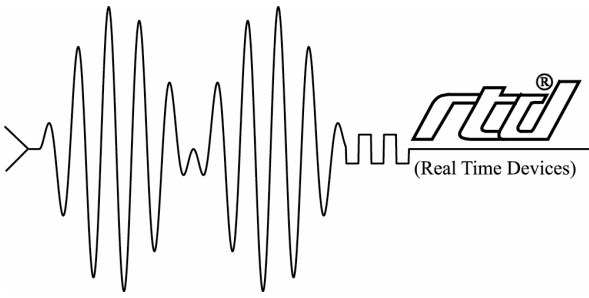
Linux Software Manual (DM6425)

Version 1.0.x



RTD Embedded Technologies, Inc.
"Accessing the Analog World"®

SWM-640010025
rev A

ISO9001 and AS9100 Certified

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803-0906

Phone: +1-814-234-8087
FAX: +1-814-234-5218

E-mail
sales@rtd.com
techsupport@rtd.com

web site
http://www.rtd.com

Revision History

03/22/2006	Revision A issued

Linux Software Manual (DM6425)
Published by:

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College, PA 16803-0906

Copyright © by RTD Embedded Technologies, Inc.
All rights reserved
Printed in U.S.A.

The RTD logo and dataModule are registered trademarks of RTD Embedded Technologies. Linux is a registered trademark of Linus Torvalds. All other trademarks appearing in this document are the property of their respective owners.

Table of Contents

TABLE OF CONTENTS **4**

INTRODUCTION **5**

NOTATIONAL CONVENTIONS **6**

INSTALLATION INSTRUCTIONS **6**

- EXTRACTING THE SOFTWARE 6
- CONTENTS OF INSTALLATION DIRECTORY 6
- BUILDING THE DRIVER 7
- BUILDING THE LIBRARY 8
- BUILDING THE EXAMPLE PROGRAMS 8

USING THE API FUNCTIONS **9**

FUNCTION REFERENCE **10**

API FUNCTION GROUPS **11**

- ANALOG TO DIGITAL CONVERSION 11
- BOARD ID DATA 11
- DIGITAL TO ANALOG CONVERSION 11
- DIGITAL I/O 12
- DIRECT MEMORY ACCESS (DMA) DATA TRANSFER 12
- GENERAL 12
- INTERRUPT CONTROL AND STATUS 13
- TIMER/COUNTER CONTROL AND STATUS 13

