIFO = 0, Scan = 0, Recycle = 0

Sampling Operation	Each A/D clock generates a single A/D conversion on the current channel. The channel register is then incremented to the next in the range (or resets to the lowest if already at the end). All A/D samples are evenly spaced apart in time.
Sample Rate	The overall sample rate is equal to the A/D clock rate. The sample rate for each channel in the range is equal to the A/D clock rate divided by the number of channels in the range.
Interrupt Occurrence	An interrupt occurs after each conversion is complete.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate.
Interrupt Operation	The interrupt routine reads a single A/D value from the board and stores it in the program's sample buffer.
Data Buffer	The buffer size is equal to the number of A/D samples to be taken.
Termination	After the specified number of conversions is complete, the interrupt operation automatically terminates. The user may also terminate early by calling the function dscCancelOp.

Mode 1: FIFO = 0, Scan = 0, Recycle = 1

Sampling Operation	Each A/D clock generates a single A/D conversion on the current channel. The channel register is then incremented to the next in the range (or resets to the lowest if already at the end). All A/D samples are evenly spaced apart in time.
Sample Rate	The overall sample rate is equal to the A/D clock rate. The sample rate for each channel in the range is equal to the A/D clock rate divided by the number of channels in the range.
Interrupt Occurrence	An interrupt occurs after each conversion is complete.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate.
Interrupt Operation	The interrupt routine reads a single A/D value from the board and stores it in the program's sample buffer.
Data Buffer	The buffer size must be equal to the number of samples num_conversions specified in the function call. This parameter is not related to the total number of samples to be taken, since recycle mode is enabled.
Termination	After the specified number of A/D conversions is complete, the interrupt operation returns to the beginning of the buffer and repeats. The user may terminate the operation at any time by calling dscCancelOp.

Mode 2: FIFO = 0, Scan = 1, Recycle = 0

Sampling Operation	Each A/D clock generates an A/D conversion on all channels in the programmed channel range, beginning with the low channel.
Sample Rate	The overall sample rate is equal to the A/D clock rate times the scan size. The sample rate for each channel in the range is equal to the A/D clock rate.
Interrupt Occurrence	An interrupt occurs after each entire scan is complete.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate.
Interrupt Operation	The interrupt routine reads a complete scan of data in from the board each time it runs and stores the data in the driver's sample buffer.
Data Buffer	The buffer size is equal to the number of scans to be performed times the scan size.
Termination	After the specified number of A/D conversions is complete, the interrupt operation automatically terminates. The user may also terminate early by calling the function dscCancelOp.

Mode 3: FIFO = 0, Scan = 1, Recycle = 1

Sampling Operation	Each A/D clock generates an A/D conversion on all channels in the programmed channel range, beginning with the low channel.
Sample Rate	The overall sample rate is equal to the A/D clock rate times the scan size. The sample rate for each channel in the range is equal to the A/D clock rate.
Interrupt Occurrence	An interrupt occurs after each entire scan is complete.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate.
Interrupt Operation	The interrupt routine reads a complete scan of data in from the board each time it runs and stores the data in the driver's sample buffer.
Data Buffer	The buffer size must be equal to num_conversions. This number must be an integral multiple of the scan size. This number is not related to the total number of samples, since recycle mode is enabled.
Termination	After the specified number of scans (the total number of conversions divided by the scan size) is complete, the interrupt operation will return to the beginning of the buffer and repeat the process indefinitely. The user may terminate the operation at any time by calling dscCancelOp.

Mode 4: FIFO = 1, Scan = 0, Recycle = 0

Sampling Operation	Each A/D clock generates a single A/D conversion on the current channel. The channel register is then incremented to the next in the range (or resets to the lowest if already at the end). All A/D samples are evenly spaced apart in time.
Sample Rate	The overall sample rate is equal to the A/D clock rate. The sample rate for each channel in the range is equal to the A/D clock rate divided by the number of channels in the range.
Interrupt Occurrence	An interrupt occurs after the FIFO reaches or exceeds the programmed threshold.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate divided by the FIFO threshold.
Interrupt Operation	The interrupt routine reads a number of A/D samples from the board equal to the FIFO depth and stores them in the program's sample buffer.
Data Buffer	The buffer size must be equal to the number of A/D samples. The buffer size and FIFO threshold must be chosen such that the buffer size is equal to an integral multiple of the FIFO threshold.
Termination	After the specified number of A/D conversions is complete, the interrupt operation automatically terminates. The user may also terminate early by calling the function dscCancelOp.

Mode 5: FIFO = 1, Scan = 0, Recycle = 1

Sampling Operation	Each A/D clock generates a single A/D conversion on the current channel. The channel register is then incremented to the next in the range (or resets to the lowest if already at the end). All A/D samples are evenly spaced apart in time.
Sample Rate	The overall sample rate is equal to the A/D clock rate. The sample rate for each channel in the range is equal to the A/D clock rate divided by the number of channels in the range.
Interrupt Occurrence	An interrupt occurs after the FIFO reaches or exceeds the programmed threshold.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate divided by the FIFO threshold.
Interrupt Operation	The interrupt routine reads a number of A/D samples from the board equal to the FIFO threshold and stores them in the program's sample buffer.
Data Buffer	The buffer size must be equal to the number of samples, num_conversions, specified in the function call. This number does not indicate the total number of samples to be taken, since recycle mode is enabled. The buffer size and FIFO threshold must be chosen such that the buffer size is equal to an integral multiple of the FIFO threshold.
Termination	After the specified number of conversions is complete, the interrupt operation will return to the beginning of the buffer and repeat the process indefinitely. The user may terminate the operation at any time by calling dscCancelOp.

Mode 6: FIFO = 1, Scan = 1, Recycle = 0

Sampling Operation	Each A/D clock generates an A/D conversion on all channels in the programmed channel range, beginning with the low channel.
Sample Rate	The overall sample rate is equal to the A/D clock rate times the scan size. The sample rate for each channel in the range is equal to the A/D clock rate.
Interrupt Occurrence	An interrupt occurs after the FIFO reaches or exceeds the programmed threshold and the current scan is complete.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate times the no. of channels in the scan range divided by the FIFO threshold.
Interrupt Operation	The interrupt routine reads a number of A/D samples from the board equal to the FIFO threshold and stores them in the driver's sample buffer.
Data Buffer	The buffer size must equal to the number of A/D samples to be taken. The buffer size and FIFO threshold must be chosen such that the buffer size is equal to an integral multiple of the FIFO threshold.
Termination	After the specified number of conversions is complete, the interrupt operation automatically terminates. The user may also terminate early by calling the function dscCancelOp.

Mode 7: FIFO = 1, Scan = 1, Recycle = 1

Sampling Operation	Each A/D clock generates an A/D conversion on all channels in the programmed channel range, beginning with the low channel.
Sample Rate	The overall sample rate is equal to the A/D clock rate times the scan size. The sample rate for each channel in the range is equal to the A/D clock rate.
Interrupt Occurrence	An interrupt occurs after the FIFO reaches or exceeds the programmed threshold and the current scan is complete.
Interrupt Rate	The interrupt rate is equal to the A/D clock rate times the no. of channels in the scan range divided by the FIFO threshold.
Interrupt Operation	The interrupt routine reads a number of A/D samples from the board equal to the FIFO threshold and stores them in the driver's sample buffer.
Data Buffer	The buffer size must be equal to the number of samples, num_conversions, specified in the function call. This number does not indicate the total number of samples to be taken, since recycle mode is enabled. The buffer size and FIFO threshold must be chosen such that the buffer size is qual to an integral multiple of the FIFO threshold.
Termination	After the specified number of conversions is complete, the interrupt operation will return to the beginning of the buffer and repeat the process indefinitely. The user may terminate the operation at any time by calling dscCancelOp.

6. User Interrupts

The User Interrupt feature of the Universal Driver enables you to run your own code when a hardware interrupt is generated by an I/O board. This is useful for applications that require special operations to be performed in conjunction with the interrupt or applications where you want to run your code at regular fixed intervals. Universal Driver includes example programs for each board with user interrupt capability to illustrate how to use this feature.

6.1. User Interrupt Types

The "After" Type
The user's function will be called every time the standard kernel-mode interrupt routine breaks to user-mode. This is convenient for those programs that want to add additional functionality to the interrupt routine. The break to user-mode occurs when a) the sample buffer has been filled or b) the selected dump threshold has been reached. The user's function does not need to manage any hardware interrupt details on the board, since all operation upkeep is performed in the kernel-mode code.

Note that this means that the user interrupt function is not called on every sample or on every interrupt.

- If you want your function to be called on every interrupt, set the dump threshold equal to the FIFO size.
- If you want your function to be called on every sample (or every scan), set the FIFO size to 1 (or the scan size) or disable the FIFO, and also set the dump threshold to 0.

The "Instead" Type
The user's function is called *instead of* the standard kernel routine. This method completely bypasses the routines inside of the dscudkp.sys/.vxd kernel code. When an interrupt is generated by the board, the kernel routine will immediately break out and call the user's function.

This means that the user defined function is responsible for everything that the standard interrupt function does, including (if necessary): resetting the interrupt flipflop on the board, removing A/D data from the FIFO, putting the data in a user-accessible buffer, checking for FIFO overflow, and determining if interrupts should be terminated. If you do not want to perform these functions yourself, consider the "Solo" type below.

The "Solo" Type
This interrupt is generated by either a counter/timer on the board or an external signal, depending on the available features on the board. When the interrupt is generated, it does not trigger any other functionality on the board (i.e. no A/D conversion).

This interrupt type is used typically when the user needs to call a background function at regular intervals, but does not desire A/D samples to be generated or processed.

Why is there no "Before" type?
Diamond Systems' Universal Driver uses a driver toolkit called WinDriver to handle interrupt operations on several operating systems. WinDriver runs interrupt functions in a Kernel mode separate from the application, where the user interrupt function resides. When an interrupt is generated, control transfers to the kernel code. With the structure of WinDriver, it is not possible to run non-kernel code before kernel code. There is no way to have the kernel interrupt routine "break out" of itself, run a user-mode function, and then "break in" to the kernel again. Therefore the user interrupt function must run after the kernel code containing the main interrupt function is finished.

6.2. The User Interrupt Function Interface

These are the lines from DSCUD.H that provide the interface to user interrupt functions. For complete details see the function descriptions and the datatype descriptions.

```
// User Interrupt Modes
#define USER_INT_CANCEL         0
#define USER_INT_AFTER          1
#define USER_INT_INSTEAD  2

// User Interrupt Sources
#define USER_INT_SOURCE_INTERNAL  0
#define USER_INT_SOURCE_EXTERNAL  1

// Type Definitions

typedef void (*DSCUserInterruptFunction) (void* parameter);

// Structures

// The following structure is used by all type of user interrupts
typedef struct
{
        DSCUserInterruptFunction func;
        BYTE int_mode;
} DSCUSERINTFUNCTION;

// The following structure is used only by the solo type user interrupts
typedef struct
{
        BYTE intsource;
        FLOAT rate;
        BYTE clksource;
        BYTE counter;
        DWORD int_type;
        DSCUserInterruptFunction func;
} DSCUSERINT;

// Function Prototypes

BYTE DSCUDAPICALL dscSetUserInterruptFunction(DSCB board, DSCUSERINTFUNCTION * dscuserintfunction);

BYTE DSCUDAPICALL dscClearUserInterruptFunction(DSCB board);

BYTE DSCUDAPICALL dscUserInt(DSCB board, DSCUSERINT* dscuserint,
        DSCUserInterruptFunction func);
```

6.3. Create a User Interrupt Function

The programmer must first write a user interrupt function. This function must have a void return value, and a single void* parameter. This is for WinDriver compatibility. It is not possible to pass parameters into the user interrupt function. When the user's function is called, the void* parameter will always be NULL (0). The function below is a sample function that will simply increment a global counter variable.

```
void MyUserInterruptFunction(void* param)

{
        counter++;
}
```

6.4. Instructions for After and Instead Type User Interrupts

The user must tell Universal Driver to run the user's function when interrupts occur, and select either "after" or "instead" operating mode:

```
...

DSCUSERINTFUNCTION dscuserintfunction;
dscuserintfunction.func = (DSCUserInterruptFunction) MyUserInterruptFunction;
dscuserintfunction.int_mode = USER_INT_AFTER;
// or dscuserintfunction.int_mode = USER_INT_INSTEAD;
dscSetUserInterruptFunction(boardhandle, &dscuserintfunction);

...
```

Once dscSetUserInterruptFunction is called, the driver will know to use the user's interrupt function for all subsequent interrupt operations on the specified board that are initiated by the currently running program.

Now, to make use of the user interrupt function, you must call one of the many driver functions that use interrupts (dscADSampleInt, dscADScanInt, dscDAConvertScanInt, dscUserInt, etc.)

The user interrupt function will run only when interrupts occur. If the interrupt function terminates (for example if dscADSampleInt running in one-shot mode completes, or if dscCancelOp is called), then the user interrupt function will stop running as well.

To disable the user's interrupt function, you have two options:

A. Call dscSetUserInterruptFunction to cancel:

- Set dscuserint.func = NULL.
- Set dscuserint.int_mode = DSC_USER_INT_CANCEL.
- Call dscSetUserInterrupt() again.

The example code below illustrates these 3 steps:

```
...
```

```
DSCUSERINTFUNCTION dscuserintfunction;
dscuserintfunction.func = NULL;
dscuserintfunction.int_mode = USER_INT_CANCEL;
dscSetUserInterruptFunction(boardhandle, &dscuserintfunction);

...
```

-or-

B. Call DscClearUserInterruptFunction:

```
...

dscClearUserInterruptFunction(boardhandle);

...
```

If you try to clear the user interrupt function (using either method) while an "instead" interrupt routine is actively running, you will receive an error and the function will not be cleared. You are only allowed to clear an "after" interrupt function while the routine is running.

When the user calls one of these interrupt enabling functions, the behavior of the user interrupt function is determined by the "int_mode" parameter in the DSCUSERINTFUNCTION structure.

If USER_INT_AFTER is used, then the user's function will be called every time the standard kernel-mode interrupt routine breaks to user-mode, as described above.

If USER_INT_INSTEAD is used, then the user's function will completely bypass the kernel-mode routine. Instead, every interrupt request will directly call the user's interrupt function, and will not perform any internal board upkeep. For this reason, if the user is using "instead" user interrupt functions in an A/D interrupt mode, the user interrupt function must perform all necessary upkeep (reset the interrupt flipflop, checking/reseting overflow, reading data from the FIFO, etc.). This mode should only be used if you are extremely familiar with the board's operation and interrupt requirements.

6.5. Instructions for Solo Type User Interrupts

The "solo" type user interrupt does not call a standard driver interrupt function like dscADSampleInt. Instead it has its own function named dscUserInt that simply passes control to the user interrupt function when the interrupt occurs. This function may be triggered either by a counter/timer on the board or by an external signal, depending on the available features on the board.

The solo type user interrupt also uses its own data structure, DSCUSERINT, which provides interrupt and counter/timer configuration data for the board.

The dscUserInt function does not call any kernel-mode routines or provide any extra monitoring features. The user is in charge of monitoring the interrupt and turning it off with dscCancelOp when desired.

When using "Solo" mode user interrupts, the function dscGetStatus does not function properly because it uses an internal operation counter that is not accessible by the user interrupt function.

To start the user interrupts, call dscUserInt by passing in your DSCUSERINT structure and the handle to your user interrupt function.

IMPORTANT: dscUserInt performs a hidden dscSetUserInterruptFunction on the user's interrupt function with mode USER_INT_INSTEAD (see section on "after" and "instead" interrupts.) This means that any previous calls to dscSetUserInterruptFunction will be invalidated.

The user interrupt function is not automatically uninstalled when dscCancelOp is called. To uninstall the function, you must call DscClearUserInterruptFunction before any subsequent interrupt operations.

Example code for solo-type user interrupts driven by on-board counter/timer with driver-programmed sample rate on a DMM-32-AT:

```
void UserIntSample1()

{
        DSCUSERINT dscuserint;
        dscuserint.intsource = USER_INT_SOURCE_INTERNAL;
        dscuserint.rate = 100.0; // 100Hz rate
        dscuserint.clksource = 0;          // use 10MHz on-board clock source
        dscuserint.counter = 0;
        dscUserInt(board, &dscuserint,
        (DSCUserInterruptFunction) MyUserInterruptFunction);
}
```

This will call MyUserInterruptFunction() at a rate of 100Hz.

Example code for solo-type user interrupts driven by on-board counter/timer with user-programmed sample rate on a DMM-32-AT:

```
void UserIntSample2()

{
        dscCounterDirectSet(...);          // separate call to program counter 0
        DSCUSERINT dscuserint;
        dscuserint.intsource = USER_INT_SOURCE_INTERNAL;
        dscuserint.rate = 0.0;             // 0 means don't program the counter here
        dscuserint.clksource = 0;
        dscuserint.counter = 0;
        dscUserInt(board, &dscuserint,
        (DSCUserInterruptFunction) MyUserInterruptFunction);
}
```

This will call MyUserInterruptFunction() at a rate determined by the dscCounterDirectSet function call.

Example code for solo-type user interrupts driven by external signal on a DMM-32-AT:

```
void UserIntSample3()

{
        DSCUSERINT dscuserint;
        dscuserint.intsource = USER_INT_SOURCE_EXTERNAL;
```

```
        dscuserint.rate = 0.0;              // not used since source is external
        dscuserint.clksource = 0;           // not used since source is external
        dscuserint.counter = 0;  // not used since source is external
        dscUserInt(board, &dscuserint,
        (DSCUserInterruptFunction) MyUserInterruptFunction);
}
```

This will call MyUserInterruptFunction() in response to an external TTL signal applied to the board's external trigger input.

To stop the user interrupts, you must cancel them with dscCancelOp():

```
void UserIntCancel()

{
        dscCancelOp(board);
}
```

6.5.1. After or Instead Modes

1. Create a user interrupt function with type void and a void* parameter.
2. Call dscSetUserInterruptFunction to install the user interrupt routine and select After or Instead mode.
3. To start interrupts, call the desired driver interrupt function, such as dscADSampleInt.
4. If the interrupt operation is in cycle mode, call dscCancelOp to terminate. If the operation is in one-shot mode, it will automatically terminate when the chosen number of samples is taken, and the user interrupt function will terminate as well.
5. To uninstall the user interrupt function, call dscSetUserInterruptFunction with Clear mode, or call DscClearUserInterruptFunction.

6.5.2. Solo Mode

1. Create a user interrupt function with type void and a void* parameter.
2. For counter/timer-based interrupts, you may program the counter/timer yourself using dscCounterDirectSet, or you may pass in the rate parameter to dscUserInt.
3. Call dscUserInt to install the function and initiate the interrupt operation.
4. When done, call dscCancelOp to terminate the interrupt operation.
5. To uninstall the user interrupt function, call dscSetUserInterruptFunction with Clear mode, or call DscClearUserInterruptFunction.

6.6. Board-Specific Information for Solo Type User Interrupts

All boards with interrupt capability may use the After and Instead types of user interrupts. However only the boards listed below have Solo type user interrupt capability. For each of these boards, all valid combinations for the DSCUSERINT structure are listed along with a description of the behavior. See the user interrupt example program in each board's example program set for more explanation.

Diamond-MM-AT
On Diamond-MM-AT, user interrupts are always driven by Counter 0. You have 2 choices for Counter 0's clock source. Diamond-MM-AT and Diamond-MM-16-AT behave identically with respect to user interrupts.

intsource	counter	clksource	Interrupt source
0	0	0	Counter 0, driven by on-board 100Khz clock
0	0	1	Counter 0, driven by external clock source connected to pin 29 on the I/O header (In0- pin)

Diamond-MM-16-AT
On Diamond-MM-16-AT, user interrupts are always driven by Counter 0. You have 2 choices for Counter 0's clock source. Diamond-MM-AT and Diamond-MM-16-AT behave identically with respect to user interrupts.

{\ border=1 ! intsource !! counter !! clksource !! Interrupt source |- | 0 || 0 || 0 || Counter 0, driven by on-board 100Khz clock |- | 0 || 0 || 1 || Counter 0, driven by external clock source connected to pin 29 on the I/O header (In0- pin) |}

Diamond-MM-32-AT
On Diamond-MM-32-AT, user interrupts are always driven by Counter 0. You have 3 choices for Counter 0's clock source:

intsource	counter	clksource	Interrupt source
0	0	0	Counter 0, driven by on-board 10MHz clock
0	0	1	Counter 0, driven by on-board 10KHz clock
0	0	2	Counter 0, driven by external clock source connected to pin 48 on the I/O header J3 (CLK0/DIN0 pin)

Diamond-MM-48-AT

intsource	counter	clksource	Interrupt source
0	0	0	Counter 1, driven by on-board reference clock
0	0	1	Counter 1, driven by external signal Clk0 on J3
1	0	0	Edges on enabled Digital I/O lines
2	0	0	Edges on enabled Optoinput lines

Emerald-MM-DIO

Emerald-MM-DIO has no counter/timers. It may be programmed to generate interrupts when a change of state is detected on digital I/O lines on ports 0-2. All parameters for dscuserint are ignored and should be left at 0. However the function must still be called to initiate interrupts. The user interrupt operation is configured with dscEMMDIOSetState.

Garnet-MM-24, Garnet-MM-48

On Garnet-MM, user interrupts are generated by the digital I/O lines C0 on each 82C55. There is no hardware configuration required, so all values for dscuserint are 0. Garnet-MM-24 has one interrupt using C0 on chip no. 1, while Garnet-MM-48 has two interrupts using C0 on each chip.

Onyx-MM

User interrupt features apply only to Onyx-MM. Onyx-MM-DIO has no interrupt capability. On Onyx-MM, dscUserInt.counter is not used and set to 0 in all cases.

intsource	counter	clksource	Interrupt source
0	0	0	Counter 0, clock source is In0 (J5 pin 1) *
0	0	1	Counter 0, clock source is on-board 4MHz oscillator
1	0	0	Counter 1, clock source is In1 (J5 pin 2) *
1	0	1	Counter 1, clock source is on-board 4MHz oscillator
2	0	0	Counter 2, clock source is In2 (J5 pin 7) *
2	0	1	Counter 2, clock source is on-board 4MHz oscillator
3	0	0	Counters 0 & 1 cascaded, clock source is In0 (J5 pin 1) *
3	0	1	Counters 0 & 1 cascaded, clock source is on-board 4MHz oscillator
4	0	0	Counters 1 & 2 cascaded, clock source is In1 (J5 pin 2) *
4	0	1	Counters 1 & 2 cascaded, clock source is on-board 4MHz oscillator
5	0	0	Counters 0 & 1 & 2 cascaded, clock source is In0 (J5 pin 1) *
5	0	1	Counters 0 & 1 & 2 cascaded, clock source is on-board 4MHz oscillator
6	0	0	82C55 #1 bit C0 (J3 pin 31), IRQ header 1
7	0	0	82C55 #2 bit C0 (J4 pin 31), IRQ header 2
8	0	0	External trigger pin (J5 pin 9), IRQ header 3

* In these modes, if clksource = 0 (external clock), then rate = 0 and the user must program the counter/timer(s) beforehand for the desired rate using dscCounterDirectSet(), since the driver has no way of knowing the input frequency in order to determine the proper divider.

Prometheus

On Prometheus, only Counter 1 is used for solo-type user interrupts.

intsource	counter	clksource	Interrupt source
0	0	0	Counter 1, driven by on-board 100KHz clock
0	0	1	Counter 1, driven by on-board 10MHz clock
1	0	0	External trigger, pin 25 on I/O header J14

Hercules-EBX

On Hercules, only Counter 1 is used for solo-type user interrupts.

intsource	counter	clksource	Interrupt source
0	0	0	Counter 1, driven by on-board 100KHz clock
0	0	1	Counter 1, driven by on-board 10MHz clock
1	0	0	External trigger

7. Watchdog Timer

Hercules-II EBX Watchdog Timer

The Watchdog timer circuit on the Hercules-II CPU module provides a means for protecting the CPU and the system in which it is installed against software crashes or hangups. Once the watchdog timer counters are loaded and the circuit is armed, the circuit must be either disarmed or retriggered before time runs out - otherwise a hard reset occurs. In normal operation either an internal sofware retrigger or a hardware retrigger from an external device will continuously retrigger the watchdog timer so that it never resets the system. The circuit can be programmed to drive the NMI interrupt signal and an external warning signal Watchdog Output (WDO) before reset occurs.

The watchdog timer circuit contains two counters. Counter 1 (WD1) is triggered by the input signal Watchdog In (WDI) or a software trigger. This timer is driven by a 10KHz clock and contains a 16-bit divisor. The maximum time delay for counter 1 is 65535 / 10KHz = 6.5535 seconds. When WD1 times out, it does several things:

1. It generates signal WDO, visible to external devices
2. It triggers counter 2 (WD2)
3. It also can be programmed to generate any combination of 2 hardware signals:
    - NMI Non-maskable interrupt
    - RESET Hardware reset

To prevent counter 1 from timing out, either the watchdog timer must be disabled or it must be retriggered. The retrigger may always be performed by a software retrigger command. It may also be performed by an external signal using the input signal WDI (not to be confused with the name of the counter WD1) when hardware retriggering is enabled.

Counter 2 (WD2) is also driven by the 10KHz clock, but it contains an 8-bit divisor. Its maximum delay is 255 / 10KHz = 25.5 milliseconds. When counter 2 times out, it generates an unconditional hardware reset. Once counter 2 begins counting, the only way to prevent it from timing out and generating a hardware reset is to disable the watchdog timer circuit. Counter 2 is provided to give the external device time to respond to the WDO signal and perform any necessary tasks before the system resets.

Helios SBC Watchdog Timer

Helios board has two watchdog timers viz. WDT0 and WDT1. Both of these timers can be set in the BIOS configuration or controlled by software using simple port I/O.

Both the watchdogs can be set to perform configurable functionality upon timer expiration. The watchdog timings are configurable from 1 sec to 512 sec. The valid times are 1sec, 2sec, 4sec, 8sec, 16sec, 32sec, 64sec, 128sec, 256sec and 512 sec. The watchdog timer can be programmed to perform a specific action when the timer expires. The action could be to Reset the system (recommended by DSC) or send an NMI or generate any of the interrupts from IRQ3 to IRQ15.

Both the watchdog timers are controlled by the Universal Driver. The watchdog APIs are discussed in the subsequent chapters of this manual.

7.1. Watchdog Timer API

The following code in DSCUD.H pertains to the watchdog timer features:

```
//Configuration options for Prometheus Watchdog
#define PROM_WD_TRIGGER_SCI            0x01
#define PROM_WD_TRIGGER_NMI            0x02
#define PROM_WD_TRIGGER_SMI            0x04
#define PROM_WD_TRIGGER_RESET          0x08
#define PROM_WD_WDI_ASSERT_FALLING_EDGE   0x10
#define PROM_WD_WDO_TRIGGERED_EARLY      0x20
#define PROM_WD_ENABLE_WDI_ASSERTION     0x40

//Configuration options for Hercules Watchdog
#define HERC_WD_TRIGGER_NMI            0x10
#define HERC_WD_TRIIGER_RST           0x08
#define HERC_WD_WDI_ASSERT_FALLING_EDGE   0x02
#define HERC_WD_WDO_TRIGGERED_EARLY      0x04
#define HERC_WD_ENABLE_WDI_ASSERTION     0x01


// SDWORD options below is any bitwise OR ( | ) combination of the above constants

//Functions
BYTE DSCUDAPICALL dscWatchdogEnable(DSCB board, WORD wd1, BYTE wd2, SDWORD options);
BYTE DSCUDAPICALL dscWatchdogDisable(DSCB board);
BYTE DSCUDAPICALL dscWatchdogTrigger(DSCB board);
```

7.1.1. Definitions of Constants

The definition of these constants is the same for both the Prometheus and Hercules (as long as the constant is supported by the board.)

XXXX_WD_TRIGGER_SCI	1 = Watchdog counter 1 will generate SCI on time-out 0 = no SCI occurs.
XXXX_WD_TRIGGER_NMI	1 = Watchdog counter 1 will generate NMI on time-out 0 = no NMI occurs.
XXXX_WD_TRIGGER_SMI	1 = Watchdog counter 1 will generate SMI on time-out 0 = no SMI occurs.
XXXX_WD_TRIGGER_RESET	1 = Watchdog counter 1 will generate a hardware reset immediately on time-out. In this case WD2 serves no purpose, since its function is to delay the assertion of hardware reset from the timeout of WD1. However a valid load value for WD2 must still be supplied to the function. 0 = no hardware reset occurs now (hardware reset will still be generated upon timeout of WD2).
XXXX_WD_WDI_ASSERT_FALLING_EDGE	1 = WDI will retrigger the watchdog timer on a falling edge. 0 = WDI will retrigger on a rising edge.

XXXX_WD_WDO_TRIGGERED_EARLY	1 = output on WDO will be generated one clock cycle before counter 1 times out. In this case, WDO can be used to retrigger WDI by wiring the two signals together. This causes a bypass condition that prevents the watchdog timer from ever timing out. 0 = output on WDO will be generated when watchdog counter 1 times out (normal operation).
XXXX_WD_ENABLE_WDI_ASSERTION	1 = hardware trigger input WDI will retrigger watchdog timer circuit and reload counter 1. Software retrigger command may also be used. 0 = input signal WDI is disabled; only software retrigger command may be used.

This information can also be found on page 104 of the ZFx86 Training Manual included in the \Prometheus\Docs folder on the Diamond Systems CD.

7.1.2. Watchdog Timer Functions

An overview of the watchdog timer functions is provided here. Full details of the functions are given in the UD Function Reference.

dscWatchdogEnable (DSCB board, WORD wd1, BYTE wd2, SDWORD options)
This function loads the watchdog counters to the specified values and configures the general behavior of watchdog counter 1, and arms the watchdog timer circuit.

dscWatchdogDisable (DSCB board)
This function disables the watchdog timer circuit on the specified board. A previously enabled but subsequently disabled watchdog timer may be re-enabled using the existing configuration by a subsequent call to dscWatchdogTrigger().

dscWatchdogTrigger (DSCB board)
This function retriggers the watchdog timer circuit, causing both counters WD1 and WD2 to be reloaded with their initial values. After a call to dscWatchdogTrigger(), the application has the amount of time specified in WD2 before it must call the function again.

If the watchdog timer has been disabled with a call to dscWatchdogDisable(), than this function will reenable the watchdog timer. It uses the existing watchdog configuration from the last dscWatchdogEnable() call, so any calls to dscWatchdogTrigger() before dscWatchdogEnable() will have undefined results.

In a typical application, dscWatchdogTrigger () is used in a continuous loop to keep retriggering the watchdog timer circuit as evidence that the system is running properly. If the loop ever crashes, or otherwise fails to call dscWatchdogTrigger() in time, the system will reset.

7.2. Application Instructions

Certain CPU boards provide the hardware retriggering of the watchdog timers. To use the watchdog timer, you must first decide if you want to enable hardware retriggering with an external signal. The AthenaII SBC has this feature enabled. Please refer to the AthenaII user manual for more details on how to use this feature.

All of the Diamond SBC boards also provide a software triggering of the watchdog timer. The general guidelines for using a software programmable watchdog timer are as below.

- Call the function dscWatchdogEnable to enable and configure the watchdog timer. Once this function is called, one of two things must happen continually at a rate faster than the timeout value of WD1:
- Your program must call dscWatchdogTrigger to retrigger the watchdog timer and restart WD1's countdown process.
- If hardware triggering is enabled, the external watchdog input signal must provide the selected active edge (rising or falling) to pin WDI.

If neither event happens before WD1 times out, the watchdog circuit will reset the system.

To disable the watchdog timer, call dscWatchdogDisable at any time. To reenable the watchdog timer with the same parameters, you can simply call dscWatchdogTrigger again. To reenable the watchdog timer with different parameters, you must call dscWatchdogEnable with the new parameters.

Example programs are provided with the CPU Universal Driver examples zip file to illustrate watchdog timer operation. This program will reset the system in order to demonstrate the functionality of the watchdog timer circuit!

8. UD Function Reference

BYTE DscInit (WORD version)
Initializes the Universal Driver. Must be called once at the beginning of each program.

BYTE DscFree() (void)
Frees the system resources used by the universal driver. Must be called once at the end of each program. This function will automatically call DscFreeBoard for all initialized boards.

BYTE DscInitBoard (BYTE boardtype, DscCB* dsccb, DSCB* board)
Initializes and sets the hardware settings of the given board. Must be called once for each board before using that board.

BYTE DscFreeBoard (DSCB board)
Frees the system resources used by the given board and disables any of the board's currently running interrupt operations. Must be called once for each board after finishing using that board. To free up all initialized boards call DscFree instead of this function.

A/D Functions
BYTE DscADSetSettings (DSCB board, DscADSettings* settings)
Sets the configuration for future A/D conversions.

BYTE DscADSetChannel (DSCB board, BYTE low_channel, BYTE high_channel)
Sets the channel range for future A/D conversions.

BYTE dscADSample (DSCB board, DSCSAMPLE* sample)
Performs a single A/D conversion on the currently selected channel.

BYTE dscADSampleAvg (DSCB board, DFLOAT* sample, int count)
Performs count A/D conversions on the currently selected channel and returns the average.

BYTE dscADSampleInt (DSCB, DSCAIOINT* dscaioint)
Performs A/D conversions using interrupt-based I/O with one conversion per A/D clock tick.

BYTE dscADScan (DSCB board, DSCADSCAN* dscadscan, DSCSAMPLE* sample_values)
Performs a set of A/D conversions on the selected range of channels.

BYTE DscADScanAvg (DSCB board, DSCADSCAN* dscadscan, DFLOAT* sample_values, int count)
Performs count sets of A/D conversions on the selected range of channels and returns the averages for each channel.

BYTE DscADScanInt (DSCB board, DSCAIOINT* dscaioint)
Performs A/D scans using interrupt-based I/O with one scan per A/D clock tick.

D/A Functions
BYTE DscDASetSettings (DSCB board, DSCDASETTINGS* settings)
Sets the configuration for future D/A conversions.

BYTE DscDASetPolarity (DSCB board, BYTE polarity)
Sets the current software-based polarity setting for D/A conversions.

BYTE dscDAConvert (DSCB board, BYTE channel, DSCDACODE output_code)
Performs a single D/A conversion on the given channel.

BYTE dscDAConvertScan (DSCB board,DSCDACS* dscdacs)
Performs a set of D/A conversions on multiple target channels.

BYTE dscDAConvertScanInt (DSCB board, DSCAIOINT* dscaioint)
Performs D/A conversion scans using interrupt-based I/O with one scan per interrupt.

DIO Functions
BYTE dscDIOSetConfig (DSCB board, BYTE* config_bytes)
Sets the DIO port configuration for future DIO operations.

BYTE dscDIOInputByte (DSCB board, BYTE port, BYTE* digital_value)
Receives a BYTE from a given digital input port.

BYTE dscDIOOutputByte (DSCB board, BYTE port, BYTE digital_value)
Sends a BYTE to a given digital output port.

BYTE dscDIOInputBit (DSCB board, BYTE port, BYTE bit, BYTE* digital_value)
Receives a bit value from a given digital input port at a specified bit location (0-7).

BYTE dscDIOOutputBit (DSCB board, BYTE port, BYTE bit, BYTE digital_value)
Sends a bit value to a given digital output port at a specified bit location (0-7).

BYTE dscDIOSetBit (DSCB board, BYTE port, BYTE bit)
Writes a 1 to the specified bit on the specified port.

BYTE dscDIOClearBit (DSCB board, BYTE port, BYTE bit)
Writes a 0 to the specified bit on the specified port.

Counter Functions
BYTE DscCounterSetRate (DSCB board, float hertz)
Sets the overall clock rate by using all individual counters.

BYTE DscCounterSetRateSingle (DSCB board, float hertz, DWORD ctr)
Sets the clock rate for a specific counter or group of counters.

BYTE dscCounterDirectSet (DSCB board, BYTE code, WORD data, BYTE ctr_number)
Sets the configuration for an individual counter.

BYTE DscCounterRead (DSCB board, DSCCR* dsccr)
Reads the configuration for all individual counters.

BYTE Dsc9513SetMMR (DSCB board, DSCCR* dscqmmmmr)
Programs the 9513 chip Master Mode Register on GPIO-MM 11/12.

BYTE Dsc9513SetCMR (DSCB board, DSCQMMCMR* dscqmmcmr)
Programs a Counter Mode Register on GPIO-MM 11/12.

BYTE Dsc9513SingleCounterControl (DSCB board, BYTE counter, BYTE action)
Performs load, arm, disarm, and save actions on a single counter on GPIO-MM 11/12.

BYTE Dsc9513CounterControl (DSCB board, DSCQMMMCC* dscqmmmcc, BYTE* status)
Performs load, arm, disarm, and save actions on a group of counters on GPIO-MM 11/12.

BYTE Dsc9513SetLoadRegister (DSCB board, BYTE counter, WORD value)
Loads a value into a counter's Load register on Quartz-MM.

BYTE Dsc9513SetHoldRegister (DSCB board, BYTE counter, WORD value)
Loads a value into a counter's Hold register on Quartz-MM.

BYTE Dsc9513ReadHoldRegister (DSCB board, BYTE counter, WORD* value)
Reads a counter's Hold register on Quartz-MM.

BYTE Dsc9513SpecialCounterFunction (DSCB board, DSCQMMSCF* dscqmmscf)
Performs special operations on a counter on Quartz-MM, including alarms, output values, and stepping.

BYTE Dsc9513MeasureFrequency (DSCB board, BYTE interval, BYTE source, WORD* pulses)
Measures the frequency of an input signal on Quartz-MM.

BYTE Dsc9513MeasurePeriod (DSCB board, BYTE frequency, DWORD* periods)
Measures the period of an input signal on Quartz-MM.

BYTE Dsc9513PulseWidthModulation (DSCB board, DSCQMMPWM* pwm)
Generates a PWM output on Quartz-MM.

Calibration Functions
BYTE DscSetCalMux (DSCB board, BOOL on)
Turns the calibration multiplexer on or off.

BYTE DscAACCommand (DSCB board, DWORD cmd)
Command to start, stop, trigger, and reset auto auto-calibration.

BYTE DscAACGetStatus (DSCB board, DSCAACSTATUS* status)
Command to get status of auto auto-calibration operation and register status

BYTE dscADAutoCal (DSCB board, DSCADCALPARAMS* params)
Performs an A/D auto-calibration on a selected A/D mode or on all A/D modes.

BYTE dscDAAutoCal (DSCB board, DSCDACALPARAMS* params)
Performs a D/A auto-calibration.

BYTE dscADCalVerify (DSCB board, DSCADCALPARAMS* params)
Verifies the accuracy of the A/D calibration.

BYTE dscDACalVerify (DSCB board, DSCDACALPARAMS* params)
Verifies the accuracy of the D/A calibration.

BYTE DscGetReferenceVoltages (DSCB board, DFLOAT* refs)
Reads the reference voltages from the EEPROM.

BYTE DscSetReferenceVoltages (DSCB board, DFLOAT* refs)
Sets the reference voltages in the EEPROM.

BYTE DscDAGetOffsets (DSCB board, DFLOAT* offsets, int count)
Retrieves the D/A offsets from the EEPROM.

BYTE DscDASetOffsets (DSCB board, DFLOAT* offsets, int count)
Stores the D/A offsets into the EEPROM.

BYTE DscSetTrimDac (DSCB board, DWORD trimDac, BYTE value)
Modifies the onboard autocalibration TrimDAC values.

User Interrupt Functions
BYTE dscSetUserInterruptFunction (DSCB board, DSCUSERINTFUNCTION* dscuserintfunction)
Installs a user interrupt function on all interrupt types in the driver for later use.

BYTE DscSetUserInterruptFunctionType (DSCB board, DSCUSERINTFUNCTION* dscuserintfunction, DWORD int_type)
Installs a user interrupt function on all interrupt types in the driver for later use.

BYTE DscClearUserInterruptFunction (DSCB board)
Uninstalls all user interrupt functions from this board.

BYTE DscClearUserInterruptFunctionType (DSCB board, DWORD int_type)
Uninstalls all user interrupt functions from this board.

BYTE dscUserInt (DSCB board, DSCUSERINT* dscuserint, DSCUserInterruptFunction)
Starts execution of user interrupts.

Watchdog Timer Functions
BYTE dscWatchdogEnable (DSCB board, WORD wd1, BYTE wd2, SDWORD options)
Enables the watchdog timer circuit on the CPU boards supported and configures it according to the given parameters.

BYTE dscWatchdogDisable (DSCB board)
Disables the watchdog timer circuit on the CPU board.

BYTE dscWatchdogTrigger (DSCB board)
Retriggers the watchdog timer circuit on the CPU board.

D/A Wave Form Generator Functions
BYTE DscWGCommand (DSCB board, DWORD cmd)
Function to stop, start, trigger, and reset the D/A wave form generator.

BYTE DscWGConfigSet (DSCB board, DSCWGCONFIG* config)
Sets up the D/A wave form depth, number of output per trigger, and input source.

BYTE DscWGBufferSet (DSCB board, DWORD address, DSCDACODE value, DWORD channel, BOOL simul)
Sets D/A output code for D/A wave form generator.

Error Functions
BYTE dscGetLastError (ERRPARAMS errparams)
Returns the most recent error that occurred during a Universal Driver function call.

char* dscGetErrorString (BYTE error_code)
Returns the corresponding error string for the given error code.

Optoinput Functions
BYTE DscOptoInputByte (DSCB board, BYTE port, BYTE * optoValue)
Simultaneously gets the state of 8 optoinputs from the board.

BYTE DscOptoInputBit (DSCB board, BYTE port, BYTE bit, BYTE* optoValue)
Gets the state of a single optoinput from the board.

BYTE DscOptoGetPolarity (DSCB board, BYTE* polarity)
Gets the polarity of the optoinputs.

BYTE DscOptoGetState (DSCB board, DSCOPTOSTATE* state)
Gets the overall state of optoinputs from the board.

BYTE DscOptoSetState (DSCB board, DSCOPTOSTATE* state)
Sets the overall state of optoinputs on the board.

Relay Functions
BYTE DscSetRelay (DSCB board, BYTE relay, BYTE value)
Sets the state of one relay on the board.

BYTE DscGetRelay (DSCB board, BYTE relay, BYTE* value)
Gets the state of one relay on the board.

BYTE DscSetRelayMulti (DSCB board, BYTE relayGroup, BYTE value)
Simultaneously sets the state of multiple relays on the board.

BYTE DscGetRelayMulti (DSCB board, BYTE relayGroup, BYTE* value)
Simultaneously gets the state of multiple relays on the board.

IR104 Functions
BYTE DscIR104SetRelay (DSCB board, BYTE relay)
Sets an individual relay on an IR104 board.

BYTE DscIR104ClearRelay (DSCB board, BYTE relay)
Clears an individual relay on an IR104 board.

BYTE DscIR104RelayInput (DSCB board, BYTE relay, BYTE* value)
Reads back an individual relay's current state on an IR104 board.

BYTE DscIR104OptoInput (DSCB board, BYTE opto, BYTE* value)
Reads an individual optoisolated digital input bit on an IR104 board.

EMMDIO Functions

BYTE dscEMMDIOGetState (DSCB board, DSCEMMDIO* state)
Returns the current configuration for an Emerald-MM-DIO board.

BYTE dscEMMDIOSetState (DSCB board, DSCEMMDIO* state)
Sets the current configuration for an Emerald-MM-DIO board.

BYTE dscEMMDIOResetInt (DSCB board, DSCEMMDIORESETINT* resetinfo)
Resets interrupt status on an Emerald-MM-DIO board.

Miscellaneous Functions

BYTE DscGetTime (DWORD *ms)
Get a millisecond precision clock time

BYTE DscSleep (DWORD ms)
Waits for a specified number of milliseconds.

BYTE DscGetEEPROM (DSCB board, DWORD address, BYTE* data)
Reads data from the EEPROM at the specified address.

BYTE DscSetEEPROM (DSCB board, DWORD address, BYTE data)
Writes data to the EEPROM at the specified address.

BYTE dscGetStatus (DSCB board, DSCS* status)
Returns the current status of any interrupt operations.

BYTE dscCancelOp (DSCB board)
Terminates any currently running interrupt operation.

BYTE DscCancelOpType (DSCB board, DWORD int_type)
Terminates any currently running interrupt operation.

BYTE DscInp (WORD address, BYTE *value)
Direct I/O read a byte from the address

BYTE DscInpw (WORD address, WORD *value)
Direct I/O read a word from the address

BYTE DscInpl (WORD address, DWORD *value)
Direct I/O read a double word from the address

BYTE DscInpws (WORD address, WORD *value, WORD n)
Direct I/O read n words from the address

BYTE DscOutp (WORD address, BYTE value)
Direct I/O write a byte to the address

BYTE DscOutp (WORD address, WORD value)
Direct I/O write a word to the address

BYTE DscOutpw (WORD address, WORD value)
Direct I/O write a word to the address

BYTE DscOutpl (WORD address, DWORD value)
Direct I/O write a double word to the address

BYTE DscOutpws (WORD address, WORD *buffer, WORD n)
Direct I/O write n words to the address

BYTE DscGetBoardMacro (char* boardtype, BYTE* macro)
Returns the corresponding board macro for the given board type string.

BYTE DscRegisterRead (DSCB board, WORD address, BYTE* data)
Reads a BYTE from an I/O port address.

BYTE DscRegisterWrite (DSCB board, WORD address, BYTE data)
Writes a BYTE to an I/O port address.

BYTE DscGetFPGARev (DSCB board, WORD* fpga)
Returns the FPGA revision of the board.

BYTE DscSetSystemPriority (DWORD priority)
Sets the system priority for the interrupt handling thread.

BYTE DscADCodeToVoltage (DSCB board, DSCADSETTINGS adsettings, DSCSAMPLE adsample, DWORD *voltage)
Unit conversion utility function for converting AD units to voltage.

BYTE DscVoltageToADCode (DSCB board, DSCADSETTINGS adsettings, DWORD voltage, DSCSAMPLE *adsample)
Unit conversion utility function for converting voltage to AD units.

BYTE DscDACodeToVoltage (DSCB board, DSCDASETTINGS dasettings, DSCDACODE dacode, DWORD *voltage)
Unit conversion utility function for converting DA units to voltage.

BYTE DscVoltageToDACode (DSCB board, DSCDASETTINGS dasettings, DWORD voltage, DSCDACODE*dacode)
Unit conversion utility function for converting voltage to DD units.

BYTE DscEnhancedFeaturesEnable (DSCB board, BOOL enable)
Enables/disables enhanced features

9. Data Type Reference

Structures

DSCCB
Structure containing hardware settings for the current board.

DSCADSETTINGS
Structure containing current A/D conversion settings.

DSCDASETTINGS
Structure containing current D/A conversion settings.

DSCADSCAN
Structure containing A/D scan settings.

ERRPARAMS
Structure containing DSCUD error information.

DSCAIOINT
Structure containing interrupt-based analog I/O settings.

DSCAUTOCAL
Legacy reference to maintain backwards-compatibility; refer to DSCADCALPARAMS

DSCADCALPARAMS
Structure containing A/D auto-calibration settings.

DSCDACS
Structure containing D/A conversion scan settings.

DSCDACALPARAMS
Structure containing D/A auto-calibration settings.

DSCS
Structure containing interrupt operation status information.

DSCCS
Structure containing individual counter information.

DSCCR
Structure containing information on all counters.

DSCEMMDIO
Structure containing current configuration of an Emerald-MM-DIO board.

DSCEMMDIORESETINT
Structure used to reset EMM-DIO user interrupts.

DSCUSERINT
Structure containing configuration data for user interrupt operation.

DSCUSERINTFUNCTION
Structure containing information about the user interrupt function and execution time.

DSCOPTOSTATE
Structure containing information on optoinput state

DSCPWM
Structure containing PWM parameters

DSCWATCHDOG
Structure containing watchdog settings

DSCQMMMMR
Structure containing configuration data for the Master Mode Register on the 9513 chip on GPIO-MM-11.

DSCQMMCMR
Structure containing configuration data for the Counter Mode Register of a counter on the 9513 chip on GPIO-MM-11.

DSCQMMMCC
Structure containing configuration data for Multiple Counter Control.

DSCQMMSCF
Structure containing configuration data for Special Counter Functions.

DSCQMMPWM
Structure containing configuration data for pulse width modulation function on GPIO-MM-11.

Recommended Usage of structures
It is recommended by DSC that any of the structures used should be initialized to 0 before using the same in the user code.

For example:

```
DSCADSETTINGS adsettings ;

memset ( &adsettings , 0 , sizeof ( DSCADSETTINGS ) ) ;
```

Data Type Macros
BYTE
unsigned char

SBYTE
signed char

WORD
unsigned short

SWORD
signed short

DWORD
unsigned long

SWORD
signed long

LONG
signed long

FLOAT
float

DFLOAT
double

BOOL
int

TRUE
(BOOL)1

FALSE
(BOOL)0

DSCSAMPLE
SWORD

DSCDACODE
DWORD

DSCB
SWORD

DSCUserInterruptFunction
Function datatype for user interrupts.

10. Board Reference

This section lists the boards that are currently supported in Universal Driver. The features and appropriate parameters for the various function calls are explained. The information presented here is only an overview of each board, intended to support software development using the Universal Driver. For complete functional and operating details on each board, please refer to that board's user manual. All of Diamond Systems' product user manuals (including this one) are available to download for free on our website at http://www.diamondsystems.com/support/techliterature. Note: Serial port operation is not supported in Universal Driver. Only the digital I/O and board configuration functions are supported on the serial port modules (Emerald-MM series).

10.1. Board Function Lists

Each board description contains a list of Universal Driver functions that work with that board. In addition to the functions listed for each board, all boards may also use the following universal functions:

Functions applicable to all boards:

- dscFree ()
- dscFreeBoard ()
- dscInit ()
- dscInitBoard ()
- DscRegisterRead()
- DscRegisterWrite ()
- dscSleep ()
- DscGetBoardMacro ()
- dscGetErrorString ()
- dscGetLastError ()
- DscSetSystemPriority ()

10.2. Analog IO Ranges

Several of the boards support programmable input ranges and resolution. For these boards, a table has been included detailing all possible A/D range settings. Here is a detailed description of the data in the tables:

Code — This is the code for the particular input range

Range: — This bit selects the board's A/D positive full-scale voltage range; 0 = 5V, 1 = 10V. Do not confuse this term with the board's "input range."

ADBU — This bit selects the board's A/D bipolar/unipolar setting; 0 = bipolar, 1 = unipolar.

G1, G0 — These bits select the board's A/D gain setting. The gain settings and their respective control bits are as follows:

G1	G0	Gain
0	0	1

0	1	2
1	0	4
1	1	8

Input Range	This column lists the input range resulting from the particular configuration of control bits.
Resolution	This column lists the voltage resolution resulting from the particular configuration of control bits. The resolution is equal to the smallest change in input voltage detectable by the A/D circuit. This smallest change results in a change in A/D code of 1 least significant bit, or LSB. The resolution is equal to the total voltage range (maximum input voltage - minimum input voltage) divided by the number of possible A/D codes (2n where n is the number of bits in the A/D converter). For an input range of ±5V and a 16-bit A/D converter, the resolution is (+5V - -5V) / 216 = 10 / 65536 = .000153V or 153µV.

10.3. Detailed Board Information

The list below provide basic information on each board required to use the various Universal Driver functions. For more complete descriptions of each board, refer to that board's user manual.

Athena
Athena-II
Diamond-MM
Diamond-MM-AT
Diamond-MM-16-AT
Diamond-MM-32-AT
Diamond-MM-32X-AT
Diamond-MM-32DX-AT
Diamond-MM-48-AT
Emerald-MM-8
Emerald-MM-DIO
GPIO-MM-11
GPIO-MM-21
Helios
Hercules-EBX
Hercules-II
IR104
Mercator
Neptune
Onyx-MM
Onyx-MM-DIO
Opal-MM
Pearl-MM
Poseidon
Ruby-MM
Ruby-MM-416
Ruby-MM-1612

11. Example Programs

Types of Example Programs

Diamond Systems has prepared a number of example programs to demonstrate the use of our products and software. These examples are all available from our website as well as the product CD-ROM which is delivered with most product shipments.

Demo Programs
Each board has its own set of example programs packaged as a zip file in the Demos folder of the product CD-ROM. These are also available on our website in the software section. These demos come with source code and build files for each supported operating system. Executable versionf of the demos are included for DOS as well. The demo programs are organized by the Universal Driver function they demostrate. You can run the program to see how the feature works, and then copy and paste the code into your own projects as as start to your own programming.

Compiling Demos on DOS
A project file for each demo is included for Borland C++ 5.0. You can launch this program file to compile the demo using Borland. The demos are already compiled for DOS so you don't need to do this unless you'd like to make changes to the demo program. You must install the Universal Driver on your system before compiling the demo programs.

Compiling Demos on Windows or WinCE 6.0
For compiling the demo programs on Windows CE 6.0, you would require the Visual Studio 2005 IDE from Microsoft along with the Platform Builder 6.0.

The Demo programs already have the required include file DSCUD.H included in the code. The project needs to statically link with the library file dscud_api.lib to create the application executable.

The driver DLL file DSCUD_API.DLL MUST be present in the final image of the platform to run the demo program. All the demo prgorams for Helios board are included in the runtime image provided by DSC.

Compiling Demos on Linux
A Makefile is included at the root of each demo package which can be used to compile all the demos for either Linux. Just type "make" at the base directory of the demos where you see the Makefile and it will compile all the demos for that board. You must have installed the Universal Driver before compiling. The driver must be installed at /usr/local/dscud_6.01_1 on Linuxfor the Makefile to find the files it requires.

Utility Programs
Utility programs with source code are available for checking or diagnosing various features on some boards.

Source Code Examples
We also have written example source code which demonstrates how to program Diamond Systems without using the Universal Driver. This is for customers who are using an operating system which is not supported, or for whatever reason are writing their own low level driver software instead of using the Universal Driver. These example programs use simple inp() and outp() type direct I/O calls to program the boards. To make use of the source code you'll need to port them to your own platform and find the local eqivalent of those functions before compiling.

12. Error Codes

Error Code Table

Here are all Universal Driver error codes along with a description of the error. Some additional error codes may be listed in dscud.h but they are not in use by the driver and are not documented here.