ALPHABETICAL FUNCTION LISTING **14**

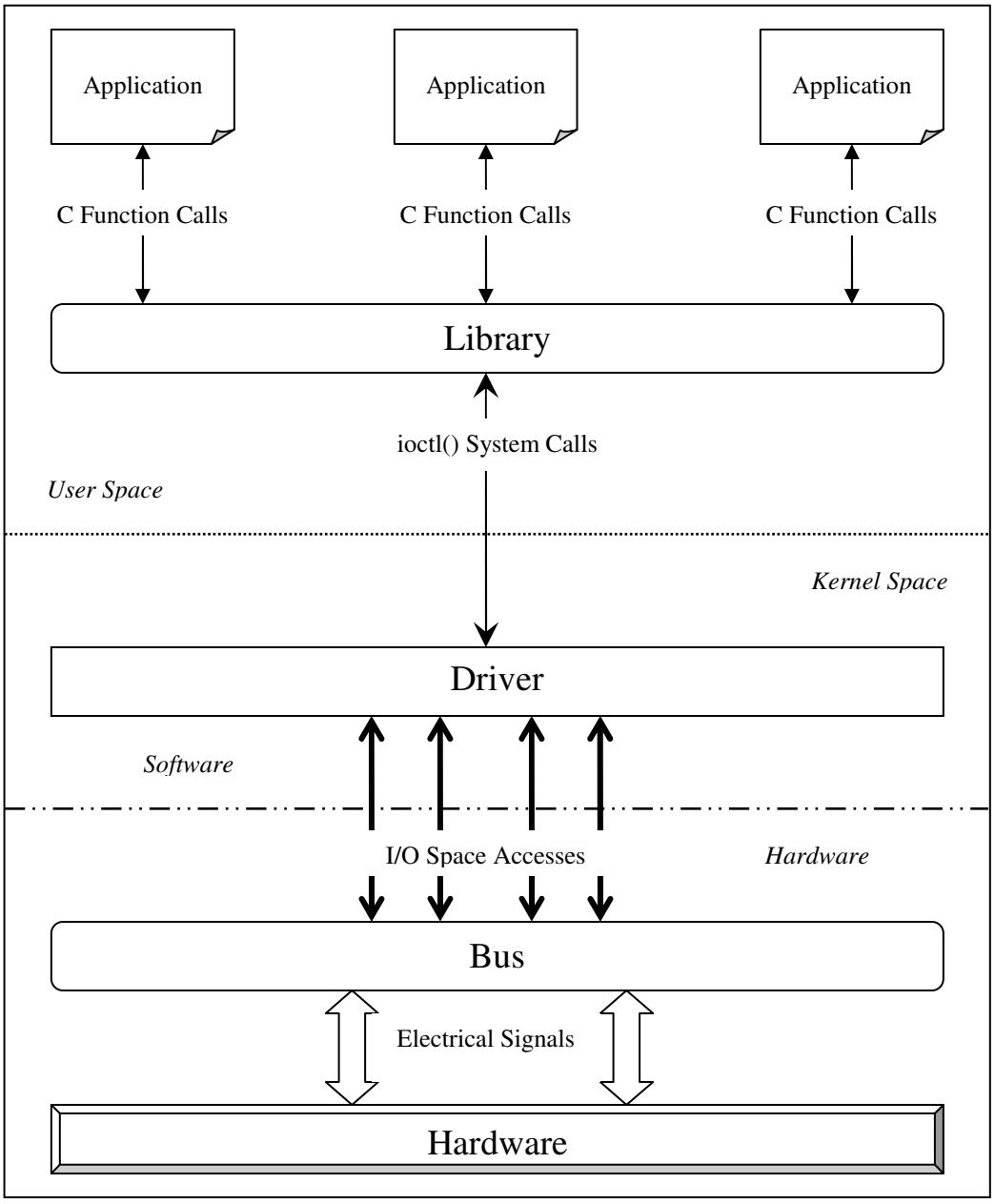
EXAMPLE PROGRAMS REFERENCE **76**

LIMITED WARRANTY **78**

Introduction

This document targets anyone wishing to write Linux applications for an RTD DM6425 analog I/O dataModule. It provides information on building the software and about the Application Programming Interface used to communicate with the hardware and driver. Each high-level library function is described.

The diagram below 1) provides a general overview of what hardware and software entities are involved in device access, 2) shows which units communicate with each other, and 3) illustrates the methods used to transfer data and control information.



Notational Conventions

RTD Linux drivers are assigned version numbers. These version numbers take the form "**A**.**B**.**C**" where:

* **A** is the major release number. This will be incremented whenever major changes are made to the software. Changing the major release number requires updating the software manual.
* **B** is the minor release number. This will be incremented whenever minor, yet significant, changes are made to the software. Changing the minor release number requires updating the software manual.
* **C** is the patch level number. This will be incremented whenever very minor changes are made to the software. Changing the patch level number does not require updating the software manual.

In this document, you will see driver version numbers with a letter in them. For example, 1.0.x indicates that the topic being discussed is applicable to driver versions with a major release number of 1, a minor release number of 0, and any patch level number.

Occasionally you will notice text placed within the < and > characters, for example <installation path>. This indicates that the text represents something which depends upon choices you have made or upon your specific system configuration.

Installation Instructions

Extracting the Software

All software comes packaged in a gzip'd tar file named dm6425_Linux_v1.0.0.tar.gz. First, decide where you would like to place the software and make a copy of the tar file in this directory by issuing the command "cp <path to tar file>/dm6425_Linux_v1.0.0.tar.gz <installation path>" substituting the appropriate paths for <path to tar file> and <installation path>. Next, change your current directory to where you made a copy of the tar file by issuing the command "cd <installation path>". Once you are in this directory, extract the software by issuing the command "tar –xvzf dm6425_Linux_v1.1.0.tar.gz"; this will create a directory dm6425_Linux_v1.1.0/ which contains all the files that are part of the software package.

Contents of Installation Directory

Once the tar file is extracted, you should see the following files and directories within dm6425_Linux_v1.0.0/:

driver/
examples/
include/
lib/
CHANGES.TXT
LICENSE.TXT
README.TXT

The file CHANGES.TXT describes the changes made to the software for this release, as well as for previous releases. The file LICENSE.TXT provides details about the RTD end user license agreement which must be agreed to and accepted before using this software. The file README.TXT contains a general overview of the software and contact information should you experience problems, have questions, or need information. The directory driver/ contains the source code and Makefile for the drivers. The directory examples/ holds the source code and Makefile for the example programs. The directory include/ contains all header files used by the driver, example programs, library, and your application programs. Library source code and Makefile reside in the directory lib/.

Building the Driver

Driver source code uses files located in the kernel source tree. Therefore, you must have the full kernel source tree available in order to build the driver. The kernel source tree consumes a lot of disk space, on the order of 100 to 200 megabytes. Because production systems rarely contain this much disk space, you will probably use a development machine to compile the driver source code. The development system, which provides a full compilation environment, must be running the exact same version of the kernel as your production machine(s); otherwise the kernel module may not load or may load improperly. After the code is built, you can then move the resulting object files, libraries, and executables to the production system(s).

Building the driver consists of several steps: 1) compiling the source code, 2) loading the resulting kernel module into the kernel, and 3) creating hardware device files in the /dev directory. To perform any of the above steps, you must change your current directory to driver/. The file Makefile contains rules to assist you.

To compile the source code, issue the command “make”. This will create the driver object file rtd-dm6425.o. The GNU C compiler gcc is used to build the driver code.

Before the driver can be used, it must be loaded into the currently running kernel. Using the command “make insmod” will load the DM6425 driver into the kernel. This target assumes that:

* A single DM6425 is installed.
* The board's base I/O address is set to the factory default of 0x300.
* The DM6425 will use IRQs 3 and 5.
* The board will use DMA channels 5 and 6.

If the previous assumptions do not match your hardware setup, you will need to edit the Makefile and change this rule to reflect your board configuration or manually issue an appropriate insmod command.

When you load the kernel driver, the insmod command will print the message "Warning: loading ./rtd-dm6425.o will taint the kernel: no license". You can safely ignore this message since it pertains to GNU General Public License (GPL) licensing issues rather than to driver operation.

The final step is to create /dev entries for the hardware. The driver asks the kernel to dynamically assign a major number. Since this major number may change each time you load the driver, the best way to create the device files is to use the command "make devices"; this generates four files in /dev named rtd-dm6425-0 through rtd-dm6425-3.

If you ever need to unload the driver from the kernel, you can use the command "make rmmod".

Building the Library

The example programs and your application use the DM6425 library, so it must be built before any of these can be compiled. To build the library, change your current directory to lib/ and issue the command “make”. The GNU C compiler gcc is used to compile the library source code.

The DM6425 library is statically linked and is created in the file librtd-dm6425.a.

Building the Example Programs

The example programs may be compiled by changing your current directory to examples/ and giving the command “make”, which builds all the example programs. If you wish to compile a subset of example programs, there are targets in Makefile to do so. For example, the command “make dm6425-auto-burst” will compile and link the source file dm6425-auto-burst.cc and the command "make dm6425-stream" will compile and link the source file dm6425-stream.cc. The GNU C compiler gcc is used to compile the example program code.