Name	Error Code	Description
DE_NONE	0	No errors reported. The operation succeeded.
DE_HW_FAILURE	1	Hardware failure reported. Check that you have specified the correct board type and board address. May indicate a problem with the board, or a hardware conflict at that board I/O address.
DE_SW_FAILURE	2	Software failure reported. This is a general purpose error. Check the error string for more detailed information about what went wrong.
DE_SW_NOT_SUPPORTED	4	Software does not support this operation. Check the board and driver documentation to see which features and driver functions are supported for that board.
DE_INVALID_PARM	5	A parameter to the function is invalid. Check the parameters you are passing to the driver function. Check the error string for more detailed information about the parameter that was rejected.
DE_ALTERNATE_IN_PROGRESS	6	Alternate interrupt operation in progress. You cannot have multiple interrupt operations of one type running at the same time. For example, if you call dscADScanInt () twice without calling dscCancelOp () between them, this error will be returned.
DE_NONE_IN_PROGRESS	7	No interrupt operation in progress to cancel. Will occur if dscCancelOp() is called while no interrupt operation is running.
DE_BUFFER_ROLLOVER	8	Pointer passed in + sizeof(data buffer) would roll over a data segment
DE_OVERFLOW	11	Am9513A counter function overflowed
DE_DSCUDH_INVALID	13	Header / library version mismatch. Check that the dscud.h header file you are including in your program is the same version of the Universal Driver as the library you are linking with. This version is checked using the version parameter passed to dscInit ().
DE_INVALID_BOARD	14	Invalid board type specified. Also returned if the DSCB parameter passed to a Universal Driver function is not a valid board identifier. Valid identifiers are obtained by calling dscInitBoard ().
DE_BOARD_LIMIT_REACHED	15	Tried to initialize too many boards. The maximum allowed is 10. Call dscFreeBoard () on boards which are not in use.
DE_INVALID_WINDRVR_HANDLE	17	WinDriver initialization failed. Windows only.
DE_INVALID_WINDRVR_VERSION	18	WinDriver version mismatch. Windows only.
DE_BAD_WINDRVR_BOARD_INIT	19	WinDriver could not initialize the board. Windows only.
DE_OPERATION_TIMED_OUT	20	Current operation timed out. May occur if a busy wait on a hardware status bit times out. May indicate a hardware problem or conflict.
DE_INVALID_WINDRVR_KP	21	WinDriver kernel plug-in initialization failed. Windows only.

13. Board Macros

Board Macro List

The following table lists all currently supported board macros and their corresponding board.

Macro	Code	Description
DSC_RMM	1	Ruby-MM
DSC_OPMM	3	Opal-MM
DSC_DMM	4	Diamond-MM
DSC_PMM	9	Pearl-MM
DSC_OMM	10	Onyx-MM
DSC_RMM416	11	Ruby-MM-416
DSC_DMM32	12	Diamond-MM-32-AT
DSC_EMMDIO	13	Emerald-MM-DIO
DSC_RMM1612	14	Ruby-MM-1612
DSC_DMMAT	15	Diamond-MM-AT
DSC_DMM16AT	16	Diamond-MM-16-AT
DSC_IR104	17	IR104
DSC_EMM8	18	Emerald-MM-8
DSC_HERCEBX	20	Hercules-II
DSC_DMM48	22	Diamond-MM-48-AT
DSC_OMMDIO	23	Onyx-MM-DIO
DSC_MRC	24	Mercator
DSC_DMM32XAT	27	Diamond-MM-32X-AT
DSC_GPIO11_9513	29	GPIO-MM-11
DSC_GPIO11_DIO	30	GPIO-MM-11
DSC_GPIO21	31	GPIO-MM-21
DSC_PSD	32	Poseidon
DSC_ATHENAII	33	Athena-II
DSC_DMM32DX	34	Diamond-MM-32DX-AT
DSC_HELIOS	35	Helios
DSC_NEPTUNE	36	Neptune
DSC_TEST	126	Virtual Test Board
DSC_RAW	127	Raw Board

Index

Athena

Overview

Athena's Data Acquistion is compatible with that of Prometheus. They have the same register map and code written for Prometheus will work just as well on Athena and vice versa. The only different software wise is watchdog timer; Athena has a different watchdog timer register map.

Analog Input

Max Input Channels: 16 (single-ended) or 8 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltrage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, or ±1.25V
Input Ranges (Unipolar): 0-8.3V, 0-5V, or 0-2.5V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate:
Software Command: approx. 20,000 samples per second, depending on code and operating system
Interrupt Routine w/FIFO: 100,000 samples per second
FIFO: 48 samples with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±10V	305µV
1	0	0	0	1	±5V	153µV
2	0	0	1	0	±2.5V	76µV
3	0	0	1	1	±1.25V	38µV
4	0	1	0	0	Invalid Setting	-
5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 8.3V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	Invalid Setting	-

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output

Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Bipolar Output Ranges: ±10V
Unipolar Output Ranges: 0-10V

Digital I/O
Max Ports: 3, bi-directional, programmable in 8-bit groups, TTL-compatible, similar to 82C55 Digital I/O ports on Prometheus require direction to be set with DscDIOSetConfig() before use.

All DIO lines power-up in input mode and have readback capability.

All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.
Athena Universal Driver Functions

- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- dscSetUserInterruptFunction()
- DscClearUserInterruptFunction()
- dscUserInt()
- dscWatchdogDisable()
- dscWatchdogEnable()
- dscWatchdogTrigger()

Athena-II

Overview

Athena-II's Data Acquistion is compatible with that of Athena-I. They have the same register map and code written for Athena-I or Prometheus will work just as well on Athena-II.

Athena-II in addition provides an enhanced features mode which makes an extended FIFO of 2048 samples available for storing AD data.

Board Initialization

To use the Athena-II board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_ATHENAII. This is shown in the example below...

The base address should be 0x280 and the default IRQ to use is IRQ 5.

dscInitBoard (DSC_ATHENAII , &dsccb, &board);

Analog Input

Max Input Channels: 16 (single-ended) or 8 (differential)

A/D Resolution: 16 bits (1/65536 of full-scale)

Data range: -32768 to +32767 for all voltrage ranges

Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, or ±1.25V

Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V or 0-1.25V

Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal

Maximum Conversion Rate:

Software Command: approx. 20,000 samples per second, depending on code and operating system

Interrupt Routine w/FIFO: 100,000 samples per second

FIFO: 48 samples with programmable threshold in standard mode. Upto 2048 in Enhanced FIFO mode. In the enhanced FIFO mode, it is recommended to set the FIFO threshold to 1024 samples by setting the FIFO threshold register Base + 5 with a value of 0x80.

To enable this, the EnhancedFeatures must be enabled using DscEnhancedFeaturesEnable API. The register Base + 12 must be set at 0x01 to turn the Enhanced FIFO ON.

The FIFO threshold register at location Base + 5 of the Athena-II board, contains the threshold for the enhanced mode. The threshold is an 11 bit number as the maximum FIFO depth available in Athena-II is 2048 samples. Thus to set the threshold value of 11 bit in an 8 bit register, the threshold value is programmed as 8 sample blocks. The universal driver takes care of the conversion.

For example, to set a FIFO threshold of 1024, the Universal Driver will write a value of 0x80 to the register.

This board supports the following programmable input ranges and resolutions:

A/D Ranges				
Polarity	G1	G0	Input Range	Resolution (1 LSB)
Bipolar	0	0	±10V	305µV
Bipolar	0	1	±5V	153µV
Bipolar	1	0	±2.5V	76µV
Bipolar	1	1	±1.25V	38µV
Unipolar	0	0	Invalid Mode	N/A
Unipolar	0	1	0 - 10V	153µV
Unipolar	1	0	0 - 5V	76µV
Unipolar	1	1	0 - 2.5V	38µV

Analog Output

Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Bipolar Output Ranges: ±10V
Unipolar Output Ranges: 0-10V

Digital I/O
Max Ports: 3, bi-directional, programmable in 8-bit groups, TTL-compatible, similar to 82C55 Digital I/O ports on Athena-II require direction to be set with DscDIOSetConfig() before use.

All DIO lines power-up in input mode and have readback capability.

Athena-II Universal Driver Functions

- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- dscSetUserInterruptFunction()
- DscClearUserInterruptFunction()
- dscUserInt()
- dscWatchdogDisable()
- dscWatchdogEnable()
- dscWatchdogTrigger()

BOOL
int.

BYTE
unsigned char

Diamond-MM

Diamond-MM is available in four models. Models DMM-NA and DMM-NA-XT do not have analog outputs, so the D/A information below does not apply to them. The remaining information applies to all four models.

Board Initialization

To use the DMM board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_DMM. This is shown in the example below...

dscInitBoard (DSC_DMM , &dsccb, &board);

Analog Input

Max Input Channels: 16 (single-ended) or 8 (differential)
A/D Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1V, ±0.5V, or Custom
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, 0-1V, or Custom
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate:
Software Command: Approx. 2,000-4,000 samples per second, depending on code and hardware platform
Interrupt Routine:
DOS: 20,000-40,000samples per second
Windows / Linux: 1,000-2,000 samples per second
FIFO: None

This board does not support programmable input ranges and resolutions. To change these settings, you must adjust the hardware jumpers on the board.

Analog Output (models DMM and DMM-XT only)

Max Output Channels: 2
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095
Output Ranges: 0-5V, custom (via potentiometer), or external reference; individual setting per channel

Digital I/O

Max Input Ports: 1 (1 byte or 8 bits), TTL compatible, no readback capability
Max Output Ports: 1 (1 byte or 8 bits), TTL compatible

Digital I/O lines on Diamond-MM are fixed direction and do not require configuration prior to use.

Diamond-MM Universal Driver Functions

- dscADSample ()
- dscADSampleInt ()
- dscADScan ()
- dscADScanInt ()
- dscADSetChannel ()
- DscADSetSettings ()
- DscClearUserInterruptFunction ()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- dscDAConvert() Models DMM and DMM-XT only
- dscDAConvertScan() Models DMM and DMM-XT only
- dscDAConvertScanInt() Models DMM and DMM-XT only
- dscDIOClearBit ()

- dscDIOInputBit ()
- dscDIOInputByte ()
- dscDIOOutputBit ()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscSetUserInterruptFunction()

Diamond-MM-AT

Board Initialization

To use the DMM-AT board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_DMM. This is shown in the example below...

dscInitBoard (DSC_DMM , &dsccb, &board);

Analog Input

Max Input Channels: 16 (single-ended) or 8 (differential)
A/D Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1V, ±0.5V, or Custom
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, 0-1V, or Custom
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): 2,000 (approx.) samples per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 100,000 samples per second
FIFO: 512 samples with fixed threshold of 256

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	2.44mV
1	0	0	0	1	±2.5V	1.22mV
2	0	0	1	0	±1.25V	0.61mV
3	0	0	1	1	±0.625V	0.305mV
4	0	1	0	0	0 - 10V	2.44mV
5	0	1	0	1	0 - 5V	1.22mV
6	0	1	1	0	0 - 2.5V	0.61mV
7	0	1	1	1	0 - 1.25V	0.305mV
8	1	0	0	0	±10V	4.88mV
9	1	0	0	1	±5V	2.44mV
10	1	0	1	0	±2.5V	1.22mV
11	1	0	1	1	±1.25V	0.61mV
12	1	1	0	0	Invalid Setting	-
13	1	1	0	1	Invalid Setting	-
14	1	1	1	0	Invalid Setting	-
15	1	1	1	1	Invalid Setting	-

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output

Max Output Channels: 2
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095
Output Ranges (Fixed): 0-5V
Output Ranges (Programmable): 0-1V to 0-10V

Digital I/O

Max Input Ports: 1 (1 byte or 8 bits), TTL compatible, no readback capability
Max Output Ports: 1 (1 byte or 8 bits), TTL compatible

Digital I/O lines on Diamond-MM-AT are fixed direction and do not require configuration prior to use.

Diamond-MM-AT Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- DscGetEEPROM()
- DscSetEEPROM()
- DscSetReferenceVoltages()
- DscGetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()

Diamond-MM-16-AT

Board Initialization

To use the DMM-16-AT board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_DMM16. This is shown in the example below...

dscInitBoard (DSC_DMM16 , &dsccb, &board);

Analog Input

Max Input Channels: 16 (single-ended) or 8 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, or ±1.25V
Input Ranges (Unipolar): 0-10V, 0-5V, or 0-2.5V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): 2,000 (approx.) samples per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 100,000 samples per second
FIFO: 512 samples with fixed threshold of 256

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	153µV
1	0	0	0	1	±2.5V	76µV
2	0	0	1	0	±1.25V	38µV
3	0	0	1	1	Invalid Setting	-
4	0	1	0	0	Invalid Setting	-
5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 10V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	Invalid Setting	-

NOTE: The A/D modes for codes 3 through 7 and 15 are invalid, and the settings for codes 9 through 11 are identical to codes 0 through 2.

Analog Output (model DMM-16-AT only)

Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Output Ranges (Bipolar, Fixed): ±5V
Output Ranges (Bipolar, Programmable): ±1V to ±10V
Output Ranges (Unipolar, Fixed): 0-5V
Output Ranges (Unipolar, Programmable): 0-1V to 0-10V

Digital I/O

Max Input Ports: 1 (1 byte or 8 bits), TTL compatible, no readback capability
Max Output Ports: 1 (1 byte or 8 bits), TTL compatible
Digital I/O lines on Diamond-MM-16-AT are fixed direction and do not require configuration prior to use.

Diamond-MM-16-AT Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()

Diamond-MM-32-AT

Board Initialization
To use the DMM-32-AT board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_DMM32. This is shown in the example below...
dscInitBoard (DSC_DMM32 , &dsccb, &board);

Analog Input
Max Input Channels: 32 (single-ended), 24 (16/8 single-ended/differential), or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1.25V, or ±0.625V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): system-dependent, up to 100,000 per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 200,000 samples per second
FIFO: 512 samples with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	153µV
1	0	0	0	1	±2.5V	76µV
2	0	0	1	0	±1.25V	38µV
3	0	0	1	1	±0.625V	19µV
4	0	1	0	0	Invalid Setting	-
5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 10V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	0 - 1.25V	19µV

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output

Max Output Channels: 4 D/A
Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Output Ranges (Bipolar): ±5V, ±10V, or programmable
Output Ranges (Unipolar): 0-5V, 0-10V, or programmable

Digital I/O

Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) similar to 82C55
Digital I/O ports on Diamond-MM-32-AT require direction to be set with dscDIOSetConfig() before use.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.

DSCUD API Notes
For these functions DMM32-AT has the following restrictions

dscADSampleInt, dscADScanInt
1. FIFO threshold (dscaioint.fifo_depth) must be a multiple of the number of channels
2. Num_conversions (dscaioint.num_conversions) must be a multiple of fifo threshold
3. FIFO threshold (dscaioint.fifo_depth) must be an even number between 0 to 510

Diamond-MM-32-AT Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()

Diamond-MM-32X-AT

Overview

DMM32X-AT is backwards compatible with DMM32X-AT. However the DMM32X-AT has many enhanced features that improve performance and make it more attractive for certain applications. Some of the features are a 1024 sample FIFO (2048 upon request), 250 KHz max sample rate, auto auto-calibration, D/A wave form generator, and programmable FPGA. DMM32X-AT also has the capability to receive commands via RS232.

Board Initialization

To use the DMM32X-AT board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_DMM32X. This is shown in the example below...

```
dscInitBoard ( DSC_DMM32X , &dsccb, &board );
```

Analog Input

Max Input Channels: 32 (single-ended), 24 (16/8 single-ended/differential), or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1.25V, or ±0.625V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): system-dependent, up to 100,000 per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 250,000 samples per second
FIFO: 1024 samples (2048 upon request) with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	153µV
1	0	0	0	1	±2.5V	76µV
2	0	0	1	0	±1.25V	38µV
3	0	0	1	1	±0.625V	19µV
4	0	1	0	0	Invalid Setting	-
5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 10V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	0 - 1.25V	19µV

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output

Max Output Channels: 4 D/A
Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges

Output Ranges (Bipolar): ±5V, ±10V, or programmable
Output Ranges (Unipolar): 0-5V, 0-10V, or programmable

Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) similar to 82C55
Digital I/O ports on Diamond-MM-32-AT require direction to be set with DscDIOSetConfig () before use.

All DIO lines power-up in input mode and have readback capability.

All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.

Universal Driver API Notes
For these functions DMM32X-AT has the following restrictions

DscADSampleInt, DscADScanInt,
1. FIFO threshold (dscaioint.fifo_depth) must be a multiple of the number of channels
2. Num_conversions (dscaioint.num_conversions) must be a multiple of fifo threshold
3. FIFO threshold (dscaioint.fifo_depth) must be an even number between 0 to 1022 (or 0 to 2046 for 2048 size FIFO)

Diamond-MM-32X-AT Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()

- dscSetUserInterruptFunction()
- dscUserInt()
- DscEnhancedFeaturesEnble()
- DscAACCommand()
- DscAACGetStatus()
- DscWGCommand()
- DscWGConfigSet()
- DscWGBufferSet()

Diamond-MM-32DX-AT

Overview

DMM32DX-AT is backwards compatible with DMM32X-AT. However the DMM32DX-AT has many enhanced features that improve performance and make it more attractive for certain applications. Some of the features are a 1024 sample FIFO (2048 upon request), 250 KHz max sample rate, auto auto-calibration, D/A wave form generator, and programmable FPGA. DMM32DX-AT also has the capability to receive commands via RS232.

Board Initialization

To use the DMM-32DX-AT board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_DMM32DX. This is shown in the example below...

Apart from the regular parameters filled in for the DSSCB structure, for DMM32DX, an additional member needs to be filled called DAC_Config.

dsccb.DAC_Config = 1 ; // for 16 bit DAC usage or 0 for 12 bit DAC usage.

This member will let the universal driver know the type of DAC usage intended by the user. If the user wants to use only 12 bit operation (even though the DAC chip installed is 16 bit), this parameter should be set to 0. If the user desires to use the extended mode 16 bit DAC functionality, this member should be set to 1.

dscInitBoard (DSC_DMM32DX , &dsccb, &board);

Analog Input

Max Input Channels: 32 (single-ended), 24 (16/8 single-ended/differential), or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1.25V, or ±0.625V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): system-dependent, up to 100,000 per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 250,000 samples per second
FIFO: 1024 samples (2048 upon request) with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	153µV
1	0	0	0	1	±2.5V	76µV
2	0	0	1	0	±1.25V	38µV
3	0	0	1	1	±0.625V	19µV
4	0	1	0	0	Invalid Setting	-
5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 10V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	0 - 1.25V	19µV

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output
Max Output Channels: 4 D/A
Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Output Ranges (Bipolar): ±2.5V, ±5V, ±10V, or programmable
Output Ranges (Unipolar): 0-5V, 0-10V, or programmable
Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) similar to 82C55
Digital I/O ports on Diamond-MM-32-AT require direction to be set with DscDIOSetConfig() before use.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.

Universal Driver API Notes
For these functions DMM32X-AT has the following restrictions

DscADSampleInt, DscADScanInt
1. FIFO threshold (dscaioint.fifo_depth) must be a multiple of the number of channels
2. Num_conversions (dscaioint.num_conversions) must be a multiple of fifo threshold
3. FIFO threshold (dscaioint.fifo_depth) must be an even number between 0 to 1022 (or 0 to 2046 for 2048 size FIFO)

Diamond-MM-32DX-AT Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()

- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()
- DscEnhancedFeaturesEnble()
- DscAACCommand()
- DscAACGetStatus()
- DscWGCommand()
- DscWGConfigSet()
- DscWGBufferSet()

Diamond-MM-48-AT

Analog Input
Max Input Channels: 16 (single-ended)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ?5V
Input Ranges (Unipolar): 0-5V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): system-dependent, up to 100,000 per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 200,000 samples per second
FIFO: 2048 samples with programmable threshold (1024 or 256)

Analog Output
Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Output Ranges (Bipolar): 0-4.096V

Digital I/O
Max Ports: 8 TTL-compatible DIO lines.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.

Diamond-MM-48-AT Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()

- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()

DFLOAT
double

Dsc9513CounterControl
Initiates a Load, Arm, Load & Arm, Disarm, Save, or Disarm & Save command on a group of counters on GPIO-MM-11 or GPIO-MM-12.
An Arm command causes the counter to start counting, and a Disarm command causes it to stop. The counter will not count until it receives an Arm command.

The Save command or Disarm and Save command are used to latch the current contents of the counter into the Hold register for reading. If just the Save command is executed, the counter will continue to count; this is equivalent to the Lap button on a stopwatch. After executing one of these commands, use the Dsc9513ReadHoldRegister () function to read the latched data.

The Load command or Load and Arm command are used to load the contents of the Load register into the counter before or while arming it. Before executing one of these commands, use the dscSetLoadRegister() function to load the Load register with the desired data.

The function returns the current counter status bytes in the two-byte array status, which primarily indicates the current state of the outputs of all 5 counters in each group. To just read the status bytes, set the action flags to QMM_ACTION_NONE.

Function Definition
BYTE Dsc9513CounterControl(DSCB board, DSCQMMMCC* dscqmm_mcc, BYTE* status)

Function Parameters

Name	Description
board	The handle of the board to operate on
dscqmm_mcc	Data structure for the function
status	The counter chip status byte:

Return Value
Error code or 0.
Status Bit Table

Bit	7	6	5	4	3	2	1	0
Name	CMP2	CMP1	OUT5	OUT4	OUT3	OUT2	OUT1	BPTR

CMPn -- Status of the comparators for counters 2 and 1
OUTn -- Logic state of the indicated counter output signal
BPTR -- Byte pointer internal to the 9513 chip; used by Universal Driver; not relevant to the user application

Dsc9513MeasureFrequency

This function uses counters 4 and 5 to measure the frequency of a TTL-level signal. The input signal can be connected to any of the nine sources listed below. In order for this function to run, Counter 4 output (pin 12) must be wired externally to Counter 5 gate (pin 15). Otherwise, the program will hang.

The output of Counter 4 is initially high, and the output of Counter 5 is initially low. Counter 4's output will remain high for the duration specified in interval. The frequency is measured by using Counter 5 to count the number of rising edges of the input signal which occur during this measured time interval, and the count is returned in pulses. Counter 5 is gated by the output of Counter 4, so it counts for a known period of time. If the measurement period is too long, Counter 5 will reach its maximum count of 65535, its output will toggle high, and the function will return an error code of DE_QMM_OVERFLOW. Thus the programmer has both a hardware and a software indication of overflow. In this case use the next smaller measurement interval and try again, continuing in this fashion until no error code is returned.
The entire procedure lasts exactly as long as the specified measurement interval. For the longest interval (10 seconds), the computer may appear to have hung; however, it is simply in a monitoring loop waiting for Counter 4's output to toggle low, and it will return to the calling program as soon as this event occurs.

The fastest measurable frequency is limited by the 9513 chip specifications to 7MHz; specifying a measurement interval of 1 ms for a 7MHz input frequency will yield a count of 7000, well within the 65535 count limit. The slowest measurable frequency depends on the allowable error tolerance, since the edge count could be off by 1 due to alignment of the input edges with the measurement interval.

Measuring the frequency of a very slow signal is more accurately done by measuring the period with Dsc9513MeasurePeriod() and taking the reciprocal.
The frequency of the input signal is derived from the formulas
Frequency = pulses / Measurement period or Frequency = pulses * Measurement frequency
The function will return DE_QMM_OVERFLOW if a measurement overflow occurred. Otherwise it will return DE_NONE .

Function Definition
BYTE Dsc9513MeasureFrequency(DSCB board, BYTE interval, BYTE source, WORD* pulses)

Function Parameters

Name	Description
board	The handle of the board to operate on
interval	Desired measurement time interval. See the table below.
source	Source signal to measure. See the table below Note that Gate 5 is not a valid choice since it is used by the function.
pulses	Pointer to a variable that will contain the number of pulses

Return Value
Error code or 0.

QMM Interval Table

Macro
QMM_INTERVAL_1MS_1KHZ
QMM_INTERVAL_10MS_100HZ
QMM_INTERVAL_100MS_10HZ
QMM_INTERVAL_1S_1HZ
QMM_INTERVAL_10S_01HZ

QMM Source Table

Macro
QMM_SOURCE_SRC1
QMM_SOURCE_SRC2
QMM_SOURCE_SRC3
QMM_SOURCE_SRC4
QMM_SOURCE_SRC5
QMM_SOURCE_GATE1
QMM_SOURCE_GATE2
QMM_SOURCE_GATE3
QMM_SOURCE_GATE4

Dsc9513MeasurePeriod

This function measures the period of an input signal by counting the number of pulses of a known reference frequency which occur between two successive rising edges of the input signal. The input signal is connected to Gate 5 (pin 15), and the counting frequency is selected by frequency. The accuracy of the measurement is equal to the period of the chosen reference frequency.

The function operates as follows: Counter 5's output is preset low. On the first rising edge of the input signal, Counter 5 will begin to count edges of the reference frequency, and on the second rising edge its count will be transferred into its hold register. The contents of the hold register are returned in the parameter periods and represents the number of edges, or periods, of the reference frequency equal to a single period of the input signal. Note that since only the rising edge of the input signal is used to control Counter 5, the input signal's duty cycle is irrelevant. Also note that the function waits for the first rising edge before beginning the measurement. This adds to the total execution time for the function.

If the signal's period is too long, Counter 5 will reach its maximum count of 65535 and its output will toggle high. In this case the function will return error code DE_QMM_OVERFLOW, and periods will be set to 0. To correct this situation, select the next slower count frequency and repeat the procedure in this fashion until no error code is returned.
The longest period which can be measured using this function is 65,535 x the period of the chosen reference frequency. In the case of 400Hz, the slowest option, the longest period is 163,837ms, or 2 mins. 44 secs. Since the fastest reference frequency is 4MHz, the smallest period which can be captured is greater than the period of this frequency, or 0.25µs. However the measurement error will increase substantially as the period of the input signal approaches this lower limit.

Measuring the period of a fast input signal is more accurately done by measuring the frequency instead with Dsc9513MeasureFrequency() and taking the reciprocal.

The period of the input signal is calculated by the formula: Period = periods * reference period

The function will return DE_QMM_OVERFLOW if a measurement overflow occurred. Otherwise it will return DE_NONE .

Function Definition
BYTE dscQMMMeasurePeriod(DSCB board, BYTE frequency, DWORD* periods);

Function Parameters

Name	Description
board	The handle of the board to operate on
frequency	The desired reference interval of frequency to use for the measurement. See the table below.
periods	Pointer to a variable to hold the result of the measurement

Return Value
Error code or 0.

QMM Interval Table

Macro
QMM_INTERVAL_1MS_1KHZ
QMM_INTERVAL_10MS_100HZ
QMM_INTERVAL_100MS_10HZ
QMM_INTERVAL_1S_1HZ
QMM_INTERVAL_10S_01HZ

Dsc9513PulseWidthModulation

This function generates an output signal on the selected counter of GPIO-MM-11 or GPIO-MM-12 to the selected frequency with the selected duty cycle. The duty cycle is defined as the percent of time that the output is high. A 50% duty cycle represents a square wave (equal high and low periods), while a duty cycle of 0% means always low and a duty cycle of 100% means always high. The range for duty cycle is 0 to 99.99 in steps of .01. a duty cycle of 100% is not possible with this function.

The function takes as inputs the desired counter, output frequency, and duty cycle. It returns status information indicating the selected input frequency, the resulting load and hold register values, and a flag indicating whether the function has reached either duty cycle limit.

Function Definition
BYTE dsc9513PulseWidthModulation(DSCB board, DSCQMMPWM* pwm);

Function Parameters

Name	Description
board	The handle of the board to operate on
pwm	Structure containing configuration data for the function; see definition DSCQMMPWM.

Return Value
Error code or 0.

Dsc9513Reset

Reset 9513 registers on the board.

Function Definition
BYTE dsc9513Reset(DSCB board)

Function Parameters

Name	Description
board	The handle of the board to operate on

Return Value
0.