Using the API Functions

DM6425 hardware and the associated driver functionality can be accessed through the library API (Application Programming Interface) functions. Applications wishing to use library functions must include the include/dm6425lib.h header file and be statically linked with the lib/librtd-dm6425.a library file.

The following function reference provides for each library routine a prototype, description, explanation of parameters, and return value or error code. By looking at a function's entry, you should gain an idea of: 1) why it would be used, 2) what it does, 3) what information is passed into it, 4) what information it passes back, and 5) how to interpret error conditions that may arise.

Note that errno codes other than the ones indicated in the following pages may be set by the library functions. Unless otherwise noted in the description of a function's return value, please see the ioctl(2) man page for more information.

Function Reference

API Function Groups

Analog to Digital Conversion

ADFifoWriteControl6425
ADTriggerThresholdControl6425
ClearADFIFO6425
ClearChannelGainTable6425
EnableADTriggerThreshold6425
EnableTables6425
GetAutoincData6425
IsAboutTrigger6425
IsADConverting6425
IsADFIFOEmpty6425
IsADFIFOFull6425
IsADHalted6425
LoadADSampleCounter6425
LoadADTable6425
LoadDigitalTable6425
LoadTriggerRegister6425
ReadADData6425
ReadADDataMarker6425
ReadADDataWithMarker6425
ResetChannelGainTable6425
SetBurstTrigger6425
SetBurstTriggerDigitalPort6425
SetChannelGain6425
SetConversionSelect6425
SetPauseEnable6425
SetStartTrigger6425
SetStopTrigger6425
SetTriggerPolarity6425
SetTriggerRepeat6425
StartConversion6425

Board ID Data

ReadBoardIdData6425

Digital to Analog Conversion

ClearDAC6425
DACSimultaneousUpdate6425
LoadDAC6425
SetDACRange6425

Digital I/O

ClearDINFIFO6425
ConfigDINClock6425
DIOClearChip6425
DINClockEnable6425
DIOClearIrq6425
DIOEnableIrq6425
DIOIsChipBytePortOutput6425
DIOIsChipIrq6425
DIOIsChipIRQEnabled6425
DIOIsChipIRQEventMode6425
DIOIsChipStrobe6425
DIOIsChipSystemClock6425
DIOLoadCompare6425
DIOLoadMask6425
DIORead6425
DIOReadCompareRegister6425
DIOReadStatus6425
DIOSelectClock6425
DIOSelectIrqMode6425
DIOSelectRegister6425
DIOSetBitPortDirection6425
DIOSetBytePortDirection6425
DIOWrite6425
IsDINFIFOEmpty6425
IsDINFIFOFull6425
IsDINFIFOHalf6425
LoadDINConfigRegister6425
ReadDINFIFO6425

Direct Memory Access (DMA) Data Transfer

ClearADDMADone6425
DeInstallDMA6425
GetDmaData6425
InstallDMA6425
IsADDMADone6425
IsFirstADDMADone6425
StartDMA6425
StopDMA6425

General

ClearBoard6425
ClearRegister6425
CloseBoard6425
InitBoard6425
LoadControlRegister6425
OpenBoard6425
ReadStatus6425

Interrupt Control and Status

ClearIRQ06425
ClearIRQ16425
DisableIRQ6425
EnableIRQ6425
GetIRQCounter6425
InstallCallbackIRQHandler6425
LoadIRQRegister6425
RemoveIRQHandler6425
SetIRQ0Source6425
SetIRQ1Source6425

Timer/Counter Control and Status

ClockDivisor6425
ClockMode6425
DoneTimer6425
IsBurstClockOn6425
IsPacerClockOn6425
ReadTimerCounter6425
SelectTimerCounter6425
SetBurstClock6425
SetPacerClock6425
SetPacerClockSource6425
SetSampleCounterStop6425
SetUserClock6425

Alphabetical Function Listing

ADFifoWriteControl6425

int ADFifoWriteControl6425(int descriptor, u_int8_t enable);

Description:

Enable or disable A/D FIFO writes when analog threshold triggering is being used.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
enable:	Flag to indicate whether A/D FIFO writes should be enabled. A value of zero means disable A/D FIFO write of samples prior to threshold triggering. Any other value means enable A/D FIFO write of samples prior to threshold triggering.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.