Dsc9513ReadHoldRegister

This function returns the 16-bit data in the selected counter's hold register. The counter data must first be latched by a Save or Save and Disarm command using Dsc9513CounterControl() or Dsc9513SingleCounterControl().

Function Definition
BYTE Dsc9513ReadHoldRegister(DSCB board, BYTE counter, WORD* value)

Function Parameters

Name	Description
board	The handle of the board to operate on
counter	Counter no., 1-10 for GPIO-MM-11 and GPIO-MM-21
value	16-bit value from the counter's hold register, range 0-65535

Return Value
Error code or 0.

Dsc9513SetCMR

Configures the Counter Mode Register for a counter on GPIO-MM-11 or GPIO-MM-12. The Counter Mode Register for a counter must be configured before that counter can be used.

Function Definition
BYTE dsc9513SetCMR(DSCB board, DSCQMMCMR* dscqmm_cmr)

Function Parameters

Name	Description
board	The handle of the board to operate on
dscqmm_cmr	Structure containing configuration data for the Counter Mode Register; see page 180.

Return Value
Error code or 0.

Dsc9513SetHoldRegister

This function is identical to Dsc9513SetLoadRegister() except the data is loaded into the selected counter's Hold register instead of the Load register. In some operating modes of the 9513 counter chip, the counter alternates between the Load register and the Hold register when reloading. This feature can be used to generate square waves with variable duty cycles (PWM signals) by loading different values in the Load and Hold registers. See the descriptions of the counter modes in the 9513 datasheet for more detail on the uses of this register.

Function Definition
BYTE dsc9513SetHoldRegister(DSCB board, BYTE counter, WORD value)

Function Parameters

Name	Description
board	The handle of the board to operate on
counter	Counter no., 1-10 for GPIO-MM-11 and GPIO-MM-12
value	16-bit load value, range 0-65535

Dsc9513SetLoadRegister

This function loads the supplied data into the specified counter's Load register. The Load register is used to specify a counter's initial value as well as to set the reload value when the counter reaches its terminal count during repetitive counting modes. The data is not actually loaded into the counter's count register until a Load or Load and Arm command is issued with Dsc9513CounterControl().

Function Definition
BYTE dscQMMSetLoadRegister (DSCB board, BYTE counter, WORD value)

Function Parameters

Name	Description
board	The handle of the board to operate on
counter	Counter no., 1-10 for GPIO-MM-11
value	16-bit load value, range 0-65535

Return Value
Error code or 0.

Dsc9513SetMMR

Sets the configuration for the Master Mode Register on the 9513 counter/timer chip on GPIO-MM-11 or GPIO-MM12. The Master Mode Register must be configured before any counter on the chip can be used. See the definition of the data type DSCQMMMMR for information on the parameters required. Each 9513 has its own Master Mode Register, i.e. on GPIO-11 with 2 chips, the function must be called once for each chip.

Function Definition
BYTE dsc9513SetMMR (DSCB board, DSCQMMMMR* dscqmm_mmr)

Function Parameters

Name	Description
board	The handle of the board to operate on
dscqmm_mmr	Structure containing configuration data for the Master Mode Register

Return Value
Error code or 0.

Dsc9513SingleCounterControl

Initiates a Load, Arm, Load & Arm, Disarm, Save, or Disarm & Save command on a single counter on GPIO-MM-11 or GPIO-MM-12. See information on the commands in Dsc9513CounterControl ().

Function Definition
BYTE Dsc9513SingleCounterControl(DSCB board, BYTE counter, BYTE action)

Function Parameters

Name	Description
board	The handle of the board to operate on
counter	Counter no., 1-10 for GPIO-MM-11 and GPIO-MM-12
action	Action to perform on the selected counter. Select from choices below: QMM_ACTION_NONE 0 QMM_ACTION_ARM 1 QMM_ACTION_LOAD 2 QMM_ACTION_LOAD_AND_ARM 3

Return Value
Error code or 0.

Dsc9513SpecialCounterFunction

This function is used to program the alarm registers for counters 1 and 2 of each 9513 chip, to set the initial output state of any counter, or to step (increment / decrement) any counter. The most common use is to set the initial output state of a counter operating in toggle mode.

Function Definition
BYTE dsc9513SpecialCounterFunction(DSCB board, DSCQMMSCF* dscqmm_scf)

Function Parameters

Name	Description
board	The handle of the board to operate on
dscqmm_scf	Structure containing data required for the special counter function.

Return Value
Error code or 0.

DscAACCommand

Sets the commands for auto auto-calibration. These commands can be used to enable, disable, trigger, and reset the auto auto-calibration

Function Definition

```
#define AAC_CMD_HOLD  0x10
#define AAC_CMD_REL   0x08
#define AAC_CMD_RESET 0x04
#define AAC_CMD_ABORT 0x02
#define AAC_CMD_TRIG  0x01  DWORD
```

BYTE dscAACCommand(DSCB board, DWORD cmd)

Function Parameters

Name	Description
board	The handle of the board to operate on
cmd	Command used to enable, disable, trigger, and reset auto auto-calibration. Masks are AAC_CMD_XXX

Return Value
Error code or 0.

DscAACGetStatus

Gets the status of the auto auto-calibration operation and registers

Function Definition
BYTE dscAACGetStatus(DSCB board, DSCAACSTATUS* status)

Function Parameters

Name	Description
board	The handle of the board to operate on
status	Status of the AAC operation and registers

Return Value
Error code or 0.

DSCAACSTATUS

Structure containing auto auto-calibration parameters for function DscAACGetStatus().

Structure Definition

```
typedef struct {
  BOOL pic_present;
  BOOL pic_busy;
  BOOL aac_hold;
  BOOL aac_error;
  BOOL aac_active;
} DSCAACSTATUS;
```

Structure Members

Name	Description
pic_present	TRUE if PIC device is present, FALSE otherwise
pic_busy	TRUE if PIC device is busy with an operation, FALSE otherwise
aac_hold	TRUE if AAC Hold-Off is enabled, FALSE otherwise
aac_error	TRUE if the last AAC command has failed, FALSE otherwise
aac_active	TRUE if the AAC routing is currently running, FALSE otherwise

dscADAutoCal

Performs an A/D auto-calibration on a selected A/D range or on all A/D ranges. Valid settings for params->adrange are 0-15 for an individual range or 255 for all ranges. See the board-specific information in Chapter 10 for descriptions of the valid ranges for each A/D board. This function incurs a delay of 0.5 - 2 seconds per A/D range to perform the auto-calibration process.

Function Definition
BYTE dscADAutoCal(DSCB board, DSCADCALPARAMS* params)

Function Parameters

Name	Description
board	The handle of the board to operate on

params	The A/D calibration settings to be used in the auto-calibration process. See the definition of DSCADCALPARAMS.

Return Value
Error code or 0.

DSCADCALPARAMS

Structure containing A/D auto-calibration settings. Used by dscADAutocal().

Structure Definition

```
typedef struct {
   BYTE adrange;
   BYTE boot_adrange;
   FLOAT ad_offset;
   FLOAT ad_gain;

} DSCADCALPARAMS;
```

Structure Members

Name	Description
adrange	A/D range to calibrate; if adrange = 255, then all A/D ranges will be calibrated
boot_adrange	A/D range whose calibration settings will be loaded from the EEPROM when the board is first powered on.
ad_offset	The maximum offset error in A/D LSBs resulting from the autocalibration process
ad_gain	The maximum gain error in A/D LSBs resulting from the autocalibration process

dscADCalVerify

Verifies the accuracy of the most recent A/D autocalibration process and returns the maximum measured errors. If params->adrange is not set to a valid A/D range code, the function returns the error code DE_INVALID_PARM. The calibration verification process lasts approx. 2-5 seconds.

Function Definition
BYTE dscADCalVerify(DSCB board, DSCADCALPARAMS* params)

Function Parameters

Name	Description
board	The handle of the board to operate on
params	The A/D calibration settings to be used in the verification process

Return Value
Error code or 0.

DscADCodeToVoltage

Utility function for converting AD units to voltage based on the AD settings used.

Function Definition
BYTE DscADCodeToVoltage(DSCB board, DSCADSETTINGS adsettings, DSCSAMPLE adcode, DFLOAT *voltage);

Function Parameters

Name	Description
Board	The board handle
Adsettings	AD settings used when the sample was taken
Adcode	AD code to be converted
Voltage	Pointer to the variable which will hold the converted voltage

Return Value
Error code or 0.

Usage Example

```
ERRPARAMS errparams;
DSCADSETTINGS adsettings;
DSCB dscb;
DSCSAMPLE sample;
DFLOAT voltage;
BYTE result;

/* Board initialization code omitted */

if ((result = dscADSample(dscb, &sample)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscADSample failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}

if ((result = dscADCodeToVoltage(dscb, adsettings, sample, &voltage)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscADCodeToVoltage failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}

printf("Sample readout: %hd, Actual voltage: %5.3lfV\n", sample, voltage);
```

dscADSample

Performs a single A/D conversion on the currently selected channel. The function first waits for the board to be ready for a conversion. It then starts the conversion and waits for it to finish before reading data from the board. A built-in timer is used to check for board failure. If the timer times out before the conversion completes, the board returns the error code

Function Definition
BYTE dscADSample(DSCB board, DSCSAMPLE* sample);

Function Parameters

Name	Description
board	The handle of the board to operate on

sample	The sample resulting from the A/D conversion; the range of return values depends on the type of A/D board.

Return Value
Error code or 0.

dscADSampleAvg

Performs count A/D conversions on the currently selected channel and returns the average. The function first waits for the board to be ready for a conversion. It then starts the conversion and waits for it to finish before reading data from the board. A built-in timer is used to check for board failure. If the timer times out before the conversion completes, the board returns the error code

Function Definition
BYTE dscADSampleAvg (DSCB board, DFLOAT* sample, int count);

Function Parameters

Name	Description
board	The handle of the board to operate on
sample	The average sample resulting from the A/D conversions; the range of return values depends on the type of A/D board.
count	The number of samples to taken when computing the average

Return Value
Error code or 0.

dscADSampleInt

Performs A/D conversions using interrupt-based I/O with one A/D conversion per A/D clock tick. Note that calling this function only starts the interrupt operations in a separate system thread. The function call does not result in an atomic transaction with immediate results. Rather, it starts a real-time process that terminates only when the number of conversions reaches the maximum specified (one-shot mode) or if a call to dscCancelOp () is made.

Function Definition
BYTE dscADSampleInt(DSCB board, DSCAIOINT*dscaioint);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscaioint	The interrupt-based analog I/O settings to be used

Return Value
Error code or 0.

A/D samples resulting from the interrupt operation are stored in the buffer whose pointer is passed to the function in the DSCAIOINT structure.

DscADScanAvg

Performs count A/D conversions on each channel in the selected channel range. The channels are sampled individually in rapid sequence. The channels are not sampled simultaneously. The samples for each channel are averaged and returned. The high channel number must be greater than or equal to the low channel number, and both values must be

less than the total number of channels on the board. Setting high = low is the same as using dscADSample(). The function returns DE_OPERATION_TIMED_OUT if a timeout occurs because the A/D circuit is not ready (A/D Busy signal stays true). This usually indicates a hardware error.

Function Definition
BYTE dscADScanAvg (DSCB board, DSCADSCAN* dscadscan, DFLOAT* sample_values, int count);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscadscan	The A/D scan settings to be used
sample_values	Pointer to an array or allocated memory for holding the average scan values.
count	The number of conversions to do for each channel when computing the average

Return Value
Error code or 0.

DSCADSCAN

Structure Definition

```
typedef struct {
   BYTE low_channel;
   BYTE high_channel;
   DSCSAMPLE* sample_values;
   BYTE gain;
} DSCADSCAN;
```

Structure Members

Name	Description
low_channel	The starting channel in the scan range on which to perform A/D conversions
high_channel	The ending channel in the scan range on which to perform A/D conversions; must be greater than or equal to low_channel.
sample_values	Pointer to a buffer to hold the results of the A/D conversions
gain	Gain setting for A/D conversions; valid settings are GAIN_1, GAIN_2, GAIN_4, or GAIN_8

dscADScan

Performs a single A/D conversion on each channel in the selected channel range. The channels are sampled individually in rapid sequence. The channels are not sampled simultaneously. The high channel number must be greater than or equal to the low channel number, and both values must be less than the total number of channels on the board. Setting high = low is the same as using DscADSample. The function returns DE_OPERATION_TIMED_OUT if a timeout occurs because the A/D circuit is not ready (A/D Busy signal stays true). This usually indicates a hardware error.

Function Definition
BYTE dscADScan(DSCB board, DSCADSCAN* dscadscan, DSCSAMPLE* sample_values);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscadscan	The A/D scan settings to be used

sample_values	Pointer to an array or allocated memory for holding the scan values.

Return Value
Error code or 0.

dscADScanInt

Performs A/D scans using interrupt-based I/O with one scan per A/D clock tick. Note that calling this function only starts the interrupt operations in a separate system thread. The function call does not result in an atomic transaction with immediate results. Rather, it starts a real-time process that terminates only when the number of conversions reaches the maximum specified (one-shot mode) or if a call to dscCancelOp() is made.

Function Definition
BYTE dscADScanInt(DSCB board, DSCAIOINT* dscaioint);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscaioint	The interrupt-based analog I/O settings to be used

Return Value
Error code or 0.

dscADSetChannel

Sets the input channel range for future A/D conversions. A slight delay of approximately 10 microseconds occurs during this function call in order to allow the analog circuit on the board to settle on the new settings.

Function Definition
BYTE dscADSetChannel(DSCB board, BYTE low_channel, BYTE high_channel);
Function Parameters

Name	Description
board	The handle of the board to operate on
low_channel	The low channel in the desired range of channels
high_channel	The high channel in the desired range of channels

Return Value
Error code or 0.

DSCADSETTINGS

Structure Definition

```
typedef struct {
   BYTE current_channel;
   BYTE gain;
   BYTE range;
   BYTE polarity;
   BYTE load_cal;
   BYTE scan_interval;
   BYTE addiff;
} DSCADSETTINGS;
```

Structure Members

Name	Description
current_channel	The channel on which to perform A/D conversions
gain	The gain setting to use for A/D conversions; valid settings are GAIN_1, GAIN_2, GAIN_4, and GAIN_8
range	The range setting for A/D conversions; this is not to be confused with the term "A/D range" or "input range". Valid settings are RANGE_5 or RANGE_10
polarity	The polarity setting to use for A/D conversions; valid settings are BIPOLAR or UNIPOLAR
load_cal	If TRUE, the board will retrieve the calibration settings for the selected A/D range from the EEPROM and store it in the calibration circuit before performing the A/D conversion. This results in higher accuracy but causes a slight delay of a few milliseconds. If FALSE, the board will use the current calibration settings and start the A/D conversion immediately. Once a particular A/D range's calibration settings are loaded into the board, they will remain loaded until the board powers off or a new set of settings is loaded. When the board powers up, it automatically loads the settings defined as the boot mode. The boot mode is set with the autocal function.
scan_interval	Determines the time delay between consecutive A/D conversions during an A/D scan operation. When you perform an A/D scan, the individual A/D samples will be spaced apart by exactly this amount of time. The time delay affects the maximum possible sample rate. For example, if you want to achieve a 100KHz sample rate, the maximum scan interval is 1/100KHz = 10 microseconds. 0 20 microseconds (default value) 1 15 microseconds 2 10 microseconds 3 5 microseconds 4 9 microseconds 5 4 microseconds
addif	For Hercules and Elektra only, 0 = SINGLE_ENDED or 1 = DIFFERENTIAL

DscADSetSettings

Sets the A/D configuration for future A/D conversions. A delay of approximately 10 microseconds occurs during this function call in order to allow the analog circuit on the board to settle on the new settings. On -AT (auto-calibrating) boards, if adsettings->load_cal is TRUE, then load the A/D calibration settings from the EEPROM for the current A/D range. This results in slightly more accurate A/D conversions but causes an additional, longer delay of approximately 10 milliseconds.

Function Definition

BYTE dscADSetSettings (DSCB board, DSCADSETTINGS* adsettings);

Function Parameters

Name	Description
board	The handle of the board to operate on
adsettings	The A/D conversion settings to be used in subsequent A/D conversions; these settings will be used for all A/D conversions until they are changed with dscADSetChannel() or a call to an A/D interrupt function. See DSCADSETTINGS.

Return Value
Error code or 0.

DSCAIOINT

Structure containing interrupt-based analog I/O settings. These settings determine the behavior of the interrupt-based A/D sampling operation. Please refer to the A/D interrupt description for complete explanation of interrupt behavior resulting from the various settings in this structure.

Structure Definition

```
typedef struct {
   DWORD num_conversions;
   FLOAT conversion_rate;
   BOOL cycle;
   BOOL internal_clock;
   BYTE low_channel;
   BYTE high_channel;
   BOOL external_gate_enable;
   BOOL internal_clock_gate;
   DSCSAMPLE* sample_values;
   BOOL fifo_enab;
   WORD fifo_depth;
   DWORD dump_threshold;
   BYTE clksource;
} DSCAIOINT;
```

Name	Description
num_conversions	In one-shot mode (cycle = 0), this is the maximum number of A/D or D/A conversions to perform. In recycle mode, this is the buffer size.
conversion_rate	The desired rate at which single conversions (sample mode) or scans (scan mode) should occur. In most cases the input clock is 10MHz. This signal is divided by an integer value to obtain the desired sample rate. Because of the integer arithmetic, not all sample rates are obtainable. The driver will select the closest possible rate.
cycle	If TRUE, the function will perform A/D or D/A conversions repeatedly. When the current number of transfers reaches the number set in num_conversions, The process will continue. The buffer pointer will be reset to the beginning of the buffer and new data will overwrite old data. If FALSE, the function will take the number of samples specified by num_conversions and then terminate.
internal_clock	TRUE = use the board's internal clock to generate interrupts. FALSE = use the external clock (source pin depends on the board).
low_channel	The starting channel of the channel range on which to perform A/D or D/A conversions.
high_channel	The ending channel of the channel range on which to perform A/D conversions.
external_gate_enable	This parameter defines the control bit C2 on boards DMM, DMM-16, DMM-AT, and DMM-16-AT. It is used to enable an external signal on pin 29 (In0-) of the boards I/O connector, which provides a gating control for A/D sampling when an external clock is used to control the sample rate. If enabled, a high level on In0- will enable the sampling to run, and a low level will halt sampling. TRUE = use the external signal IN0- to gate the external sampling clock. FALSE = No gating is enabled and sampling continues uninterrupted. See sections regarding C2, In0-, and DIO in the board-specific manuals for more details.
internal_clock_gate	This parameter defines the control bit C0 on boards DMM, DMM-16, DMM-AT, and DMM-16-AT. It is used to enable an external signal on pin 48 (Din0) of the boards' I/O connector to provide a gating control for A/D sampling when the internal clock is used to control the sample rate. If enabled, a high level on Din0 will enable the sampling to run, and a low level will halt sampling. On the DMM-32-AT this will set GT12EN high. In general: TRUE = Use an external signal to gate the internal sampling clock, while FALSE = No gating is enabled and sampling continues uninterrupted.
sample_values	Pointer to a buffer holding either the results of the A/D conversions or the output codes to use in D/A conversions.
fifo_enab	If TRUE, will use the on-board FIFO in analog input operations. Used only on DMM-AT, DMM-16-AT, and DMM-32.
fifo_depth	The number of A/D samples to store in the FIFO before generating an interrupt request.
dump_threshold	The number of A/D conversions to perform before copying the driver's sample buffer to the user-accessible buffer.
clksource	(Prometheus only) Selects the clock source for counter/timer 1 during D/A conversions. 1 = 10MHz, 0 = 100KHz.

DSCAUTOCAL

Legacy reference to maintain backwards-compatibility; identical to DSCADCALPARAMS.

dscCancelOp

Terminates all currently running interrupt operations on the board. If there is no current interrupt operation on the board (OP_TYPE_NONE), the function will return DE_NONE_IN_PROGRESS.

Function Definition
BYTE dscCancelOp (DSCB board)

Function Parameters

Name	Description
board	The handle of the board to operate on

Return Value
Error code or 0.

DscCancelOpType

Terminates a currently running interrupt operation on the board based on int_type. If there is no current interrupt operation on the board matching the interrupt type, the function will return DE_NONE_IN_PROGRESS.

Function Definition
BYTE dscCancelOpType(DSCB board, DWORD int_type);

Function Parameters

Name	Description
board	The handle of the board to operate on
int_type	The interrupt type (one of INT_TYPE_*) to cancel

Return Value
Error code or 0.

DSCB

SWORD

DSCCB

Structure containing hardware settings for the current board. Some elements are unique to particular boards.

Structure Definition

```
typedef struct {

   BYTE boardtype;
   DSCB boardnum;
   WORD base_address;
   BYTE int_level;
   BOOL RMM_external_trigger;
   BOOL RMM_external_trigger_c3;
   WORD EMM_IOAddr[8];
   WORD EMM_Interrupt[8];
   BYTE clkfrq0;
   BYTE clkfrq1;
   BYTE clksel1;
```

```
   WORD address_space;
   BYTE DAC_Config; // ONLY USED FOR DMM32DX
} DSCCB;
```

Structure Members

Name	Description
boardtype	The board type constant; automatically filled by dscInitBoard;
boardnum	The handle to the board; automatically filled in by dscInitBoard
base_address	Base address of the board; refer to the board's user manual for valid base address settings
int_level	Interrupt level of the board; used for boards with only one IRQ.
RMM_external_trigger	Enable or disable the external trigger; Used only on RMM416 and RMM1612
RMM_external_trigger_c3	Enable or disable the external trigger; RMM416- and RMM1612-specific
EMM_IOAddr[8]	I/O addresses for up to eight ports; EMM8-specific
EMM_Interrupt[8]	Interrupts for up to eight ports; EMM8-specific
clkfrq0	Frequency for on-board counter 0; used only on Prometheus; 0 = 10 MHz, 1= 1 MHz;
clkfrq1	Frequency for on-board counter 1; used only on Prometheus; 0 = 10 MHz, 1= 100KHz;
clksel1	0 = internal oscillator with frequency set by clkfrq1, 1= external clock input CLK1; used only on Prometheus
address_space	Size of I/O block in bytes to allocate. Only needed for DSC_RAW board type.
DAC_Config	Resolution of the DAC to use in the code. If this flag is set to 0, the driver will always use the DAC as a 12 bit DAC - This helps for backward compatibility. When set to 1, 16 bit DAC will be used. This is only used by the DMM32DX board.For all other boards this field is ignored

DSCCR

Structure containing information on all counters. This is used for functions operating on 82C54 counters (all boards except Quartz-MM).

Structure Definition

```
typedef struct {
   BYTE control_code;
   BYTE counter_number;
   DWORD counter_data;
   DSCCS counter0;
   DSCCS counter1;
   DSCCS counter2;
} DSCCR;
```

Structure Members

Name	Description
control_code	Control code to write to or read from the control word register
counter_number	Selected counter, 0-2
counter_data	Counter divisor value, 0-65535

counter0	Status and data read from counter 0
counter1	Status and data read from counter 1
counter2	Status and data read from counter 2

DSCCS

Structure containing individual counter information. This is used for functions operating on 82C54 counters (all boards except Quartz-MM).

Structure Definition

```
typedef struct {
   DWORD value;
   BYTE status;
} DSCCS;
```

Structure Members

Name	Description
value	The counter read-back value
status	Counter read-back status

DscClearUserInterruptFunction

Uninstalls the user interrupt function for all interrupt types.

Function Definition

```
BYTE dscClearUserInterruptFunction (DSCB board);
```

Function Parameters

Name	Description
board	The handle of the board to operate on

Return Value

Error code or 0.

DscClearUserInterruptFunctionType

Uninstalls a user interrupt function from a specific interrupt type.

Function Definition

```
BYTE dscClearUserInterruptFunctionType(DSCB board, DWORD int_type);
```

Function Parameters

Name	Description
board	The handle of the board to operate on
int_type	The interrupt type (one of INT_TYPE_*) to clear the function from

Return Value

Error code or 0.

dscCounterDirectSet

Sets the configuration for an individual counter on an 82C54 chip. This chip is used on all boards with counter/timers except Quartz-MM and Prometheus. This function provides direct access to the 82C54 configuration register, enabling functionality other than simple rate generator. For information on the possible configuration data, see the 82C54 datasheet included with each board's user manual.

Function Definition
BYTE dscCounterDirectSet (DSCB board, BYTE code, WORD data, BYTE ctr_number);

Function Parameters

Name	Description
board	The handle of the board to operate on
code	The configuration code to write to the 82C54 chip; see the 82C54 datasheet at the back of the board's user manual for configuration code information
data	The initial count to write to the selected counter; range is 0-65535
ctr_number	The individual counter to set; range is 0-2

Return Value
Error code or 0.

DscCounterRead

Reads the data from an 82C54 counter.

Function Definition
BYTE dscCounterRead(DSCB board, DSCCR* dsccr)

Function Parameters

Name	Description
board	The handle of the board to operate on
dsccr	The settings for all counters in the structure dsccr; see DSCCR definition

Return Value
Error code or 0.

DscCounterSetRate

Sets the output frequency of the counter/timers dedicated to A/D sample rate timing. On all Diamond-MM boards, counters 1 and 2 are used together (2 x 16 bits) for A/D timing, and this function will program both counters such that the output of counter 2 is as close as possible to the selected rate. On Prometheus, counter 0 (24 bits) is used for A/D timing.

Note that not all rates are possible with complete precision. Each rate is set by programming the counter(s) with the divisor(s) resulting from the calculation (Input Clock) / Rate. On Diamond-MM boards, Input Clock is usually 10MHz. In each case a best fit is determined. However because of the integer math used, any errors resulting from truncation of the divisor to an integer value will cause the actual rate to differ from the desired rate. If rate < 1.0 then the function returns DE_INVALID_PARM.

This function works on all boards with 82C54 counter/timer chips. Quartz-MM uses a different counter/timer chip and has separate functions for programming.

Function Definition
BYTE dscCounterSetRate (DSCB board, float rate);

Function Parameters

Name	Description
board	The handle of the board to operate on
rate	Desired rate for the A/D counter/timer circuit

Return Value
Error code or 0.

DscCounterSetRateSingle

This function is similar to DscCounterSetRate() except that it operates only on the specified counter or counter combination (instead of all onboard counters.)

Function Definition

```
#define COUNTER_0     0x01
#define COUNTER_1     0x02
#define COUNTER_2     0x04
#define COUNTER_0_1 0x08
#define COUNTER_1_2 0x10
#define COUNTER_0_1_2         0x20
```

BYTE dscCounterSetRateSingle(DSCB board, float rate, DWORD ctr);

Function Parameters

Name	Description
board	The handle of the board to operate on
rate	Desired rate for the A/D counter/timer circuit
ctr	The desired counter or counter combination to program.

Return Value
Error code or 0.

dscDAAutoCal

Performs a D/A auto-calibration. This function incurs a delay of 2-5 seconds in order to perform the D/A auto-calibration process.

Function Definition
BYTE dscDAAutoCal(DSCB board, DSCDACALPARAMS* params);

Function Parameters

Name	Description
board	The handle of the board to operate on
params	The AD/A calibration settings to be used in the auto-calibration process. See the definition of DSCADCALPARAMS.

Return Value
Error code or 0.

DSCDACALPARAMS

Structure containing parameters used in D/A auto-calibration.