ADTriggerThresholdControl6425

```
int int ADTriggerThresholdControl6425(
    int descriptor,
    u_int8_t level,
    enum DM6425HR_AD_THRESH_TRIGGER channel,
    u_int8_t trigger_above
);
```

Description:

Configure A/D converter trigger threshold.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
level:	Threshold trigger level.

channel: A/D converter channel to trigger on. Valid values are AD_THRESH_TRIGGER_1, AD_THRESH_TRIGGER_2, AD_THRESH_TRIGGER_3, AD_THRESH_TRIGGER_4, AD_THRESH_TRIGGER_5, AD_THRESH_TRIGGER_6, AD_THRESH_TRIGGER_7, AD_THRESH_TRIGGER_8, AD_THRESH_TRIGGER_9, AD_THRESH_TRIGGER_10, AD_THRESH_TRIGGER_11, AD_THRESH_TRIGGER_12, AD_THRESH_TRIGGER_13, AD_THRESH_TRIGGER_14, AD_THRESH_TRIGGER_15, AD_THRESH_TRIGGER_16, AD_THRESH_TRIGGER_17, AD_THRESH_TRIGGER_18, AD_THRESH_TRIGGER_19, AD_THRESH_TRIGGER_20, AD_THRESH_TRIGGER_21, AD_THRESH_TRIGGER_22, AD_THRESH_TRIGGER_23, AD_THRESH_TRIGGER_24, AD_THRESH_TRIGGER_25, AD_THRESH_TRIGGER_26, AD_THRESH_TRIGGER_27, AD_THRESH_TRIGGER_28, AD_THRESH_TRIGGER_29, AD_THRESH_TRIGGER_30, AD_THRESH_TRIGGER_31, and AD_THRESH_TRIGGER_32.

trigger_above: Flag to indicate whether triggering should occur when the A/D value is greater than the threshold. A value of zero means trigger when the A/D value is less than or equal to the threshold value. Any other value means trigger when the A/D value is greater than the threshold.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL channel is not valid.

ClearADDMADone6425

---

int ClearADDMADone6425(int descriptor);

Description:

Clear a board's A/D DMA done flag.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearADFIFO6425

---

int ClearADFIFO6425(int descriptor);

Description:

Clear a board's A/D sample FIFO.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearBoard6425

---

int ClearBoard6425(int descriptor);

Description:

Reset a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearChannelGainTable6425

int ClearChannelGainTable6425(int descriptor);

Description:

Clear the contents of a board's channel/gain table.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearDAC6425

int ClearDAC6425(int descriptor, enum DM6425HR_DAC dac);

Description:

Clear one of the D/A converters on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

dac: D/A converter to clear. Valid values are DM6425HR_DAC1, DM6425HR_DAC2, DM6425HR_DAC3, and DM6425HR_DAC4.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL dac is not valid.

ClearDINFIFO6425

int ClearDINFIFO6425(int descriptor);

Description:

Clear a board's digital input FIFO.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearIRQ06425

int ClearIRQ06425(int descriptor);

Description:

Clear the first interrupt circuit on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearIRQ16425

int ClearIRQ16425(int descriptor);

Description:

Clear the second interrupt circuit on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClearRegister6425

---

int ClearRegister6425(int descriptor, u_int16_t ClearValue);

Description:

Write a bit mask into a board's Program Clear Register and then read from the Clear Register to clear some part(s) of the board. Other library functions make use of this routine to perform their work.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

ClearValue: Bit mask to write into Program Clear Register before reading Clear Register.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

ClockDivisor6425

---

int ClockDivisor6425(int descriptor, enum DM6425HR_CLK Timer, u_int16_t Divisor);

Description:

Set the divisor for the specified counter on the 8254 chip. Before calling this function, the counter must have been set to receive the divisor least significant byte first then most significant byte.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Timer: Indicates which counter to use. Valid values are DM6425HR_CLK0, DM6425HR_CLK1, and DM6425HR_CLK2.