Structure Definition

```
typedef struct {
   DFLOAT darange;
   FLOAT offset;
   FLOAT gain;
   FLOAT cal_point;
   BOOL ch0pol, ch0prog, ch0ext;
   BOOL ch1pol, ch1prog, ch1ext;
   FLOAT ref;
   BYTE darange_calibrate;
} DSCDACALPARAMS;
```

Structure Members

Name	Description
darange	Desired full-scale voltage for custom D/A range setting, in volts. Used only when in Programmable mode, otherwise ignored. Range is anywhere between 1.0 and 10.0 in increments of .001.
offset	Error in D/A LSB counts at the mid-scale of the D/A range
gain	Error in D/A LSB counts at the full-scale (high end) of the D/A range
cal_point	Desired D/A point voltage for custom calibration setting, in volts. Range is anywhere in valid D/A range.

The following parameters are used only on Diamond-MM-AT. This board has independent D/A settings for each analog output channel. The parameters below correspond to the jumper settings on the board.

Name	Description
ch0pol, ch1pol	Polarity setting for each D/A channel: 0 = bipolar, 1 = unipolar
ch0prog, ch1prog	Programmable setting for each D/A channel: 0 = fixed range, 1 = programmable range
ch0ext, ch1ext	Reference source for each D/A channel: 0 = internal, 1 = external
ref	For internal use only.

The parameter darange_calibrate is used for Helios board only. This parameter is used to perform DA calibration verification on the mode being verified. The valid modes are specified in the DA table. The valid values for this parameter are
0 (0 to 10V UNIPOLAR),
1 (0 to 5V UNIPOLAR),
4 (+/- 2.5V BIPOLAR),
5 (+/- 5V BIPOLAR), and
6 (+/- 10V BIPOLAR).

dscDACalVerify

Verifies the accuracy of the most recent D/A auto-calibration process and returns the measured errors.

Function Definition

```
BYTE dscDACalVerify(DSCB board, DSCDACALPARAMS* params);
```

Function Parameters

Name	Description
board	The handle of the board to operate on
params	The D/A calibration settings to be used in the verification process

Return Value
Error code or 0.

These variables of the passed DSCDACALPARAMS structure are modified:

Name	Description
offset	Error in D/A LSB counts at the mid-scale of the D/A range
gain	Error in D/A LSB counts at the full-scale (high end) of the D/A range

DSCDACODE
DWORD

DscDACodeToVoltage
Utility function for converting DA units to voltage based on the DA settings used.

Function Definition
BYTE dscDACodeToVoltage(DSCB board, DSCDASETTINGS dasettings, DSCDACODE dacode, DFLOAT *voltage);

Function Parameters

Name	Description
board	The board handle
dasettings	DA settings used when the sample was taken
dacode	DA code to be converted
voltage	Pointer to the variable which will hold the converted voltage

Return Value
Error code or 0.

Usage Example

```
ERRPARAMS errparams;
DSCDASETTINGS dasettings;
DSCB dscb;
DSCDACODE dacode;
DFLOAT voltage;
BYTE result;

/* Board initialization code omitted */

dasettings.polarity = BIPOLAR;
dasettings.load_cal = TRUE;
dasettings.range = 10.0;

if ((result = dscDAConvert(dscb, 0, dacode)) != DE_NONE)
{
```

```
    dscGetLastError(&errparams);
    fprintf(stderr, "dscDAConvert failed: %s (%s)\n",
        dscGetErrorString(result), errparams.errstring);
    return result;
  }

  if ((result = dscDACodeToVoltage(dscb, dasettings, dacode, &voltage)) != DE_NONE)
  {
    dscGetLastError(&errparams);
    fprintf(stderr, "dscDACodeToVoltage failed: %s (%s)\n",
        dscGetErrorString(result), errparams.errstring);
    return result;
  }

  printf("DA code converted: %d, Actual voltage: %5.3lfV\n", dacode, voltage);
```

dscDAConvert

Performs a single D/A conversion on the given channel.

Function Definition
BYTE dscDAConvert(DSCB board, BYTE channel, DSCDACODE output_code);

Function Parameters

Name	Description
board	The handle of the board to operate on
channel	Selected output channel for the D/A conversion
output_code	Output code representing the desired output voltage. See the board's user manual for information on the translation between output voltage and output code. The range of output code for 12 bit DACs is 0 ~ 4095. The range of output code for 16 bit DACs is 0 ~ 65535.

Return Value
Error code or 0.

dscDAConvertScan

Performs a set of D/A conversions on multiple target channels.

Function Definition
BYTE dscDAConvertScan(DSCB board, DSCDACS* dscdacs);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscdacs	The D/A conversion scan settings to be used; see definition on page

Return Value
Error code or 0.

Usage in Helios and DMM32DX boards

In Helios SBC, the function expects that the user application has called the dscDASetSettings before calling the function. If the user called dscDASetSettings and set the DASIM bit to '1', this function will update all the 4 channels simulataneously. If the DASIM bit is set to '0', the function will start updating channels one at a time.

In DMM32DX implementation of the function, the DASIM usage is implied when this function is used. In DMM32DX, all the 4 channels D/A outputs are updated simultaneously.

dscDAConvertScanInt

Performs D/A conversions using interrupt-based I/O with one scan per interrupt. Note that calling this function only starts the interrupt operations in a separate system thread. The function call does not result in an atomic transaction with immediate results. Rather, it starts a real-time process that terminates only when the number of conversions reaches the maximum specified (one-shot mode) or if a call to dscCancelOp() is made.

Function Definition
BYTE dscDAConvertScanInt(DSCB board, DSCAIOINT*d scaioint);

Function Parameters

Name	Description
address	I/O address to access
value	A pointer to a word that stores the value

Return Value
Error code or 0.

DSCDACS

Structure containing D/A conversion scan settings. This structure is used when performing analog output on several channels simultaneously. One array is filled with data for each channel, and a second array is filled with flags indicating which channels to change. The unselected channels are unchanged.

Structure Definition

```
typedef struct {
   BOOL channel_enable[16];
   DSCDACODE* output_codes;
} DSCDACS;
```

Structure Members

Name	Description
channel_enable[16]	Input Array of flags that selects which analog output channels to change. Select TRUE for each location (0-15) for which you want to output new data.
output_codes	Input Array of output codes to send to the corresponding selected channels in channel_enable[]. Locations to use are 0-15.

DscDAGetOffsets

Reads the D/A offset voltages from the EEPROM for use in the autocalibration process. This is a utility function that normally does not need to be called in a user application. The functionDscDAAutoCal() uses this function during execution.

Function Definition
BYTE dscDAGetOffsets(DSCB board, DFLOAT* offsets, int count)

Function Parameters

Name	Description
board	The handle of the board to operate on
count	The number of offsets to read (this is usually 1 - the avarege D/A offset)
offsets	Array containing the D/A offset voltages stored in the EEPROM. For -AT boards and the Hercules, this is one value which represents the average offset across all D/A channels. It is used to normalize the D/A calibration to minimize average error across all D/A channels.

Return Value
Error code or 0.

DscDASetOffsets

Writes the D/A offset voltages from the EEPROM for use in the autocalibration process. This is a utility function that normally does not need to be called in a user application.

Function Definition
BYTE dscDASetOffsets(DSCB board, DFLOAT* offsets, int count)

Function Parameters

Name	Description
board	The handle of the board to operate on
count	The number of offsets to read (this is usually 1 - the average D/A offset)
offsets	Array containing the D/A offset voltages to be stored in the EEPROM. For AT boards and the Hercules-EBX, this is a single value which represents the average offset across all D/A channels. It is used to normalize the D/A calibration to minimize average error across all D/A channels.

Return Value
Error code or 0.

DscDASetPolarity

This is a convenience wrapper for DscDASetSettings().
It ONLY works on the Hercules II EBX SBC.

Function Definition
BYTE dscDASetPolarity (DSCB board, BYTE polarity);

Function Parameters

Name	Description
board	The handle of the board to operate on
polarity	Polarity setting (BIPOLAR or UNIPOLAR) to configure the analog output circuit

Return Value
Error code or 0.

DscDASetSettings

Sets the D/A configuration for future D/A conversions. A delay of approximately 10 microseconds occurs during this function call in order to allow the analog circuit on the board to settle on the new settings. On -AT (auto-calibrating) boards, if dasettings->load_cal is TRUE, then load the D/A calibration settings from the EEPROM for the current D/A configuration. This results in slightly more accurate D/A conversions but causes an additional, longer delay of approximately 10 milliseconds.

This function is ONLY supported by the Hercules II EBX, and Helios SBC's.

Function Definition
BYTE dscDASetSettings (DSCB board, DSCDASETTINGS* dasettings);

Function Parameters

Name	Description
board	The handle of the board to operate on
dasettings	The D/A conversion settings to be used in subsequent D/A conversions; these settings will be used for all D/A conversions.

Return Value
Error code or 0.

DSCDASETTINGS

Structure Definition

```
typedef struct {
    BYTE polarity;
    BYTE load_cal;
    FLOAT range;

     // New for the Helios and later D/A-capable boards.
    BYTE dasim ;     /* DASIM bit for simultaneous update*/
    BYTE daPolEn ;  /* 1 = Polarity from SW, 0 = polarity by HW jumpers.*/

    BYTE daUR ;  /* DA Unique Range. 1 will setup range settings for the CH*/
    BYTE daURChannel ;/* Unique range settings for the channel.Ignored if daUR is 0*/

} DSCDASETTINGS;
```

Structure Members

Name	Description
polarity	The polarity setting to use for D/A conversions; valid settings are BIPOLAR or UNIPOLAR
range	Absolute value of maximum voltage. For example, 10.0, or 5.0. Required for DscDACodeToVoltage and DscVoltageToDACode.
load_cal	If TRUE, the board will retrieve the calibration settings for the selected D/A range from the EEPROM and store it in the calibration circuit before performing future D/A conversions. This results in higher accuracy but causes a slight one-time delay of a few milliseconds. If FALSE, the board will use the current calibration settings and start the D/A conversion immediately. Once a particular D/A range's calibration settings are loaded into the board, they will remain loaded until the board powers off or a new set

	of settings is loaded.
dasim	If set to 1, set the DASIM bit in the DA mode control register. Currently only applies to the Helios SBC. When this bit is set to 1, the DA conversions are kept in the FPGA memory until a trigger is applied to enable the simultaneout output of all the DA channels on the corresponding analong outputs. It is recommended that the DASIM bit be used in the dscDAConvertScan function instead of in the dscDAConvert function. It MUST be noted that this bit is settable only in case of Helios SBC.
daPolEn	Only applicable to Helios board as of now. If set to 1, the DA polarity is controlled through software. Otherwise the polarity is provided by the hardware jumpers on the board. Thus when set to 1, the software settings override the hardware jumpers. It MUST be noted that this bit is settable only in case of Helios SBC.
daUR	When this field of the structure is set to 1, the Helios board will set the channel in the next field to have a unique range setting independent of the other 3 channels. It must be noted that if every channel is intended to be set to separate ranges, the dscDASetSettings function must be called 4 times and for every instance the daUR field needs to be set to 1.
daURChannel	The channel which gets the individual unique range setting. This channel number is different from the channel number for performing the actual DA conversion. This channel number is applicable only if the daUR is set to 1. If daUR is set to 0, the daURChannel value is ignored by the Helios board.

dscDIOClearBit

Sets the specified digital output bit of the specified port to 0 while leaving the other bits in their present states.

Some boards (Diamond-MM, Diamond-MM-AT, Diamond-MM-16-AT, Opal-MM, Pearl-MM, Quartz-MM) do not have readback capability on the digital output ports. Therefore the driver maintains its own history of the last-written values to these ports, so that it can correctly modify only the bit of interest. In order for this function to work correctly, the driver must always know the present state of the digital output bits. This can only happen if you always use the driver for digital output operations. If you mix driver calls with direct I/O to the digital output ports, the bit operations can fail, since the driver will have no record of what your program has done directly.

Function Definition
BYTE dscDIOClearBit (DSCB board, BYTE port, BYTE bit);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected output port
bit	Bit location (0-7) to write on the selected input port

Return Value
Error code or 0.

dscDIOInputBit

Receives a bit value from a given digital input port at a specified bit location (0-7). If port exceeds the board's maximum number of digital ports, the function returns DE_INVALID_PARM. If the board contains only a single digital port, then this value is ignored.

Function Definition

BYTE dscDIOInputBit (DSCB board, BYTE port, BYTE bit, BYTE* digital_value);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected input port
bit	Bit location (0-7) to read on the selected input port
digital_value	Present value of the bit read; 1 or 0

Return Value
Error code or 0.

dscDIOInputByte
Receives a BYTE from a given digital input port.

Function Definition
BYTE dscDIOInputByte (DSCB board, BYTE port, BYTE* digital_value);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected input port
digital_value	The BYTE value read from the selected digital input port; range is 0-255

Return Value
Error code or 0.

dscDIOOutputBit
Sets the specified digital output bit of the specified port to the specified value (1 or 0) while leaving the other bits in their present states.

Some boards (Diamond-MM, Diamond-MM-AT, Diamond-MM-16-AT, Opal-MM, Pearl-MM, Quartz-MM) do not have readback capability on the digital output ports. Therefore the driver maintains its own history of the last-written values to these ports, so that it can correctly modify only the bit of interest. In order for this function to work correctly, the driver must always know the present state of the digital output bits. This can only happen if you always use the driver for digital output operations. If you mix driver calls with direct I/O to the digital output ports, the bit operations can fail, since the driver will have no record of what your program has done directly.

Function Definition
BYTE dscDIOOutputBit(DSCB board, BYTE port, BYTE bit, BYTE bit_value);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected output port
bit	Bit location (0-7) to write on the selected input port
bit_value	Value to write to the bit; 1 or 0

Return Value
Error code or 0.

dscDIOOutputByte
Sends a BYTE to a given digital output port.

Function Definition
BYTE dscDIOOutputByte(DSCB board, BYTE port, BYTE digital_value);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected output port
digital_value	The value to write to the digital output port; range is 0-255

Return Value
Error code or 0.

dscDIOSetBit
Sets the specified digital output bit of the specified port to 1 while leaving the other bits in their present states.

NOTE: dscDIOSetConfig must be called to configure the direction prior to using this function.

Some boards (Diamond-MM, Diamond-MM-AT, Diamond-MM-16-AT, Opal-MM, Pearl-MM, Quartz-MM) do not have readback capability on the digital output ports. Therefore the driver maintains its own history of the last-written values to these ports, so that it can correctly modify only the bit of interest. In order for this function to work correctly, the driver must always know the present state of the digital output bits. This can only happen if you always use the driver for digital output operations. If you mix driver calls with direct I/O to the digital output ports, the bit operations can fail, since the driver will have no record of what your program has done directly.

Function Definition
BYTE dscDIOSetBit(DSCB board, BYTE port, BYTE bit);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected output port
bit	Bit location (0-7) to write on the selected input port

Return Value
Error code or 0.

dscDIOSetConfig
Sets the DIO port configuration for future DIO operations.

Note: See board user manual or 82C55 datasheet (if applicable) for config_byte details.

Function Definition
BYTE dscDIOSetConfig(DSCB board, BYTE* config_bytes);

Function Parameters

Name	Description
board	The handle of the board to operate on
config_bytes	The value(s) used to configure the digital I/O ports. See each board's user manual for information on the number and definition of the configuration bytes.

Return Value
Error code or 0.
Usage Example
The following code example configures digital I/O direction on a Garnet MM-48. See the Garnet user manual for more information.

```
ERRPARAMS errparams;
BYTE config_bytes[2];

config_bytes[0] = 0x9b;  // Set all ports on first chip to input
config_bytes[1] = 0x80;  // Set all ports on second chip to output

if ((result = dscDIOSetConfig(dscb, config_bytes)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscDIOSetConfig failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}
```

Next, an example for the use of DIO on the Helios.

•To configure the ports 1,2 and 3 (or A,B,C) of the Helios board, the config array must be filled as shown below.

config[0] = 0x00 ; // this tells DSCUD that the user wants to configure port 1,2 and 3. (or A,B,C)
config[1] = 0x## ; // configuration byte.

Refer to register at location Base + 11. It must be noted that when a bit for a corresponding port is set to 0 in this byte, the port will operate as an output port and 1 otherwise.

To Configure Vortex DIO Ports:

To configure the other two DIO ports which are provided and controlled by the Helios CPU chip the same config array needs to be filled but in a different way as shown below.

•Configure Port 4:
In order to configure the port 4, the array needs to have
config[0] = 0x01 ; // to indicate to DSCUD that Port 4 is being configured (or D)
config[1] = 0xFF ; // to setup the port as an input port
OR
config[1] = 0x00 ; // to setup the port as an output port.

•Configure Port 5:
In order to configure the port 5, the array needs to have
config[0] = 0x02 ; // to indicate to DSCUD that Port 5 is being configured (or E)
config[1] = 0xFF ; // to setup the port as an input port

OR
config[1] = 0x00 ; // to setup the port as an output port.

NOTE: Only values in config[0] are validated by DSCUD. The valid numbers to pass are only 0,1 and 2. All other values will return an Invalid Port ID error. THE VALUES IN config[1] ARE NOT VALIDATED BY DSCUD AND IT IS THE RESPONSIBILITY OF THE USER APPLICATION TO ASSIGN PROPER VALUES.

DSCEMMDIO

Structure to define the current configuration of an Emerald-MM-DIO board. This structure is used both as an input, to define the board's state, and as an output, to inquire about the board's state.

Structure Definition

```
typedef struct {
   BYTE DIOpins[6];
   BOOL lock_port[6];
   BYTE edge_polarity[3];
   BYTE edge_detect[3];
   BYTE edge_detect_clear[3];
   BYTE edge_detect_int[3];
   BOOL use_DIOpins;
   BOOL use_lock_port;
   BOOL use_edge_polarity;
   BOOL use_edge_detect;
   BYTE interrupt_status;
} DSCEMMDIO;
```

Structure Members

Name	Description
DIOpins[6]	The values to write to the DIO pins, or values read from DIO pins.
lock_port[6]	Values for the port lock control bits. A1 for any port prevents that port's output registers from being changed, while a 0 pemits them to be changed. See the Emerald-MM-DIO user manual for more details.
edge_polarity[3]	Three 8-bit values to determine the polarity of each pin on ports 0 through 2. Each BYTE represents one port with 8 pins: 0 = negative edge and 1 = positive edge for each bit in that port. Each bit of these three bytes determines the polarity of the corresponding I/O bit.
edge_detect[3]	3 byte values of either 0 or 1, corresponding to ports 0 - 2. A 1 for any port will enable edge detection for the entire port; a 0 will disable edge detection for the entire port.
edge_detect_clear[3]	3 byte values of either 0 or 1, corresponding to ports 0 - 2. A 1 for any port will clear that port's 8 edge detect registers; a 0 will leave the 8 edge detect registers unchanged.
edge_detect_int[3]	Read-only values specifying if an edge was detected on the pin since its port was last cleared. Each bit of each element corresponds to a digital input line on ports 0-2.
use_DIOpins	0 = ignore DIOpins[ ] array, 1 = use DIOpins[ ] array
use_lock_port	0 = ignore lock_port[ ] array, 1 = use lock_port[ ] array
use_edge_polarity	0 = ignore edge_polarity[ ] array, 1 = use edge_polarity[ ] array
use_edge_detect	0 = ignore edge_detect[ ] array, 1= use edge_detect[ ] array
interrupt_status	Read-only value indicating the interrupt request status of ports 0-2; bits 2-0 are the status for interrupt requests 2-0, respectively; 1 = interrupt pending / edge detected;

	0 = no interrupt pending / no edge detected

dscEMMDIOGetState

Returns the current configuration for an Emerald-MM-DIO board.

Function Definition
BYTE dscEMMDIOGetState(DSCB board, DSCEMMDIO* state)

Function Parameters

Name	Description
board	The handle of the board to operate on
state	The current configuration of the Emerald-MM-DIO board. See the definition of DSCEMMDIO.

Return Value
Error code or 0.

dscEMMDIOSetState

Sets the current configuration for an Emerald-MM-DIO board.

Function Definition
BYTE dscEMMDIOSetState(DSCB board, DSCEMMDIO* state)

Function Parameters

Name	Description
board	The handle of the board to operate on
state	The configuration to set on the Emerald-MM-DIO board. See the definition of DSCEMMDIO.

Return Value
Error code or 0.

dscEMMDIOResetInt

This function clears the interrupt request flip flop on Emerald-MM-DIO and resetes the status registers indicating edge detection. The user passes the desired behavior to the board through the DSCEMMDIORESETINT structure.

Function Definition
BYTE dscEMMDIOReestInt(DSCB board, DSCEMMDIORESETINT* resetinfo)

Function Parameters

Name	Description
board	The handle of the board to operate on
resetinfo	The configuration to set on the Emerald-MM-DIO board. See the definition of DSCEMMDIORESETINT

Return Value
Error code or 0.

DSCEMMDIORESETINT

Structure used to reset Emerald-MM-DIO user interrupts and edge detection activity.

Structure Definition

```
typedef struct {
   BOOL use_lock_port;
   BOOL lock_port[6];
   BYTE edge_detect_clear[3];
} DSCEMMDIORESETINT;
```

Structure Members

Name	Description
use_lock_port	0 = ignore lock_port values below; 1 = use lock_port values below
lock_port[6]	Values for the port lock control bits. A 1 for any port prevents that port's output registers from being changed, while a 0 pemits them to be changed. See the Emerald-MM-DIO user manual for more details.
edge_detect_clear[3]	A 1 for any port will clear that port's edge detect registers; a 0 will leave the edge detect registers unchanged.

DscEnhancedFeaturesEnable

Enables or disables the enhanced features. Enhanced features are features that are not backwards compatible with older versions of the board.

Function Definition

BYTE DscEnhancedFeaturesEnable(DSCB board, BOOL enable)

Function Parameters

Name	Description
board	The handle of the board to operate on
enable	True or false value on enabling enhanced features

Return Value

Error code or 0.

dscFreeBoard

Frees the system resources used by the given board and disables any of the board's currently running interrupt operations. This function must be called once for each board at the end of the program before calling dscFree().

Function Definition

BYTE dscFreeBoard(DSCB board);

Function Parameters

Name	Description
board	The handle of the board to operate on

Return Value

Error code or 0.

dscFree

Frees the system resources used by the universal driver. A call to this function must always be included at the end of any program utilizing Universal Driver.

Function Definition
BYTE dscFree(void);

Function Parameters
None.

Return Value
Error code or 0.

DscGetBoardMacro

Returns the corresponding board macro for the given board type string.

Function Definition
BYTE DscGetBoardMacro(char* boardtype, BYTE* macro)

Function Parameters

Name	Description
boardtype	The board type string to translate into a board macro
macro	The corresponding board macro

Return Value
Error code or 0.
Usage Example

```
BYTE board_macro;
ERRPARAMS errparams;

if ((result = dscGetBoardMacro("Prometheus", &board_macro)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscGetBoardMacro failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}
```

DscGetEEPROM

Reads 8-bit data from the EEPROM at the specified address. The EEPROM on all AT boards and Emerald-MM-8 contains 256 bytes. On the AT boards, all 256 bytes are addressable. On the Emerald-MM-8, only the lower 128 bytes are addressable. If there is a timeout failure waiting for the EEPROM to respond, the function will return DE_OPERATION_TIMED_OUT.

Function Definition
BYTE dscGetEEPROM(DSCB board, DWORD address, BYTE* data)

Function Parameters

Name	Description
board	The handle of the board to operate on

address	Address in the EEPROM to read data from. On AT boards, the range is 0-255. On Emerald-MM-8, the range is 0-127.
data	8-bit data from the specified address in the EEPROM

Return Value
Error code or 0.

dscGetErrorString

Returns the corresponding error string for the given error code.

Function Definition
char* dscGetErrorString(BYTE error_code)

Function Parameters

Name	Description
error_code	The error code to translate

Return Value
The char* error string corresponding to the given error code

Usage Example

```
ERRPARAMS errparams;

if ((result = dscInit(DSC_VERSION)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscInit failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}
```

DscGetFPGARev

Returns the FPGA revision of the board.

Function Definition
BYTE DscGetFPGARev(DSCB board, WORD* fpga)

Function Parameters

Name	Description
board	The handle of the board to operate on
fpga	The FPGA revision of the board

Return Value
Error code or 0.

dscGetLastError

Returns the most recent error that occurred within the universal driver.

Function Definition
BYTE dscGetLastError(ERRPARAMS* errparams)

Function parameters

Name	Description
errparams	Pointer to the ERRPARAMS structure that will hold the error code and error string of the most recent error

Return Value
The ErrCode and *errstring (in the parameter, *errparams) for the most recent error

DscGetReferenceVoltages

Reads the autocalibration reference voltages from the EEPROM for use in the autocalibration process. This is a utility function that normally does not need to be called in a user application. The functions DscADAutoCal() and DscDAAutoCal() use this function during execution.

Function Definition
BYTE dscGetReferenceVoltages (DSCB board, DFLOAT* refs)

Function Parameters

Name	Description
board	The handle of the board to operate on
refs	Array containing the reference voltages stored in the EEPROM. These are the voltages from the on-board precision voltage divider circuit that are used for autocalibration. During autocalibration, the driver software will measure the actual reference voltages and calibrate the board until the A/D readings match the stored values. The array size is 8 elements.

Return Value
Error code or 0.

DscGetRelay

Gets the state of one relay on the board.

Function Definition
BYTE dscGetRelay(DSCB board, BYTE relay, BYTE* value);

Function Parameters

Name	Description
board	The handle of the board to operate on
relay	Selected relay
value	Current relay state (0 or 1). Please consult the user manual for your particular product to determine the relationship between the value and the relay state

Return Value
Error code or 0.

DscGetRelayMulti

Simultaneously gets the state of multiple relays on the board. A "relay group" is a batch of relays that reside on the same internal board register:
Pearl-MM: 2 relay groups consisting of 8 relays each
Opal-MM: 1 relay group consisting of 8 relays

IR104: 3 relay groups: 2 contain 8 relays, 1 contains 4 relays
Diamond-MM-48-AT: 1 relay group consisting of 8 relays

Function Definition
BYTE dscGetRelayMulti(DSCB board, BYTE relayGroup, BYTE* value)

Function Parameters

Name	Description
board	The handle of the board to operate on
relayGroup	Selected relay group.
value	This is an eight-bit value that will store the current state of each relay in the group as one bit (0 or 1). The least significant bit in the value represents the first relay of the group.Please consult the user manual for your particular product to determine the relationship between the bit value (0 or 1) and the relay state.

Return Value
Error code or 0.

dscGetStatus

Returns the current status of any interrupt operations. This includes both the operation type (OP_TYPE_NONE or OP_TYPE_INT) and the current number of interrupts. On boards containing a FIFO (-AT boards and Prometheus), if the FIFO has overflowed, dscGetStatus() will reset the FIFO allowing interrupt operation to proceed. If interrupt operation has concluded (i.e., the current number of transfers has reached the maximum specified and one-shot mode has been specified) then dscGetStatus() will terminate interrupt processing.

Function Definition
BYTE dscGetStatus(DSCB board, DSCS* status)

Function Parameters

Name	Description
board	The handle of the board to operate on
status	Data structure used to store the current operation status

Return Value
Error code or 0.

DscGetTime

Simple clock routine returns a millisecond counter value. Usefull for measuring the elapsed milliseconds between events. The precision of the clock will vary by operating system. The counter value may reset on some operating systems if the program is run for an extended period of weeks or more.

Function Definition
dscGetTime(DWORD *ms);

Function Parameters

Name	Description
ms	Pointer to number which will hold the counter value

Return Value

Error code or 0.

Usage Example

```
DWORD start, now;

dscGetTime(&start);
myExtendedOperation();
dscGetTime(&now);

printf("Elapsed ms: %ld\n", now - start);
```

dscInit

Initializes the Universal Driver. A call to this function must always be included at the beginning of any program utilizing Universal Driver.

Function Definition
BYTE dscInit(WORD version);

Function Parameters

Name	Description
version	The expected version number of the driver. This number can be found at the top of file DSCUD.H. The driver will match this number with the number embedded in its code. If the numbers do not match, an error is returned. This feature is provided mainly to assist in debugging problems that may arise due to differences between different versions of the driver.

Return Value
Error code or 0.

Example Code

```
ERRPARAMS errparams;

if ((result = dscInit(DSC_VERSION)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscInit failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}
```

dscInitBoard

Initializes and sets the hardware configuration for the given board. This function must be called once for each board before any other function call to that board.

NOTE: The Athena data acquisition circuit is identical to the data acquisition implementation in the Prometheus CPU board. Version 5.8 of the Universal Driver supports the Athena via its support for the Prometheus DAQ. Athena customers should initialize the Universal Driver for the board using the board constant DSC_PROM. DSCUD 5.9 will include direct support for the Athena, so customers using 5.9 or later can initialize the board using the DSC_ATHENA constant instead.

Function Definition
BYTE dscInitBoard(BYTE boardtype, DSCCB* dsccb, DSCB* board);

Function Parameters

Name	Description
boardtype	The type of board to initialize; should be one of the valid board types listed in the Board Macros table.
dsccb	The hardware settings used to configure the given board

Return Value
The handle of the initialized board to be used in subsequent Universal Driver function calls

Usage Example

```
ERRPARAMS errparams;
DSCCB dsccb;
DSCB dscb;

memset(&dsccb, 0, sizeof(DSCCB));
dsccb.base_address = 0x300;
dsccb.int_level = 7;

if ((result = dscInitBoard(DSC_DMM32, &dsccb, &dscb)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscInitBoard failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}
```

DscInp

Board independent direct I/O. Reads a byte from the address.

Function Definition
BYTE dscInp(DWORD address, BYTE *value);

Function Parameters

Name	Description
address	I/O address to access
value	A pointer to a byte that stores the value

Return Value
Error code or 0.

DscInpl

Board independent direct I/O. Reads a double word from the address.

Function Definition
BYTE DscInpl(DWORD address, DWORD*value);

Function Parameters

Name	Description
address	I/O address to access

value	A pointer to a double word that stores the value

Return Value
Error code or 0

DscInpw

Board independent direct I/O. Reads a word from the address.

Function Definition
BYTE dscInpw(DWORD address, WORD *value);

Function Parameters

Name	Description
address	I/O address to access
value	A pointer to a word that stores the value

Return Value
Error code or 0.

DscInpws

Board independent direct I/O. Reads N words from the address.

Function Definition
BYTE DscInpws(DWORD address, WORD *buffer, WORD n);

Function Parameters

Name	Description
address	I/O address to access
buffer	A pointer to N words that store the values
n	Number of words to read

Return Value
Error code or 0.

DscInterruptControl

Controls GPIO11 interrupt configuration such as clear flipflop. Disable and Enable interrupt.

Function Definition
BYTE dscPauseOp(DSCB board, BYTE* config_byte)

Function Parameters

Name	Description
board	The handle of the board to operate on
config_byte	Configuration byte to set to interrupt control register

Return Value
0.

DscIR104ClearRelay

Turns off an individual relay on the IR104 (opens the contacts). The open-contact state is the power-off state.

Function Definition
BYTE dscIR104ClearRelay(DSCB board, BYTE relay);

Function Parameters

Name	Description
board	The board handle
relay	The relay number, ranging from 1 – 20 (not 0-19)

Return Value
Error code or 0.

DscIR104OptoInput

Reads an individual optoisolated digital input on an IR104.

Function Definition
BYTE dscIR104OptoInput(DSCB board, BYTE opto, BYTE *value);

Function Parameters

Name	Description
board	The board handle
opto	The optoisolated input channel number, ranging from 1 – 20 (not 0-19)
values	A pointer to a byte that stores the value (0 or 1)

Return Value
Error code or 0.

DscIR104RelayInput

Reads the current state of an individual relay on the IR104.

Function Definition
BYTE dscIR104RelayInput(DSCB board, BYTE relay, BYTE* value);

Function Parameters

Name	Description
board	The board handle
relay	The relay number, ranging from 1 – 20 (not 0-19)
value	A pointer to a byte that stores the value (0 or 1)

Return Value
Error code or 0.

DscIR104SetRelay

Sets the state of one relay on the board to either open or closed.

Function Definition
BYTE dscSetRelay(DSCB board, BYTE relay, BYTE value);

Function Parameters

Name	Description
board	The handle of the board to operate on
relay	Selected relay
value	New relay state (0 or 1). Please consult the user manual for your particular product to determine the relationship between the value and the relay state.

Return Value
Error code or 0.

DSCQMMCMR

Structure containing configuration data for the Counter Mode Register of a counter on the 9513 chip on Quartz-MM. Each counter on each chip has its own Counter Mode Register. The Counter Mode Register must be programmed before using the counter.

Structure Definition

```
typedef struct {
   BYTE counter;
   BYTE gating_control;
   BYTE active_source_edge;
   BYTE count_source;
   BYTE special_gate;
   BYTE reload_source;
   BYTE cycle;
   BYTE count_type;
   BYTE count_direction;
   BYTE output_control;
} DSCQMMCMR;
```

Structure Members

Name	Description
counter	Counter number: 1-5 for QMM-5, 1-10 for QMM-10
gating_control	Gating control is the means by which the counter is enabled or disabled in hardware. TCn-1 means the terminal count state of the counter one lower in number. When TCn-1 is specified, the counter will only count when the lower-numbered counter is in its terminal count state and a source edge is applied to the selected counter source. Specifying TCn-1 for counter 1, as well as counter 6 in QMM-10, is invalid, since in these cases counter n-1 does not exist. Specifying TCn-1 as the gating control allows several consecutive counters to be cascaded to form a wider counter in multiples of 16 bits. For example, counters 1 and 2 can be cascaded together to form a 32-bit-wide counter by specifying no gating for counter 1 and TCn-1 for counter 2, and setting both counters to the same input source. Specifying edge gating causes the counter to be triggered on the first matching edge after the counter is armed. In some modes, only the first such edge affects the counter; in other modes, successive edges also affect operation. Select from the gating control macros in above list
active_source_edge	Active edge for the counter: 0 = rising edge, 1 = falling edge
count_source	Counter source. Note that each counter may take its source from any of the source pins, any of the gate pins, and several other inputs. There is no fixed relationship between counter n, source n, and gate n. Counters may share the same source and gate pins. Select from counter source macros in above list

special_gate	Special gate function; refer to the 9513 datasheet for details. 0 = enable, 1 = disable
reload_source	The source of the data that will be used to reload the counter when it reaches terminal count. 0 = Load register; 1 = Load or Hold register (depends on the counter configuration)
cycle	Selects either one-shot or recycle mode. In one-shot mode, the counter counts once to terminal count (TC) and then automatically disarms. In recycle mode the counter reloads from the selected reload source when it reaches TC and continues repeatedly until it is disarmed. 0 = one-shot, 1 = recycle
count_type	Selects binary or BCD (binary coded decimal) counting mode. In binary mode each 16-bit counter has a maximum count value of 65535. In BCD mode each 4-bit nybble represents a digit from 0-9, and the maximum count is 9999. in BCE mode, a count of 99 will be represented by the binary no. 0000 0000 1001 1001 or 153 in standard notation. 0 = binary, 1 = BCD
count_direction	Selects up or down counting; 0 = down, 1 = up
output_control	Determines the behavior of the counter's output pin. Specifying Toggle mode causes the output of the counter to generate a square wave with 50 percent duty cycle (high and low periods equal in duration) when the counter is in repetitive count mode. In Toggle mode, each high and low level lasts for one entire count period, and the output changes state on each TC until the counter is disarmed. Specifying TC pulse causes the counter to output a pulse of the selected polarity equal to one period of the counter's input source. If the counter is being used as a totalizer and the output is not important, it can be specified as inactive, output low, or high impedance. Select from output control macros in above list.

DSCQMMMCC

Structure containing configuration data for Multiple Counter Control. This structure is used to execute load, arm, disarm, and save actions. An Arm command causes the counter to start counting, and a Disarm command causes it to stop. The counter will not count until it receives an Arm command. The Save command or Disarm and Save command are used to latch the current contents of the counter into the Hold register for reading. If just the Save command is executed, the counter will continue to count; this is equivalent to the Lap button on a stopwatch. After executing one of these commands, use the dscReadHoldRegister function to read the latched data. The Load command or Load and Arm command are used to load the contents of the Load register into the counter before or while arming it. Before executing one of these commands, use the dscSetLoadRegister() function to load the Load register with the desired data. If you are using a QMM-5 board with only one 9513 chip, or if you are acting on only one counter chip, set the other counter's action to QMM_ACTION_NONE (0).

Structure Definition

```
typedef struct {
   BYTE group1_action;
   BYTE group1_counter_select;
   BYTE group2_action;
   BYTE group2_counter_select;
} DSCQMMMCC;
```

Structure Members

Name	Description
group1_action	Determines the action for counter group 1 (counters 1-5); select from counter action macros in above list
group1_counter_select	Selects which counters will be acted on by the action in group1_action. All counters in

	the group are acted on in the same way at the same time.
group2_action	Determines the action for counter group 2 (counters 6-10)
group2_counter_select	Selects which counters will be acted on by the action in group2_action All counters selected in each group receive the same action specified for that group. The parameters for group1_counter_select and group2_counter_select use the following format: Group1_counter_select: Bit No. 7 6 5 4 3 2 1 0 Name 0 0 0 CTR5 CTR4 CTR3 CTR2 CTR1 Group2_counter_select: Bit No. 7 6 5 4 3 2 1 0 Name 0 0 0 CTR10 CTR9 CTR8 CTR7 CTR6 CTRn select counter n; 1 = selected, 0 = not selected

DSCQMMMMR

Structure containing configuration data for the Quartz-MM Master Mode Register. Each 9513 chip has its own master mode register. The master mode register must be programmed before any counter on that chip can be used.

Structure Definition

```
typedef struct {
   BYTE counter_group;
   BYTE fout_divider;
   BYTE fout_source;
   BYTE compare1_enable;
   BYTE compare2_enable;
   BYTE tod_mode;
} DSCQMMMMR;
```

Structure Members

Name	Description
counter_group	The control code to write to or read from the control word register.
fout_divider	Divider for the 9513 chip's internal frequency generator; range is 1-16. The output frequency of the fout pin is fout_source / fout_divider. The fout (frequency output) pin from counter chip 1 is available on pin 31 of the board's I/O header and may be used by counters 1-5. The fout signal from chip 2 is not available externally but may be used within the chip by counters 6-10.
fout_source	Source frequency for the 9513 chip's internal frequency generator. Follow link to the macro list below to select.
compare1_enable	TRUE = Enable the comparator on counter 1; FALSE = disabled.
compare2_enable	TRUE = Enable the comparator on counter 2; FALSE = disabled. The comparator function allows the use of counters 1 and 2 on each 9513 chip as totalizers with alarm outputs. When the count of any counter whose comparator is enabled equals the value programmed in that counter's alarm register, the counter output will be true, otherwise it will be false. The alarm register is programmed with function dscQMMSpecialCounterFunction.
tod_mode	Time-of-day mode; select from time-of-day macros by following the QMM Macros link below. The time-of-day circuit is intended to be used with an input frequency of 50Hz, 60Hz, or 100Hz. The divider of 5, 6, or 10 is chosen to match the input frequency and provide a 0.1 second resolution. On a CPU with a built-in real-time clock, the time-of-day circuit is generally not needed.

Also see this list of QMM Macros.

DSCQMMPWM

Structure containing configuration data for pulse width modulation function on Quartz-MM. The structure contains both input and output parameters.

Structure Definion

```
typedef struct {
   BYTE init;
   BYTE counter;
   FLOAT output_freq;
   FLOAT duty_cycle;
   DWORD input_freq;
   WORD load_reg;
   WORD hold_reg;
   BYTE hit_extreme;
} DSCQMM_PWM;
```

Structure Members

Name	Description
init	1 if the current function call is an initializing call, i.e. the first call for this counter and this output frequency. 0 if the current function call is just to modify the duty cycle of a currently-running PWM signal.
counter	Counter number: 1-5 for QMM-5, 1-10 for QMM-10
output_freq	Desired output frequency; the function will select the best clock source from the 9513 chip's built-in frequency generator to achieve the desired output frequency with maximum duty cycle resolution
duty_cycle	Desired duty cycle in percent: range is 0 - 99.99 in steps of .01; a duty cycle of 100% is not possible
input_freq	Input clock source selected by the function
load_reg	Resulting load register value from the selected output frequency and duty cycle
hold_reg	Resulting hold register value from the selected output frequency and duty cycle
hit_extreme	1 if the function has reached either duty cycle limit (0 or 99.99); 0 otherwise

DSCQMMSCF

Structure containing configuration data for Special Counter Functions. Refer to the 9513 datasheet for information on special functions.

Structure Definition

```
 /* QMM special counter actions */
 #define QMM_SPECIAL_CLEAR_TOGGLE_OUTPUT    0
 #define QMM_SPECIAL_SET_TOGGLE_OUTPUT      1
 #define QMM_SPECIAL_STEP_COUNTER           2
 #define QMM_SPECIAL_PROGRAM_ALARM          3
typedef struct {
   BYTE counter;
   BYTE action;
   WORD alarm_value;
} DSCQMMSCF;
```

Structure Members

Name	Description
counter	Counter no., 1-10
action	Special action for this counter; select from special counter action macros in above list
alarm_value	Alarm value, range 0-65535 Valid only if alarm is enabled (action = QMM_SPECIAL_PROGRAM_ALARM) and counter no. is 1, 2, 6, or 7.

DscOptoGetPolarity

Gets the polarity of the Diamond-MM-48-AT optoinputs, which is set by the POL jumper. Please consult the user manual for more information on this feature.

Function Definition
BYTE dscOptoGetPolarity(DSCB board, BYTE* polarity);

Function Parameters

Name	Description
board	The handle of the board to operate on
polarity	The polarity of the optoinput (0 or 1)

Return Value
Error code or 0.

DscOptoGetState

Gets the overall state of optoinputs from the board, including edge detection, polarity and input state.

Function Definition
BYTE dscOptoGetState(DSCB board, DSCOPTOSTATE* state);

Function Parameters

Name	Description
board	The handle of the board to operate on
state	Pointer to the DSCOPTOSTATE struct that will receive the optoinput state information.

Return Value
Error code or 0.

DscOptoInputBit

Gets the state of a single optoinput from the board.

Function Definition
BYTE dscOptoInputBit(DSCB board, BYTE port, BYTE bit, BYTE * optoValue);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected optoinput group
bit	Specific optoinput from the group (0-7)
optoValue	The return value corresponds to the optoinput state (0 or 1)

Return Value
Error code or 0.

DscOptoInputByte

Simultaneously gets the state of 8 optoinputs from the board.

Function Definition
BYTE dscOptoInputByte (DSCB board, BYTE port, BYTE * optoValue);

Function Parameters

Name	Description
board	The handle of the board to operate on
port	Selected optoinput group
optoValue	Each bit in the return value corresponds to one optoinput state (0 or 1)

Return Value
Error code or 0.

DSCOPTOSTATE

Structure containing information on the current optoinput state of the board. For use with the dscOptoGetState and dscOptoSetState functions.

Structure Definition

```
 #define DSCUD_MAX_OPTO 8
typedef struct {
   BYTE edge_polarity[DSCUD_MAX_OPTO];
   BYTE edge_detect_enab[DSCUD_MAX_OPTO];
   BYTE edge_status[DSCUD_MAX_OPTO];
   BYTE oint_state[DSCUD_MAX_OPTO];
   BYTE dmm48at_oint_state;
} DSCOPTOSTATE;
```

Structure Members

Name	Description
edge_polarity	Polarity state of each optoinput
edge_detect_enab	Edge detect enable state of each optoinput 0 = disabled, 1 = enabled
edge_status	Edge detect status for each optoinput. 0 = no edge detected since last check 1 = edge detected since last check
oint_state	Optoinput state (depends on polarity for boards which support this feature.)
dmm48at_oint_state	When using opto edges to drive interrupts on the Diamond-MM-48-AT, the edge registers must be read (and subsequently reset) by the kernel interrupt handler before the user has a chance to read them back. The kernel interrupt handler will save the opto state register in this location so that the user can see which edge triggered the interrupt inside of the user interrupt function.

DscOptoSetState

Sets the overall state of optoinputs on the board, including edge detection and polarity.

Function Definition

BYTE dscOptoSetState(DSCB board, DSCOPTOSTATE* state);

Function Parameters

Name	Description
board	The handle of the board to operate on
state	Pointer to the DSCOPTOSTATE struct that holds the new optoinput state information (only the output fields are used.)

Return Value
Error code or 0.

DscOutp

Board independent direct I/O. Write a byte to that address.

Function Definition
DscOutp(DWORD address, BYTE value);

Function Parameters

Name	Description
address	I/O address to access
value	A byte to write

Return Value
Error code or 0.

DscOutpl

Board independent direct I/O. Write a double word to that address.

Function Definition
DscOutpl(DWORD address, DWORD value);
Function Parameters

Name	Description
address	I/O address to access
value	A word to write

Return Value
Error code or 0.

DscOutpw

Board independent direct I/O. Write a word to that address.

Function Definition
DscOutpw(DWORD address, WORD value);

Function Parameters

Name	Description
address	I/O address to access
value	A word to write

Return Value
Error code or 0.

DscOutpws

Board independent direct I/O. Write a word to that address.

Function Definition
DscOutpws(DWORD address, WORD *buffer, WORD n);

Function Parameters

Name	Description
address	I/O address to access
buffer	A pointer to N words to write
n	Number of words to write

Return Value
Error code or 0.

DSCPWM

Structure containing pulse width modulation parameters for DscPWMFunction().

Structure Definition

```
typedef struct {
   DFLOAT output_freq;
   FLOAT duty_cycle;
   BYTE polarity;
   BYTE pwm_circuit;
   BOOL output_enab;
} DSCCR;
```

Structure Members

Name	Description
output_freq	Output frequency of the PWM signal (i.e. frequency of subsequent duty cycles.)
duty_cycle	Percentage of time that the signal will be in the active state. Valid duty cycles range from 0.0 to 100.0.
polarity	Determines the active state of the PWM signal (for boards that support this.) 0 = high, 1 = low
pwm_circuit	Which PWM circuit this function should configure. Depending on the board this will represent either physical or logical counters.
output_enab	Enable signal output. 1 = enable, 0 = disable

DscPWMClear

This function is used to clear (reset) a PWM circuit on the Hercules-EBX. Please consult the Hercules user manual for more information on the PWM circuit.
Function Definition
BYTE dscPWMClear(DSCB board, BYTE pwmCircuit);

Function Parameters

Name	Description

board	The board handle
pwmCircuit	Which PWM circuit (0-3) to reset.

Return Value
Error code or 0.

DscPWMConfig

This function is used to directly write to the PWM configuration register on the Hercules-EBX. Please consult the Hercules-EBX user manual for more information on the appropriate values.

Function Definition
BYTE dscPWMConfig(DSCB board, BYTE * config_byte);

Function Parameters

Name	Description
board	The board handle
config_byte	Pointer to the BYTE value to be written to the PWM config register

Return Value
Error code or 0.

DscPWMFunction

This is the main function used to generate PWM signals on the Hercules-EBX. Please consult the Hercules user manual for more information on the PWM circuit. This function can also be used with the Quartz-MM family of boards, but we recommend using the DscQMMPulseWidthModulation() function for maximum compatibility.

Function Definition
BYTE dscPWMFunction(DSCB board, DSCPWM* dscpwm);

Function Parameters

Name	Description
board	The board handle
dscpwm	Pointer to the DSCPWM struct containing the PWM parameters.

Return Value
Error code or 0.

DscPWMLoad

This function loads the specified counter of the specified PWM circuit with value. The behavior of this function is specific to the type of board you are calling this function with. Please note that this function should be used only for direct manipulation of the counters, and is not intended for general PWM configuration. We recommend using the DscPWMFunction() function to do most PWM signal manipulation.

Function Definition
BYTE dscPWMLoad(DSCB board, BYTE pwm_circuit, BYTE counter, DWORD value);

Function Parameters

Name	Description
board	The board handle

pwm_circuit	The PWM circuit to affect.
counter	Which counter of the PWM circuit to load
value	Value to load into the counter

Return Value
Error code or 0.

DscQMMPulseWidthModulation

This function generates an output signal on the selected counter on Quartz-MM to the selected frequency with the selected duty cycle. The duty cycle is defined as the percent of time that the output is high. A 50% duty cycle represents a square wave (equal high and low periods), while a duty cycle of 0% means always low and a duty cycle of 100% means always high. The range for duty cycle is 0 to 99.99 in steps of .01. a duty cycle of 100% is not possible with this function.

The function takes as inputs the desired counter, output frequency, and duty cycle. It returns status information indicating the selected input frequency, the resulting load and hold register values, and a flag indicating whether the function has reached either duty cycle limit.

Function Definition
BYTE dscQMMPulseWidthModulation(DSCB board, DSCQMMPWM* pwm);

Function Parameters

Name	Description
board	The handle of the board to operate on
pwm	Structure containing configuration data for the function; see definition DSCQMMPWM.

Return Value
Error code or 0.

DSCQMMPWM

Structure containing configuration data for pulse width modulation function on Quartz-MM. The structure contains both input and output parameters.

Structure Definion

```
typedef struct {
   BYTE init;
   BYTE counter;
   FLOAT output_freq;
   FLOAT duty_cycle;
   DWORD input_freq;
   WORD load_reg;
   WORD hold_reg;
   BYTE hit_extreme;
} DSCQMM_PWM;
```

Structure Members

Name	Description
init	1 if the current function call is an initializing call, i.e. the first call for this counter and this output frequency. 0 if the current function call is just to modify the duty cycle of a

	currently-running PWM signal.
counter	Counter number: 1-5 for QMM-5, 1-10 for QMM-10
output_freq	Desired output frequency; the function will select the best clock source from the 9513 chip's built-in frequency generator to achieve the desired output frequency with maximum duty cycle resolution
duty_cycle	Desired duty cycle in percent: range is 0 - 99.99 in steps of .01; a duty cycle of 100% is not possible
input_freq	Input clock source selected by the function
load_reg	Resulting load register value from the selected output frequency and duty cycle
hold_reg	Resulting hold register value from the selected output frequency and duty cycle
hit_extreme	1 if the function has reached either duty cycle limit (0 or 99.99); 0 otherwise

DscRegisterRead

Reads a BYTE from an internal register on a board. Refer to the board's user manual for the I/O map and register definitions. This function allows for direct access to the board for operations that cannot be implemented with existing driver function calls. This function is equivalent in effect to the common library function inp(). It uses the driver's internals to manage the complications that arise when attempting direct I/O to the hardware in some operating systems.

Function Definition
BYTE DscRegisterRead(DSCB board, WORD address, BYTE* data);

Function Parameters

Name	Description
board	The handle of the board to operate on
address	I/O port on the board to read from. This is indicated as the offset from the board's base address, starting with 0. On a Diamond-MM-32-AT, whose base address is 768 (0x300), address ranges from 0 to 15, not 768 - 783 (0x300 - 0x30F).
data	The 8-bit data from the specified register on the board

Return Value
Error code or 0

DscRegisterWrite

Writes a BYTE to an internal register on a board. Refer to the board's user manual for the I/O map and register definitions. This function allows for direct access to the board for operations that cannot be implemented with existing driver function calls. This function is equivalent in effect to the common library function outp(). It uses the driver's internals to manage the complications that arise when attempting direct I/O to the hardware in some operating systems.

Function Definition
BYTE DscRegisterWrite(DSCB board, WORD address, BYTE data);

Function Parameters

Name	Description
board	The handle of the board to operate on
address	I/O port address to write to. This is indicated as the offset from the board's base address, starting with 0. On a Diamond-MM-32-AT, whose base address is 768

	(0x300), address ranges from 0 to 15, not 768 - 783 (0x300 - 0x30F).
data	The 8-bit data to write to the specified register on the board

Return Value
Error code or 0.

DSCS

Structure containing interrupt operation status information. Used by dscGetStatus().

Structure Definition

```
typedef struct {

   BYTE op_type;
   DWORD transfers;
   DWORD total_transfers;
   DWORD da_transfers;
   DWORD da_total_transfers;
   DWORD overflows;

} DSCS;
```

Structure Members

Name	Description
op_type	The current operation type, either OP_TYPE_NONE or any bit-wise combination of INT_TYPE_* currently running
transfers	The number of A/D transfers that have taken place for the current cycle (identical to total_transfers in non-cycle mode.)
total_transfers	The total number of A/D conversions that have taken place since the interrupt operation began. When scans are occurring, this variable increments by the scan size.
da_transfers	Similar to transfers but for D/A conversions.
da_total_transfers	Similar to total_transfers but for D/A conversions.
overflows	On boards with FIFOs (AT boards and Prometheus), this parameter indicates the number of times the FIFO overflowed during an A/D interrupt operation. During correct operation, this number should always be zero. If it ever becomes nonzero, it means that some A/D data was missed, because the A/D FIFO reached its limit before the interrupt routine could read data out. When the FIFO becomes full, it will not accept any more data, although the board continues to take samples. Once the interrupt routine reads data from the FIFO, it will resume storing A/D data. This value is reset to zero each time an interrupt operation starts. This parameter can be used to verify that A/D data was acquired accurately with no missing samples. It can also be used to search for the maximum sampling rate achievable on your computer for the given settings.

DSCSAMPLE

DSCSAMPLE is defined as a constant to mean SWORD, which is a Signed WORD, by many compilers (please check yours) it is 16-bits. The MSB is used for sign, the rest of the bits are used to store the number.

DscSetEEPROM

Writes 8-bit data to the EEPROM at the specified address. The EEPROM on all AT boards and Emerald-MM-8 contains 256 bytes. On the AT boards, all 256 bytes are addressable. On the Emerald-MM-8, only the lower 128 bytes are

addressable. If there is a timeout failure waiting for the EEPROM to respond, the function will return DE_OPERATION_TIMED_OUT.

Function Definition
BYTE dscSetEEPROM(DSCB board, DWORD address, BYTE data)

Function Parameters

Name	Description
board	The handle of the board to operate on
address	Address in the EEPROM to write. On AT boards, the range is 0-255. On Emerald-MM-8, the range is 0-127.
data	8-bit data to write to the specified address in the EEPROM

Return Value
Error code or 0.

DscSetCalMux

Turns the calibration multiplexor on or off. The calmux is used only for autocalibration. A user application will only need this function if it is implementing a custom calibration routine. A slight delay of about 10 microseconds occurs when enabling or disabling the calibration multiplexor in order to allow the analog input circuit to settle.

Function Definition
BYTE dscSetCalMux(DSCB board, BOOL on);

Function Parameters

Name	Description
board	The handle of the board to operate on
on	TRUE for on or FALSE for off

Return Value
Error code or 0.

DscSetReferenceVoltages

Stores the autocalibration reference voltages in the EEPROM. These voltages are measured on-board and stored in the EEPROM during the factory calibration process. This is a utility function that normally does not need to be called in a user application.

Function Definition
BYTE dscSetReferenceVoltages(DSCB board, DFLOAT* refs);

Function Parameters

Name	Description
board	The handle of the board to operate on
refs	Array containing the reference voltages stored in the EEPROM. These are the voltages from the on-board precision voltage divider circuit that are used for autocalibration. During autocalibration, the driver software will measure the actual reference voltages and calibrate the board until the A/D readings match the stored values. The array size is 8 elements.

Return Value
Error code or 0.

DscSetRelay

Sets the state of one relay on the board to either open or closed.

Function Definition
BYTE dscSetRelay(DSCB board, BYTE relay, BYTE value);

Function Parameters

Name	Description
board	The handle of the board to operate on
relay	Selected relay
value	New relay state (0 or 1). Please consult the user manual for your particular product to determine the relationship between the value and the relay state.

Return Value
Error code or 0.

DscSetRelayMulti

Simultaneously sets the state of multiple relays on the board. A "relay group" is a batch of relays that reside on the same internal board register:
Pearl-MM: 2 relay groups consisting of 8 relays each
Opal-MM: 1 relay group consisting of 8 relays
IR104: 3 relay groups: 2 contain 8 relays, 1 contains 4 relays
Diamond-MM-48-AT: 1 relay group consisting of 8 relays

Function Definition
BYTE dscSetRelayMulti(DSCB board, BYTE relayGroup, BYTE value);

Function Parameters

Name	Description
board	The handle of the board to operate on
relayGroup	Selected relay group.
value	This is an eight-bit value. Each relay of the group will be set to its corresponding bit (0 or 1) of this value. The least significant bit in the value represents the first relay of the group. Please consult the user manual for your particular product to determine the relationship between the bit value (0 or 1) and the relay state.

Return Value
Error code or 0

DscSetSystemPriority

Sets the system priority for the interrupt processing thread.

Function Definition
BYTE DscSetSystemPriority(DWORD priority)

Function Parameters

Name	Description
priority	The system specific priority to use.

Return Value
Error code or 0.

System Priority on Linux and QNX
A thread is created by the driver in user mode which communicates with the interrupt handling functions. This thread is responsible for copying sample data into the user buffer. It also is responsible for calling the user provided user interrupt function. By default, the driver sets the highest priority available for this thread so that these tasks complete with a minimum of latency. The functions pthread_set_schedpolicy() and pthread_set_schedparam() are used to set this priority. The user can set a lower priority to this thread using dscSetSystemPriority(). See the pthreads documentation for scheduling for information on what priority numbers are available.

DscSetTrimDac

This function modifies the onboard autocalibration TrimDACs for all -AT boards and the Hercules-II. This function should be called only for debugging or advanced use, as improper values will throw the board out of calibration.

Function Definition
BYTE dscSetTrimDac(DSCB board, DWORD trimDac, BYTE value)

Function Parameters

Name	Description
board	The handle of the board to operate on
trimDac	Which TrimDAC (0-7) to modify
value	8-bit data to write to the specified TrimDAC

Return Value
Error code or 0.

dscSetUserInterruptFunction

Installs a user-provided function attached to all interrupt types for later execution either alone or in conjunction with another driver interrupt function. In DOS, the user-provided function may not call other Universal Driver functions to operate on the board, since at the time of execution the board will be locked by the driver and unavailable to the function. In other operating systems this restriction does not apply.

Function Definition
BYTE dscSetUserInterruptFunction (DSCB board,DSCUSERINTFUNCTION* dscuserintfunction);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscuserintfunction	Data structure containing information on the user interrupt function. See the definition DSCUSERINTFUNCTION

Return Value
Error code or 0.

DscSetUserInterruptFunctionType

Installs a user-provided function on a single interrupt type for later execution either alone or in conjunction with another driver interrupt function. In DOS, the user-provided function may not call other Universal Driver functions to operate on the board, since at the time of execution the board will be locked by the driver and unavailable to the function. In other operating systems this restriction does not apply.

Function Definition
BYTE dscSetUserInterruptFunctionType(DSCB board, DSCUSERINTFUNCTION* dscuserintfunction, DWORD int_type);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscuserintfunction	Data structure containing information on the user interrupt function. See the definition DSCUSERINTFUNCTION.
int_type	The interrupt type (one of INT_TYPE_*) to attach the function to

Return Value
Error code or 0.

DscSleep

Delays program execution for the specified number of milliseconds.

Function Definition
BYTE dscSleep(DWORD ms);

Function Parameters

Name	Description
ms	Number of milliseconds to sleep

Return Value
Error code or 0.

dscUserInt

Starts execution of a user interrupt function alone (without another driver interrupt function).

Function Definition
BYTE dscUserInt(DSCB board, DSCUSERINT* dscuserint, DSCUserInterruptFunction func);

Function Parameters

Name	Description
board	The handle of the board to operate on
dscuserint	Structure containing configuration information for the user interrupt operation
func	The name of the user interrupt function

Return Value
Error code or 0.

DSCUSERINT

Structure containing configuration data for user interrupt operation.

Structure Definition
```
typedef struct {
   BYTE intsource;
   FLOAT rate;
   BYTE clksource;
   BYTE counter;
   DWORD int_type;
   DSCUserInterruptFunction func;
} DSCUSERINT;
```

Structure Members

Name	Description
intsource	This field indicates the source of the interrupts. You may select USER_INT_SOURCE_INTERNAL to use an on-board counter/timer, or USER_INT_SOURCE_EXTERNAL to use an external clock. Refer to each board's user manual for details on the options for counter/timer and external clock.
rate	Required only when intsource = USER_INT_SOURCE_INTERNAL. If you select a non-zero value, the on-board counter/timer will be programmed to generate interrupts at this rate. If this value is set to 0 then the dscUserInt function will not program the board's counter/timer with a new value, and will assume that the counter has been preprogrammed. This is useful if you want to control the counter/timer rate yourself.
counter	If you choose intsource = USER_INT_SOURCE_INTERNAL, you must specify which on-board counter you will be using to generate the interrupts. Each board has different valid options for clksource.
clksource	Selects the source of the clock that drives the on-boardcounter-timer when intsource = USER_INT_SOURCE_INTERNAL. On Diamond-MM-32: 0 = internal 10MHz, 1 = internal 10kHz, 2 = external. On all other boards: 0 = internal, 1 = external.
int_type	Returns the interrupt type that the driver attached the interrupt handler to. This depends on which board you are using, but will generally be INT_TYPE_DIO or INT_TYPE_COUNTER.
func	This is a reference pointer to the user interrupt function that was attached to the interrupt. This is usually passed in as a parameter to the dscUserInt () function.

DSCUserInterruptFunction

Functions provided by the user for use in user interrupt calls must conform to this type.

Datatype Definition
```
typedef void (*DSCUserInterruptFunction) (void* parameter);
```

DSCUSERINTFUNCTION

Structure containing information about the user interrupt function and execution time.

Structure Definition
```
 typedef struct {
   DSCUserInterruptFunction func;
   BYTE int_mode;
   DWORD int_type;
 } DSCUSERINTFUNCTION;
 typedef void (*DSCUserInterruptFunction) (void* parameter);
 // int_mode: user interrupt execution time choices
```

```
#define USER_INT_CANCEL    0
#define USER_INT_AFTER     1
#define USER_INT_INSTEAD   2
// int_type: which interrupt type to attach to
#define INT_TYPE_AD       0x01
#define INT_TYPE_DA       0x02
#define INT_TYPE_DIOIN    0x04
#define INT_TYPE_USER     0x08
#define INT_TYPE_COUNTER  0x10
#define INT_TYPE_DIOOUT   0x20
#define INT_TYPE_OPTO     0x40
```

Structure Members

Name	Description
func	Pointer to user interrupt function of type DSCUserInterruptFunction
int_mode	Selects the execution time for the user interrupt function. Select from the choices in the list above. There is no "before" option.
int_type	Selects the interrupt type to attach this function to (e.g. A/D interrupts, D/A interrupts, DIO interrupts, etc.)

DscVoltageToADCode

Utility function for converting voltage to AD units based on the AD settings used.

Function Definition

BYTE DscVoltageToADCode(DSCB board, DSCADSETTINGS adsettings, DFLOAT voltage, DSCADSAMPLE *adsample);

Function Parameters

Name	Description
board	The board handle
adsettings	AD settings to be used in unit conversion
voltage	The voltage to be converted
adsample	Pointer to the variable that will hold the converted units

Return Value

Error code or 0.

Usage Example

```
ERRPARAMS errparams;
DSCADSETTINGS adsettings;
DSCB dscb;
DSCADSAMPLE adsample;
DFLOAT voltage;
BYTE result;

/* Board initialization code omitted */

voltage = 2.0;

if ((result = dscVoltageToADCode(dscb, adsettings, voltage, &adsample)) != DE_NONE)
```

```
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscVoltageToADCode failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}

printf("Voltage: %5.3lfV, AD code: %d\n", voltage, adsample);
```

DscVoltageToDACode

Utility function for converting voltage to DA units based on the DA settings used.

Function Definition

BYTE dscVoltageToDACode(DSCB board, DSCDASETTINGS dasettings, DFLOAT voltage, DSCDACODE *dacode);

Function Parameters

Name	Description
board	The board handle
dasettings	DA settings to be used in unit conversion
voltage	The voltage to be converted
dacode	Pointer to the variable that will hold the converted units

Return Value

Error code or 0.

Usage Example

```
ERRPARAMS errparams;
DSCDASETTINGS adsettings;
DSCB dscb;
DSCDACODE dacode;
DFLOAT voltage;
BYTE result;

/* Board initialization code omitted */

voltage = 2.0;

dasettings.polarity = BIPOLAR;
dasettings.load_cal = TRUE;
dasettings.range = 10.0;

if ((result = dscVoltageToDACode(dscb, dasettings, voltage, &dacode)) != DE_NONE)
{
   dscGetLastError(&errparams);
   fprintf(stderr, "dscVoltageToDACode failed: %s (%s)\n",
         dscGetErrorString(result), errparams.errstring);
   return result;
}

printf("Voltage: %5.3lfV, DA code: %d\n", voltage, adsample);
```

DSCWATCHDOG

Structure Definition
typedef struct {
  WORD wd1;
  BYTE wd2;
  BYTE options;

} DSCWATCHDOG;

Structure Members

Name	Description
WD1	16-bit timer value (runs at ~32KHz - max 2 seconds)
WD2	8-bit timer value (runs at ~32KHz - max 7.2ms)
options	mask PROM_WD_TRIGGER_SMI PROM_WD_TRIGGER_NMI HERC_WD_TRIGGER_SMI HERC_WD_TRIGGER_NMI ATHENA_WD_TRIGGER_SMI

dscWatchdogEnable
This function loads the watchdog counters to the specified values and configures the general behavior of watchdog counter 1, and arms the watchdog timer circuit.

Function Definition
BYTE dscWatchdogEnable (DSCB board, WORD wd1, BYTE wd2, SDWORD options);

Function Parameters

Name	Description
board	The board handle
wd1	Counter value for first watchdog timer
wd2	Counter value for second watchdog timer
options	A bitwise OR of watchdog options. See the list below.

Return Value
Error code or 0.

Watchdog Options
Each CPU has its own watchdog timer option. Below are the options for the different CPUs
Prometheus watchdog timer option

Name	Description
PROM_WD_TRIGGER_SCI	1 = Watchdog counter 1 will generate SCI on time-out 0 = No SCI occurs.
PROM_WD_TRIGGER_NMI	1 = Watchdog counter 1 will generate NMI on time-out 0 = No NMI occurs.
PROM_WD_TRIGGER_SMI	1 = Watchdog counter 1 will generate SMI on time-out 0 = No SMI occurs.
PROM_WD_TRIGGER_RESET	1 = Watchdog counter 1 will generate a hardware reset immediately on time-out. In this case WD2 serves no purpose, since its function is to delay the assertion of hardware reset from the timeout of WD1. However a valid

	load value for WD2 must still be supplied to the function. 0 = no hardware reset occurs now. Hardware reset will still be generated upon timeout of WD2.
PROM_WD_WDI_ASSERT_FALLING_EDGE	1 = WDI will retrigger the watchdog timer on a falling edge 0 = WDI will retrigger on a rising edge.
PROM_WD_WDO_TRIGGERED_EARLY	1 = output on WDO will be generated one clock cycle before counter 1 times out. In this case, WDO can be used to retrigger WDI by wiring the two signals together. This causes a bypass condition that prevents the watchdog timer from ever timing out. 0 = Output on WDO will be generated when watchdog counter 1 times out.
PROM_WD_ENABLE_WDI_ASSERTION	1 = hardware trigger input WDI will retrigger watchdog timer circuit and reload counter 1. Software retrigger command may also be used. 0 = input signal WDI is disabled; only software retrigger command may be used.

Athena watchdog timer option

Name	Description
ATHENA_WD_WDI_ASSERT_RISING_EDGE	1 = Watchdog will retrigger on a rising edge 0 = retrigger on falling edge.
ATHENA_WD_TRIGGER_SMI	1 = Watchdog counter 1 will generate SMI on time-out 0 = No SMI occurs.
ATHENA_WD_ENABLE_WDO	1 = Send an external pulse on WDO before trigger 0 = No pulse before trigger.
ATHENA_WD_ENABLED_WDI	1 = Send an external pulse on WDI before trigger 0 = No pulse before trigger.

Hercules watchdog timer option

Name	Description
HERC_WD_TRIGGER_NMI	1 = Watchdog counter 1 will generate NMI on time-out 0 = No NMI occurs.
HERC_WD_TRIGGER_RESET	1 = Watchdog counter WD1 will generate a hardware reset immediately on time-out. In this case WD2 serves no purpose, since its function is to delay the assertion of hardware reset from the timeout of WD1. However a valid load value for WD2 must still be supplied to the function. 0 = no hardware reset occurs now. Hardware reset will still be generated upon timeout of WD2.
HERC_WD_WDI_ASSERT_FALLING_EDGE	1 = WDI will retrigger the watchdog timer on a falling edge 0 = WDI will retrigger on a rising edge.
HERC_WD_WDO_TRIGGERED_EARLY	1 = output on WDO will be generated one clock cycle before counter 1 times out. In this case, WDO can be used to retrigger WDI by wiring the two signals together. This causes a bypass condition that prevents the watchdog timer from ever timing out. 0 = Output on WDO will be generated when watchdog counter 1 times out.
HERC_WD_ENABLE_WDI_ASSERTION	1 = hardware trigger input WDI will retrigger watchdog timer circuit and reload counter 1. Software retrigger command may also be used. 0 = input signal WDI is disabled; only software retrigger command may be used.

Helios Watchdog options and function usage.

The parameters to the function are used in Helios board as below.

• WD1 = Timeout settings for Watchdog 1.

The valid values are 0 through 9. The table below shows the mapping for the value supplied in the parameter wd1 and the actual time set in the watchdog timer.

WD1 Value	Watchdog Time
0	DISABLED
1	1 second
2	2 seconds
3	4 seconds
4	8 seconds
5	16 seconds
6	32 seconds
7	64 seconds
8	128 seconds
9	256 seconds
10	512 seconds

The above table describes the time taken by the watchdog timer to take generate a watchdog timer event and cause the system to reset (default setting) when set to a particular value in the WD1 parameter.

• WD2 = Timeout settings for Watchdog 2.

Similarly to setup watchdog 2, the parameter WD2 should be set to a similar value. The table representing WD2 parameter value and corresponding watchdog timer 2 timeout is as below.

WD2 Value	Watchdog Time
0	DISABLED
1	1 second
2	2 seconds
3	4 seconds
4	8 seconds
5	16 seconds
6	32 seconds
7	64 seconds
8	128 seconds
9	256 seconds
10	512 seconds

Options = Settable parameters for watchdog behavior. The options field can be any value as defined in the table below. It is recommended to set this field to 0xD0 when using the functionality. The same options apply to both the watchdog timers.

Options Parameter	Watchdog Action
0x00	Reserved
0x10	IRQ3
0x20	IRQ4
0x30	IRQ5
0x40	IRQ6
0x50	IRQ7
0x60	IRQ9
0x70	IRQ10
0x80	N/A
0x90	IRQ12
0xA0	IRQ14
0xB0	IRQ15
0xC0	NMI
0xD0	System Reset (Default , Recommended by DSC.)
0xE0-0xF0	Reserved

dscWatchdogDisable

This function disables the watchdog timer circuit on the specified board. A previously enabled but subsequently disabled watchdog timer may be re-enabled using the existing configuration by a subsequent call to dscWatchdogTrigger().

Function Definition
BYTE dscWatchdogDisable(DSCB board);

Function Parameters

Name	Description
board	The board handle

Return Value
Error code or 0.

dscWatchdogTrigger

This function retriggers the watchdog timer circuit, causing both counters WD1 and WD2 to be reloaded with their initial values. After a call to dscWatchdogTrigger(), the application has the amount of time specified in WD1 before it must call the function again.

If the watchdog timer has been disabled with a call to dscWatchdogDisable(), than this function will reenable the watchdog timer using the existing watchdog configuration from the last dscWatchdogEnable() call.

This function must not be called before dscWatchdogEnable() is called for the first time, or else undefined results will occur.

In a typical application, dscWatchdogTrigger() is used in a continuous loop to keep retriggering the watchdog timer circuit as evidence that the system is running properly. If the loop ever crashes, or otherwise fails to call dscWatchdogTrigger() in time, the system will reset. The system is configured to automatically restart the application software when it reboots. This way the amount of down time due to a software failure is minimized.

Function Definition
BYTE dscWatchdogTrigger(DSCB board);

Function Parameters

Name	Description
board	The board handle

Return Value
Error code or 0.

DscWGBufferSet

Configures D/A wave form generator D/A output. Sets up the code for D/A output. Gives users control over each output code, the D/A channel intended, and whether it is a simultaneous update or not.
Configures D/A wave form generator D/A output. Sets up the code for D/A output. Gives users control over each output code, the D/A channel intended, and whether it is a simultaneous update or not.

Function Definition
BYTE dscWGBufferSet (DSCB board, DWORD address, DSCDACODE value,DWORD channel, BOOL simul)

Function Parameters

Name	Description
board	The handle of the board to operate on
address	Address in the buffer to set. This is any where from 0 to (buffer size - 1), i.e. if buffer size is 64 this is 0 to 63
value	D/A code to output
channel	D/A channel to output from
simul	TRUE = enable simultaneous update, FALSE = no simultaneous update

Return Value
Error code or 0.

DscWGCommand

Commands to start, stop, reset, and trigger the D/A wave form generator

Function Definition
#define WG_CMD_START 0x01
#define WG_CMD_PAUSE 0x02
#define WG_CMD_RESET 0x04
#define WG_CMD_INC 0x08
BYTE dscWGCommand (DSCB board, DWORD cmd)

Function Parameters

Name	Description
board	The handle of the board to operate on
cmd	Command to stop, start, reset, and trigger D/A wave form generator. Defines can be found in DSCUD.H as WG_CMD_XXX

Return Value
Error code or 0.

DSCWGCONFIG

Structure containing D/A wave form parameters for function DscWGConfigSet().

Structure Definition

```
#define WG_SRC_MANUAL   0
#define WG_SRC_CTR0     1
#define WG_SRC_CTR12    2
#define WG_SRC_EXT      3
#define WG_CH_PER_FRAME_1          0
#define WG_CH_PER_FRAME_2          1
#define WG_CH_PER_FRAME_4          2
typedef struct
{
  DWORD depth;
  DWORD ch_per_frame;
  DWORD source;
} DSCWGCONFIG;
```

Structure Members

Name	Description
depth	WG total D/A sample size (absolute, 64, 128, 256, 512, and 1024)
ch_per_frame	Number of values to output from the buffer per frame/trigger, check define WG_CH_PER_FRAME_XXX
source	WG trigger source, check define WG_SRC_XXX

DscWGConfigSet

Configures D/A wave form generator. Sets the depth, number of output per trigger, and trigger source.
Function Definition
BYTE dscWGConfigSet (DSCB board, DSCWGCONFIG* config)

Function Parameters

Name	Description
board	The handle of the board to operate on
config	Structure containing configuration information for D/A wave form generator

Return Value
Error code or 0.

DWORD

Unsigned long

Emerald-MM-8

The following information applies to all versions of Emerald-MM-8.

Digital I/O
Max Ports: 1 8-bit bi-directional port with programmable direction bit by bit
Digital I/O port requires direction to be set with dscDIOSetConfig) before use.

The configuration byte has the following format:

Bit No.	7	6	5	4	3	2	1	0
Name	DIR 7	DIR 6	DIR 5	DIR 4	DIR 3	DIR 2	DIR 1	DIR 0

Dir 7 - 0 0 = input, 1 = output

Serial Ports
8 serial ports with RS-232, RS-422, and/or RS-485 capability, depending on the model.
Serial ports have jumper-configured protocols, base addresses and IRQ levels. See the board's user manual for information on serial port configuration.

Serial ports on Emerald-MM-8 are not supported by Universal Driver.

Emerald-MM-8 Universal Driver Functions

- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()

Emerald-MM-DIO

Digital I/O
Max Ports: 6 8-bit bi-directional ports with programmable direction bit by bit and inverting logic
The first 3 ports have edge detection capability with polarity and enable selected individually for each bit. Edge detection can be utilized by running user interrupts. See functions dscEMMDIOSetState and dscEMMDIOGetState.

Digital I/O lines on Emerald-MM-DIO do not require configuration. Writing a 0 to any bit drives the pin high and also enables it as an input. Writing a 1 to a bit drives the pin low and forces it to an output.
A high input will read back as a 0, and a low input will readback as a 1.

Interrupts
1 with jumper-selected IRQ level

Serial Ports
4 RS-232 ports with jumper-configured base addresses and IRQ levels.
Serial ports on Emerald-MM-DIO are not supported by Universal Driver.

Emerald-MM-DIO Universal Driver Functions

- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscEMMDIOGetState()
- dscEMMDIOResetInt()
- dscEMMDIOSetState()

- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()

ERRPARAMS

Structure containing error code information resulting from a Universal Driver function call. An error code is generated as the return value by each driver function..

Structure Definition

```
typedef struct {
   BYTE errcode;
   char* errstring;
} ERRPARAMS;
```

Structure Members

Name	Description
errcode	The error code representing the particular error
errstring	The string description corresponding to the error code in ErrCode

FALSE

(BOOL)0

FLOAT

float

Garnet-MM

Digital I/O
GMM-24: 3 8-bit bidirectional ports (1 82C55 chip)
GMM-48: 6 8-bit bidirectional ports (2 82C55 chips)
Digital I/O ports on Garnet-MM require direction to be set with dscDIOSetConfig() before use. One configuration byte is required for each 82C55 chip.
All DIO lines power-up in input mode and have readback capability.
DIO lines on this board do not have pull-up resistors.

Interrupts
GMM-24: 1; source is digital I/O bit C0 of the 82C55 chip
GMM-48: 2; sources are digital I/O bit C0 of each 82C55 chip Interrupt levels are configured with jumpers.

Garnet-MM Universal Driver Functions

- dscDIOSetBit()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetConfig()
- dscSetUserInterruptFunction()
- DscClearUserInterruptFunction()
- dscUserInt()

GPIO-MM-11

Overview
GPIO-MM-11 and GPIO-MM-12 boards provide 48 lines of DIO and 2 9513 based 16 bit counter timers.
Board Initialization

To use the GPIO-MM-11/12 board in an appllication using the UD, the application needs to perform a dscInitBoard on the hardware feature that is desired in the application.

The GPIO-MM-11 has two separate base address ranges for the two distinct hardware features. The DIO section of the hardware is controlled by a base address which is different from the base address for the register set to control the 9513 based counter timer circuitry.

Thus the application needs to perform a dscInitBoard twice if both the DIO and the counter timer functionality is desired.

This is shown in an example below...
To use DIO functionality, use dscInitBoard (DSC_GPIO11_DIO , &dsccb, &board);

To use the 9513 counter/timer functionality, use dscInitBoard (DSC_GPIO11_9513, &dsccb1, &board1);
Digital I/O

Max Ports: 6 8-bit bi-directional ports with programmable direction bit by bit and inverting logic
Digital I/O lines on GPIO-MM-11 do not require configuration. Writing a 0 to any bit drives the pin high and also enables it as an input. Writing a 1 to a bit drives the pin low and forces it to an output.
A high input will read back as a 0, and a low input will readback as a 1.
Interrupts

2 with jumper-selected IRQ levels

Fixed Ports
8 fixed input ports, 8 fixed output ports

Counter/Timers
2 9513 16-bit counter/timers with programmable sources and cascading

Refer to the GPIO-MM-11 user manual for detailed information on the counter and interrupt circuit features and configuration. Also refer to the user interrupt function descriptions in this manual for information on how to configure the counters and interrupts for your application.

GPIO-MM-11 Universal Driver Functions

- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- DscGetEEPROM()
- DscSetEEPROM()
- dscGetStatus()
- dscCancelOp()
- dscSetUserInterruptFunction()
- DscInterruptControl()
- dscUserInt()
- Dsc9513Reset()
- Dsc9513SetMMR()
- Dsc9513SetCMR()
- Dsc9513CounterControl()
- Dsc9513SingleCounterControl()
- Dsc9513SetLoadRegister()
- Dsc9513SetHoldRegister()
- Dsc9513ReadHoldRegister()
- Dsc9513SpecialCounterFunction()
- Dsc9513MeasureFrequency()
- Dsc9513MeasurePeriod()
- Dsc9513PulseWidthModulation()

GPIO-MM-21

Board Initialization
To use the GPIO-MM-21 board in UD, the dscInitBoard function should use the board macro DSC_GPIO21. This is shown in the example below...
dscInitBoard (DSC_GPIO21 , &dsccb, &board);

Digital I/O
Max Ports: 12 8-bit bi-directional ports with programmable direction bit by bit and inverting logic thus providing 96 DIO lines.
Digital I/O lines on GPIO-MM-21 do not require configuration. Writing a 0 to any bit drives the pin high and also enables it as an input. Writing a 1 to a bit drives the pin low and forces it to an output.

A high input will read back as a 0, and a low input will readback as a 1.

GPIO-MM-21 Universal Driver Functions

- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetConfig()

Helios

Overview

Helios's Data Acquistion circuitry is similar to that on Athena-II but some of the registers for controlling the behavior of the circuit are different on Helios from AthenaII.

The Universal Driver version 6.01 handles the differences. Below are some of the major differences in the Helios board.

1. Watchdog Timer support.
2. Additional DIO ports available in Helios.
3. DA modes are software programmable.
4. DA channels can be individually programmed for unique range settings.

In case of Helios board, all the AD and DA settings are only software programmable. Unlike the previous boards, jumpers cannot be used to perform the AD / DA mode and polarity settings.

Board Initialization

To use the Helios board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_HELIOS. This is shown in the example below...

The base address should be 0x280 (or whatever is set in the BIOS of the board) and the default IRQ to use is IRQ 5.

```
dscInitBoard ( DSC_HELIOS, &dsccb, &board );
```

It is highly recommended that the dsccb structure be memset to 0 in the application before calling the dscInitBoard API.

```
memset ( &dsccb , 0 , sizeof ( DSCCB ) );
```

The Helios board also uses the DAC_Config element available in the DSCCB structure to force 16 bit DAC operation. To use the 16 bit DAC mode of the Helios board, the DAC_Config element of the DSCCB structure must be set to 1 before calling the dscInitBoard API.

```
dsccb.DAC_Config = 1 ; // for 16 bit mode DAC.
```

Currently Helios boards have only 12 bit DAC available and thus it is advisable to set the DAC_Config to 0.

NOTE: For making the interrupt based operations work on Helios board, the IRQ5 MUST be set to Reserved in the BIOS.

Analog Input
Max Input Channels: 16 (single-ended) or 8 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltrage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, or ±1.25V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate:

- Software Command: approx. 20,000 samples per second, depending on code and operating system
- Interrupt Routine w/FIFO: 100,000 samples per second

FIFO: 48 samples with programmable threshold in standard mode. Upto 2048 in Enhanced FIFO mode. In the enhanced FIFO mode, it is recommended to set the FIFO threshold to 1024 samples by setting the FIFO threshold register Base + 5 with a value of 0x80.

The FIFO threshold register at location Base + 5 of the Helios board, contains the threshold for the enhanced mode. The threshold is an 11 bit number as the maximum FIFO depth available in Helios is 2048. Thus to set the threshold value of 11 bit in an 8 bit register, the threshold value is programmed as 8 sample blocks. The universal driver takes care of the conversion.

For example, to set a FIFO threshold of 1024, the Universal Driver will write a value of 0x80 to the register.

This board supports the following programmable input ranges and resolutions:

A/D Ranges				
Polarity	G1	G0	Input Range	Resolution (1 LSB)
Bipolar	0	0	±10V	305µV
Bipolar	0	1	±5V	153µV
Bipolar	1	0	±2.5V	76µV
Bipolar	1	1	±1.25V	38µV
Unipolar	0	0	0 - 10V	153µV
Unipolar	0	1	0 - 5V	76µV
Unipolar	1	0	0 - 2.5V	38µV
Unipolar	1	1	0 - 1.25V	19µV

Analog Output
Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Bipolar Output Ranges: ±10V
Unipolar Output Ranges: 0-10V

The Helios board provides the following programmable DA modes. The DA modes can be set using a new API function called dscDASetSettings ().

D/A Ranges			
G1	G0	DAPOL	Description
0	0	0	5V Span (0 to +5V Unipolar)
0	0	1	5V Span (-2.5V to +2.5V Bipolar)
0	1	0	10V Span (0 10 +10V Unipolar)
0	1	1	10V Span (-5V to +5V Bipolar)
1	0	1	NOT USED
1	0	1	20V Span (-10V to +10V Bipolar)
1	1	1	D/A converter shut down
1	1	0	NOT USED

Digital I/O

Helios supports 5 DIO ports. All the DIO ports are bi-directional, programmable in 8-bit groups, TTL-compatible and are similar to 82C55 Digital I/O ports on AthenaII. All the ports require the port direction to be set with dscDIOSetConfig before use. The value of 0x00 is used for the port to be an output port while 0xFF configures the port as an input port.

The Helios board supports 3 DIO ports which are 82C55 based ports provided by the on-board FPGA. There are 2 more DIO ports made available to the user which are directly provided by the CPU.

All DIO lines power-up in input mode and have readback capability.

Helios Universal Driver Functions

- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- DscDASetSettings()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- dscSetUserInterruptFunction()
- DscClearUserInterruptFunction()
- dscUserInt()
- dscWatchdogDisable()
- dscWatchdogEnable()
- dscWatchdogTrigger()

Helios Universal Driver Watchdog Usage

Example Code: The following is an example program in C to use the watchdog functionality on Helios board.

```
#include "dscud.h"

DSCB dscb;
DSCCB dsccb;

int main (void) {
    //==========================================================================
    // I. DRIVER INITIALIZATION
    //
    //   Initializes the Universal Driver library.
    //
    //==========================================================================
    if( dscInit( DSC_VERSION ) != DE_NONE )
    {
        dscGetLastError(&errorParams);
        fprintf( stderr, "dscInit error: %s %s\n", dscGetErrorString
          (errorParams.ErrCode), errorParams.errstring );
        return 0;
    }
    //==========================================================================
    // II. BOARD INITIALIZATION
    //
    //    Initialize the HELIOS board. This function passes the various
    //    hardware parameters to the driver and resets the hardware.
    //
    //==========================================================================
    printf( "\nHELIOS BOARD INITIALIZATION:\n" );
    dsccb.base_address = 0x280;// match this to the BIOS setting
    dsccb.int_level = 5;
    if(dscInitBoard(DSC_HELIOS, &dsccb, &dscb)!= DE_NONE)
    {
        dscGetLastError(&errorParams);
        fprintf( stderr, "dscInitBoard error: %s %s\n",
        dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
        return 0;
    }

 // SET WATCHDOG TIMER TIMEOUT value to 64 Seconds
    dscWatchdogEnable ( dscb , 0, 7 , 0xD0 ) ; // Action = Reset board
    while ( 1 )
    {
        // other application specific actions…
        dscWatchdogTrigger(dscb) ;
    }
    return DE_NONE ;
}
```

If the dscWatchdogTrigger (dscb) call is removed from the above example code, the Watchdog timer will reset the system after 64 seconds.

Hercules-EBX

Analog Input
Max Input Channels: 32 (single-ended) or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +2767 for all voltrage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, or ±1.25V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL
Signal Maximum Conversion Rate:
Software Command: approx. 20,000 samples per second, depending on code and operating system
Interrupt Routine w/FIFO: 250,000 samples per second
FIFO: 2048 samples with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±10V	153µV
1	0	0	0	1	±5V	76µV
2	0	0	1	0	±2.5V	38µV
3	0	0	1	1	±1.25V	19µV
4	0	1	0	0	0 - 10V	153µV
5	0	1	0	1	0 - 5V	76µV
6	0	1	1	0	0 - 2.5V	38µV
7	0	1	1	1	0 - 1.25V	19µV

Analog Output
Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Bipolar Output Ranges: ?10V
Unipolar Output Ranges: 0-10V

Digital I/O
Max Ports: 5, bi-directional, programmable in 8-bit groups, TTL-compatible Digital I/O ports on Hercules require direction to be set with dscDIOSetConfig() before use.
All DIO lines power-up in input mode and have readback capability. All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.

Hercules-EBX Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()

- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()
- DscPWMLoad()
- DscPWMConfig()
- DscPWMClear()
- DscPWMFunction()
- dscWatchdogDisable()
- dscWatchdogEnable()
- dscWatchdogTrigger()

Hercules-II

Board Initialization
To use the Hercules-II board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_HERC. This is shown in the example below...
dscInitBoard (DSC_HERC , &dsccb, &board);

Analog Input
Max Input Channels: 32 (single-ended) or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +2767 for all voltrage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, or ±1.25V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL
Signal Maximum Conversion Rate:

- *Software Command:* approx. 20,000 samples per second, depending on code and operating system
- *Interrupt Routine w/FIFO:* 250,000 samples per second

FIFO: 2048 samples with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±10V	153µV
1	0	0	0	1	±5V	76µV
2	0	0	1	0	±2.5V	38µV
3	0	0	1	1	±1.25V	19µV
4	0	1	0	0	0 - 10V	153µV
5	0	1	0	1	0 - 5V	76µV
6	0	1	1	0	0 - 2.5V	38µV
7	0	1	1	1	0 - 1.25V	19µV

Analog Output
Max Output Channels: 4
D/A Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Bipolar Output Ranges: ?10V
Unipolar Output Ranges: 0-10V

Digital I/O
Max Ports: 5, bi-directional, programmable in 8-bit groups, TTL-compatible Digital I/O ports on Hercules require direction to be set with dscDIOSetConfig() before use.
All DIO lines power-up in input mode and have readback capability. All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.

Hercules-II Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()

- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()
- DscPWMLoad()
- DscPWMConfig()
- DscPWMClear()
- DscPWMFunction()
- dscWatchdogDisable()
- dscWatchdogEnable()
- dscWatchdogTrigger()

IR104

Board Initialization

To use the IR-104 board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_IR104. This is shown in the example below...

```
dscInitBoard ( DSC_IR104 , &dsccb, &board );
```

Digital I/O

Max Input Ports: 20 lines (20 bits), optoisolated

Max Output Ports: 20 lines (20 bits), type SPST (form A) relays

A 1 in each bit turns that relay on, and a 0 turns it off. All relays power up in the off position. The output ports have read back capability. The inputs and outputs are fixed direction and don't need configuration prior to use.

Interrupts

Not supported on this board.

IR104 Universal Driver Functions

- DscIR104SetRelay()
- DscIR104ClearRelay()
- DscIR104RelayInput()
- DscIR104OptoInput()

LONG

signed long

Mercator

Overview

Mercator is a board with 3 DIO ports which are compatible with 82C55 based DIO ports.

Board Initialization

To use the Mercator board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_MRC. This is shown in the example below...
dscInitBoard (DSC_MRC , &dsccb, &board);

Digital I/O
Max Ports: 3 8-bit bi-directional ports with programmable direction bit by bit and inverting logic
Digital I/O lines on GPIO-MM-11 do not require configuration. Writing a 0 to any bit drives the pin high and also enables it as an input. Writing a 1 to a bit drives the pin low and forces it to an output.

A high input will read back as a 0, and a low input will readback as a 1.

Mercator Universal Driver Functions

- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetConfig()

Neptune

Overview
Neptune is an ETX based base board. Neptune board has universal ETX connectors for connecting ETX based CPU boards. The Neptune LX800 contains the base board with the Kontron(R) LX-800 CPU board. Similarly Neptune PM14 board contains the base board with the Kontron(R) Pm-14 CPU board.

Neptune base board provides the precision ADC,DAC and DIO features similar to those of DMM32X-AT. Neptune board does not support the Auto Auto Calibration feature.

Along with the analog features, the Neptune base board provides an additional OPTO DIO port and 4 RS-232 ports. NOTE: The Universal Driver does not provide any RS-232 communication drivers. The RS-232 ports specified above can be programmed using easy port read/write. The Universal Driver does not provide any APIs to program the base address and IRQs for the said RS-232 ports.

Board Initialization
To use the Neptune board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_NEPTUNE. This is shown in the example below...
The default base address should be 0x300 and the default IRQ to use is IRQ 7.
dscInitBoard (DSC_NEPTUNE , &dsccb, &board);

Analog Input
Max Input Channels: 32 (single-ended), 24 (16/8 single-ended/differential), or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1.25V, or ±0.625V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): system-dependent, up to 100,000 per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 250,000 samples per second
FIFO: 1024 samples (2048 upon request) with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	153µV
1	0	0	0	1	±2.5V	76µV
2	0	0	1	0	±1.25V	38µV
3	0	0	1	1	±0.625V	19µV
4	0	1	0	0	Invalid Setting	-
5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 10V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	0 - 1.25V	19µV

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output
Max Output Channels: 4 D/A
Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Output Ranges (Bipolar): ±5V, ±10V, or programmable
Output Ranges (Unipolar): 0-5V, 0-10V, or programmable

Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) similar to 82C55
Digital I/O ports on Diamond-MM-32-AT require direction to be set with DscDIOSetConfig () before use.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 4.7K Ohm pull resistors that can be configured for all pull-up or all pull-down with a jumper.
OPTO I/O

Neptune board provides an OPTO isolated DIO port. The Universal Driver provides access to the said port using the regular DscDIOOutputByte() and DscDIOInputByte() APIs. The OPTO port does not need any configuration and thus the dscDIOSetConfig API is NOT used for the OPTO port.

To use the OPTO port, the user program MUST use the port index as 4 (regular DIO ports MAX out at 3) for using in dscDIOOutputByte and dscDIOInputByte.

Universal Driver API Notes
For these functions Neptune has the following restrictions

DscADSampleInt, DscADScanInt
1. FIFO threshold (dscaioint.fifo_depth) must be a multiple of the number of channels

2. Num_conversions (dscaioint.num_conversions) must be a multiple of fifo threshold
3. FIFO threshold (dscaioint.fifo_depth) must be an even number between 0 to 1022 (or 0 to 2046 for 2048 size FIFO)

Neptune Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()
- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()
- DscEnhancedFeaturesEnble()
- DscWGCommand()
- DscWGConfigSet()
- DscWGBufferSet()

Onyx-MM

Board Initialization
To use the Onyx-MM board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_OMM. This is shown in the example below...
dscInitBoard (DSC_OMM , &dsccb, &board);

Digital I/O
6 8-bit bidirectional ports (2 82C55 chips)

Digital I/O ports on Onyx-MM require direction to be set with dscDIOSetConfig (on page 118) before use. One configuration byte is required for each 82C55 chip.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 10K Ohm pull-up resistors.

Interrupts
3 interrupts with programmable sources and jumper-configured IRQ levels
Counter/Timers
3 16-bit counter/timers with programmable sources and cascading
Refer to the Onyx-MM user manual for detailed information on the counter and interrupt circuit features and configuration. Also refer to the user interrupt function descriptions in this manual for information on how to configure the counters and interrupts for your application.

Onyx-MM Universal Driver Functions

- DscClearUserInterruptFunction()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- dscSetUserInterruptFunction()
- dscUserInt()

Onyx-MM-DIO

Board Initialization
To use the Onyx-MM-DIO board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_OMMDIO. This is shown in the example below...
dscInitBoard (DSC_OMMDIO , &dsccb, &board);

Digital I/O
6 8-bit bidirectional ports (2 82C55 chips)
Digital I/O ports on Onyx-MM-DIO require direction to be set with dscDIOSetConfig() before use. One configuration byte is required for each 82C55 chip.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 10K Ohm pull-up resistors. Interrupts Not supported on this board.

Onyx-MM-DIO Universal Driver Functions

- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()

Opal-MM

Board Initialization

To use the Opal-MM board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_OPMM. This is shown in the example below...

dscInitBoard (DSC_OPMM , &dsccb, &board);

Digital I/O

Max Input Ports: 8 lines (1 byte or 8 bits), optoisolated, inverting logic (high input = 0, low input = 1)

Max Output Ports: 8 lines (1 byte or 8 bits), type SPDT (form C) relays

A 1 in each bit turns that relay on, and a 0 turns it off. All relays power up in the off (0) position.

The output ports do not have readback capability.

The inputs and outputs are fixed direction and do not need configuration prior to use.

Interrupts

Not supported on this board.

Opal-MM Universal Driver Functions

- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()

Pearl-MM

Board Initialization
To use the Pearl-MM board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_PMM. This is shown in the example below...
dscInitBoard (DSC_PMM , &dsccb, &board);

Digital I/O
Max Ports: 16 lines (2 bytes or 16 bits), type SPDT (form C) relays
A 1 in each bit turns that relay on, and a 0 turns it off. All relays power up in the off (0) position.
The output ports do not have readback capability.
The outputs are fixed direction and do not require configuration prior to use.

Interrupts
Not supported on this board.

Pearl-MM Universal Driver Functions

- dscDIOClearBit()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()

Poseidon

Overview
Poseidon's DAQ circuit is compatible with DMM32X-AT.

Board Initialization
To use the Poseidon board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_PSD. This is shown in the example below...
dscInitBoard (DSC_PSD , &dsccb, &board);

Analog Input
Max Input Channels: 32 (single-ended), 24 (16/8 single-ended/differential), or 16 (differential)
A/D Resolution: 16 bits (1/65536 of full-scale)
Data range: -32768 to +32767 for all voltage ranges
Input Ranges (Bipolar): ±10V, ±5V, ±2.5V, ±1.25V, or ±0.625V
Input Ranges (Unipolar): 0-10V, 0-5V, 0-2.5V, or 0-1.25V
Supported Conversion Triggers: Software, Internal Clock, or External TTL Signal
Maximum Conversion Rate (Software Command): system-dependent, up to 100,000 per second
Maximum Conversion Rate (Interrupt Routine w/FIFO): 250,000 samples per second
FIFO: 1024 samples (2048 upon request) with programmable threshold

This board supports the following programmable input ranges and resolutions:

A/D Ranges						
Code	Range	ADBU	G1	G0	Input Range	Resolution (1 LSB)
0	0	0	0	0	±5V	153µV
1	0	0	0	1	±2.5V	76µV
2	0	0	1	0	±1.25V	38µV
3	0	0	1	1	±0.625V	19µV
4	0	1	0	0	Invalid Setting	-

5	0	1	0	1	Invalid Setting	-
6	0	1	1	0	Invalid Setting	-
7	0	1	1	1	Invalid Setting	-
8	1	0	0	0	±10V	305µV
9	1	0	0	1	±5V	153µV
10	1	0	1	0	±2.5V	76µV
11	1	0	1	1	±1.25V	38µV
12	1	1	0	0	0 - 10V	153µV
13	1	1	0	1	0 - 5V	76µV
14	1	1	1	0	0 - 2.5V	38µV
15	1	1	1	1	0 - 1.25V	19µV

NOTE: Ranges 9 through 11 are identical to ranges 0 through 2.

Analog Output
Max Output Channels: 4 D/A
Resolution: 12 bits (1/4096 of full-scale)
Data range: 0 to 4095 for all voltage ranges
Output Ranges (Bipolar): ±5V, ±10V, or programmable
Output Ranges (Unipolar): 0-5V, 0-10V, or programmable
Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) similar to 82C55
Digital I/O ports on Posedion require direction to be set with DscDIOSetConfig() before use.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have pull resistors that can be configured for all pull-up or all pull-down with a jumper on JP9.

Universal Driver API Notes
For these functions Poseidon has the following restrictions

DscADSampleInt, DscADScanInt
1. FIFO threshold (dscaioint.fifo_depth) must be a multiple of the number of channels
2. Num_conversions (dscaioint.num_conversions) must be a multiple of fifo threshold
3. FIFO threshold (dscaioint.fifo_depth) must be an even number between 0 to 1022 (or 0 to 2046 for 2048 size FIFO)

Poseidon Universal Driver Functions

- dscADAutoCal()
- dscADCalVerify()
- dscADSample()
- dscADSampleInt ()
- dscADScan()
- dscADScanInt()
- dscADSetChannel()
- DscADSetSettings()
- dscCounterDirectSet()
- DscCounterRead()
- DscCounterSetRate()
- DscCounterSetRateSingle()
- dscDAAutoCal()
- dscDACalVerify()

- dscDAConvert()
- dscDAConvertScan()
- dscDAConvertScanInt()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()
- DscGetEEPROM()
- DscSetEEPROM()
- DscGetReferenceVoltages()
- DscSetReferenceVoltages()
- dscGetStatus()
- DscClearUserInterruptFunction()
- dscSetUserInterruptFunction()
- dscUserInt()
- DscEnhancedFeaturesEnble()
- DscAACCommand()
- DscAACGetStatus()
- DscWGCommand()
- DscWGConfigSet()
- DscWGBufferSet()

QMM Macros

```
/* QMM counter group */
 /* Group 1 means chip 1 and counters 1-5 */
 /* Group 2 means chip 2 and counters 6-10 */
 #define QMM_COUNTER_GROUP_1    1
 #define QMM_COUNTER_GROUP_2    2
 /* QMM fout and counter source selection */
 #define QMM_SOURCE_E1_TC_NM1   0
 #define QMM_SOURCE_SRC1        1
 #define QMM_SOURCE_SRC2        2
 #define QMM_SOURCE_SRC3    3
 #define QMM_SOURCE_SRC4    4
 #define QMM_SOURCE_SRC5    5
 #define QMM_SOURCE_GATE1 6
 #define QMM_SOURCE_GATE2 7
 #define QMM_SOURCE_GATE3 8
 #define QMM_SOURCE_GATE4 9
 #define QMM_SOURCE_GATE5 10
 #define QMM_SOURCE_F1_4MHZ         11
 #define QMM_SOURCE_F2_400KHZ       12
 #define QMM_SOURCE_F3_40KHZ        13
 #define QMM_SOURCE_F4_4KHZ         14
 #define QMM_SOURCE_F5_400HZ        15
 /* QMM time of day mode */
 #define QMM_TOD_DISABLED  0
```

```
#define QMM_TOD_DIVIDE_BY_5        1
#define QMM_TOD_DIVIDE_BY_6        2
#define QMM_TOD_DIVIDE_BY_10       3
/* QMM gating control */
#define QMM_NO_GATING      0
#define QMM_ACTIVE_HIGH_TC_NM1  1
#define QMM_ACTIVE_HIGH_LEVEL_GATE_NP1          2
#define QMM_ACTIVE_HIGH_LEVEL_GATE_NM1          3
#define QMM_ACTIVE_HIGH_LEVEL_GATE_N    4
#define QMM_ACTIVE_LOW_LEVEL_GATE_N     5
#define QMM_ACTIVE_HIGH_EDGE_GATE_N     6
#define QMM_ACTIVE_LOW_EDGE_GATE_N      7
/* QMM output control */
#define QMM_INACTIVE_OUTPUT_LOW         0
#define QMM_ACTIVE_HIGH_PULSE_ON_TC     1
#define QMM_TOGGLE_ON_TC2
#define QMM_INACTIVE_OUTPUT_HIGH        4
#define QMM_ACTIVE_LOW_PULSE_ON_TC      5
/* QMM counter actions */
#define QMM_ACTION_NONE  0
#define QMM_ACTION_ARM     1
#define QMM_ACTION_LOAD   2
#define QMM_ACTION_LOAD_AND_ARM         3
#define QMM_ACTION_DISARM_AND_SAVE      4
#define QMM_ACTION_SAVE    5
#define QMM_ACTION_DISARM          6
/* QMM special counter actions */
#define QMM_SPECIAL_CLEAR_TOGGLE_OUTPUT         0
#define QMM_SPECIAL_SET_TOGGLE_OUTPUT 1
#define QMM_SPECIAL_STEP_COUNTER          2
#define QMM_SPECIAL_PROGRAM_ALARM       3
/* QMM frequency intervals */
#define QMM_INTERVAL_1MS_1KHZ    0
#define QMM_INTERVAL_10MS_100HZ1
#define QMM_INTERVAL_100MS_10HZ2
#define QMM_INTERVAL_1S_1HZ        3
#define QMM_INTERVAL_10S_01HZ      4
```

Quartz-MM

Quartz-MM Universal Driver Functions

- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscQMMCounterControl()
- dscQMMMeasureFrequency()
- dscQMMMeasurePeriod()
- dscQMMPulseWidthModulation()
- dscQMMReadHoldRegister()

- dscQMMSetCMR()
- dscQMMSetHoldRegister()
- dscQMMSetLoadRegister()
- dscQMMSetMMR()
- dscQMMSingleCounterControl()
- dscQMMSpecialCounterFunction()

Raw Board

This board type was added to allow for using Univeral Driver direct I/O and interrupt functions without interacting with a Diamond Systems DAQ hardware.

In the case of direct I/O, the DscRegisterRead() and DscRegisterWrite() functions can be used to interact with arbitrary I/O ranges. In Windows this is only allowed if I/O permission has been granted by the operating system for those ranges. Calling DscInitBoard() with a board type of DSC_RAW and the address of that range of arbitrary I/O memory will grant your software access to that memory.

In the case of interrupts, setting a user interrupt function and calling dscUserInt() on a DSC_RAW board will register an interrupt handler on the requested IRQ. Your routine will be called whenever an interrupt occurs. This is useful for easily adding your own interrupt driven features.

Ruby-MM

Board Initialization
To use the Ruby-MM board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_RMM. This is shown in the example below...
dscInitBoard (DSC_RMM , &dsccb, &board);

Analog Input
Not supported by this board.

Analog Output
Max Output Channels (RMM-4-XT): 4
Max Output Channels (RMM-8-XT): ***8***
D/A Resolution: 12 bits (1/4096 of full-scale)
Data Range: 0 to 4095
Output Ranges (Bipolar): ±10V, ±5V, or ±2.5V (per each bank of 4 channels via jumper)
Output Ranges (Unipolar): 0-10V, 0-5V, or 0-2.5V (per each bank of 4 channels via jumper)
The 2.5V ranges are user-adjustable to any voltage between 0V and 2.5V with a potentiometer.

Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) based on 82C55
Digital I/O ports on Ruby-MM require direction to be set with DscDIOSetConfig () before use.
All DIO lines power-up in input mode and have readback capability.
All DIO lines have 10K Ohm pull-up resistors.

Ruby-MM Universal Driver Functions

- dscDAConvert()
- dscDAConvertScan()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()

- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()

Ruby-MM-416

Board Initialization
To use the Ruby-MM-416 board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_RMM416. This is shown in the example below...
dscInitBoard (DSC_RMM416 , &dsccb, &board);

Analog Input
Not supported by this board.

Analog Output
Max Output Channels: 4
D/A Resolution: 16 bits (1/65536 of full-scale)
Data Range: -32768 to +32767
Output Ranges (Bipolar): ±10V or ±5V (per channel via jumper)
Output Ranges (Unipolar): 0-10V (per channel via jumper)

Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) based on 82C55
Digital I/O ports on Ruby-MM-416 require direction to be set with DscDIOSetConfig() before use.

All DIO lines power-up in input mode and have readback capability.

All DIO lines have 10K Ohm pull-up resistors.

Ruby-MM-416 Universal Driver Functions
- dscDAConvert()
- dscDAConvertScan()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()

Ruby-MM-1612

Board Initialization
To use the Ruby-MM-1612 board in an appllication using the UD, the dscInitBoard function should use the board macro DSC_RMM1612. This is shown in the example below...
dscInitBoard (DSC_RMM1612 , &dsccb, &board);

Analog Input
Not supported by this board.

Analog Output
Max Output Channels: 16
D/A Resolution: 12 bits (1/4096 of full-scale)

Data Range: 0 to 4095
Output Ranges (Bipolar): ±10V, ±5V, or ±2.5V (per each bank of 8 channels via jumper)
Output Ranges (Unipolar): 0-10V, 0-5V, or 0-2.5V (per each bank of 8 channels via jumper)

Digital I/O
Max Ports: 3 (bi-directional, programmable in 8-bit groups, TTL-compatible) based on 82C55
Digital I/O ports on Ruby-MM-1612 require direction to be set with DscDIOSetConfig() before use.

All DIO lines power-up in input mode and have readback capability.

All DIO lines have 10K Ohm pull-up resistors.

Ruby-MM-1612 Universal Driver Functions

- dscDAConvert()
- dscDAConvertScan()
- dscDIOClearBit()
- dscDIOInputBit()
- dscDIOInputByte()
- dscDIOOutputBit()
- dscDIOOutputByte()
- dscDIOSetBit()
- dscDIOSetConfig()

SBYTE
signed char

SDWORD
signed long

SWORD
signed short

TRUE
(BOOL)1

Virtual Test Board
This board type causes the driver to emulate the behavior of a typical data acquisition board. This is useful for writing or testing software without access to real hardware.

Operation

Analog Input
Note: The virtual test board acknowledges true voltages. Input modes (set using DscADSetSettings) are identical to the Diamond-MM-16-AT.
Vin 0-3 emulate a triangle wave, peak-to-peak 20V (-10V to 10V) with a period of 6 seconds. Each input is 0.75 seconds out of phase with the previous channel.
Vin 4-7 are mapped (looped back) to the 4 analog output channels. Vin4 <-> Vout0, Vin5 <-> Vout1, etc. The analog output is fixed at +/-10V.
Vin 8 returns -7.5V.
Vin 9 returns -2.5V.
Vin 10 returns 2.5V.

Vin 11 returns 7.5V.
Vin 12-15 are based on the first four output bits of DIO 0. If the bit is high, the Vin returns 5V. If low, the Vin returns -5V.
Vin 16 returns 10mV.
Vin 17 returns -10mV.
Vin 18-31 return random values on each read.

Analog Output
Vout 0-3 emulate 12-bit output in +/-10V mode, so you may pass in 0-4095 as output values.

Digital I/O
The test board DIO emulates an 82C55 chip, with 3 ports of 8 bits (A, B, CL[4], CH[4]). A is looped back to B, CL is looped back to CH.

The test board does not support DscDIOSetConfig. A is always considered an output, B is always an input, CL is output and CH input.

WORD
unsigned short