Divisor: Counter divisor value.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL Timer is not valid.

ClockMode6425

```
int ClockMode6425(
   int descriptor,
   enum DM6425HR_CLK Timer,
   enum DM6425HR_CLK_MODE Mode
);
```

Description:

Set the mode of the specified counter on the 8254 chip. This function also sets the indicated counter to receive a subsequent divisor load least significant byte first then most significant byte.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Timer: Indicates which counter to use. Valid values are DM6425HR_CLK0, DM6425HR_CLK1, and DM6425HR_CLK2.

Mode: Indicates which clock mode to set. Valid values are DM6425HR_CLK_MODE0, DM6425HR_CLK_MODE1, DM6425HR_CLK_MODE2, DM6425HR_CLK_MODE3, DM6425HR_CLK_MODE4, and DM6425HR_CLK_MODE5.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Timer is not valid.
EINVAL	Mode is not valid.

CloseBoard6425

---

int CloseBoard6425(int descriptor);

Description:

Close a DM6425 device file opened previously with OpenBoard6425().

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.

Return Value:

0:	Success.
-1:	Failure. Please see the close(2) or ioctl(2) man page for information on possible values errno may have in this case.

ConfigDINClock6425

---

int ConfigDINClock6425(int descriptor, enum DM6425HR_DI_FIFO_CLK DIN_Clock);

Description:

Set the source for a board's digital input FIFO clock.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
DIN_Clock:	Source of digital input FIFO clock. Valid values are DM6425HR_DI_FIFO_CLK_USER_TC0, DM6425HR_DI_FIFO_CLK_USER_TC1, DM6425HR_DI_FIFO_CLK_AD_WRITE_FIFO, DM6425HR_DI_FIFO_CLK_EXTERNAL_PACER, and DM6425HR_DI_FIFO_CLK_EXTERNAL_TRIG.

Return Value:

0:	Success.
-1	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	DIN_Clock is not valid.

DACSimultaneousUpdate6425

---

int DACSimultaneousUpdate6425(int descriptor);

Description:

Initiate the simultaneous update of all D/A converters on a board.

NOTE: You should not use this function unless the D/A converters have been set to update at the same time.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.

Return Value:

0:	Success.
-1	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.

DACSimultaneousUpdateControl6425

---

int DACSimultaneousUpdateControl6425(int descriptor, u_int8_t simultaneous);

Description:

Control whether or not each of the four D/A converters update at the same time.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
simultaneous:	Flag to indicate whether all four D/A converters should update at the same time. A value of zero means the D/A converters should update independently whenever a value is written to any of them. Any other value means the D/A converters should update simultaneously.

Return Value:

0:	Success.
-1	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.

DeInstallDMA6425

---

int DeInstallDMA6425(int descriptor, enum DM6425HR_DMA DMAChannel);

Description:

Configure the specified DMA circuit on a board so that DMA can no longer be performed on it.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
DMAChannel:	DMA circuit to configure. Valid values are DM6425HR_DMA1 and DM6425HR_DMA2.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	DMAChannel is not valid.
EINVAL	No DMA channel was ever allocated to DMAChannel.

DINClockEnable6425

---

int DINClockEnable6425(int descriptor, int Enable);

Description:

Enable or disable a board's digital input FIFO clock.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Enable:	Indicates whether or not to enable the digital input FIFO clock. A value of 0 means disable the clock. A nonzero value means enable the clock.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.

DIOClearChip6425

int DIOClearChip6425(int descriptor, enum DM6425HR_DIO Port);

Description:

Clear the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to clear. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOClearIrq6425

int DIOClearIrq6425(int descriptor, enum DM6425HR_DIO Port);

Description:

Clear the given bit-programmable digital I/O port IRQ status flag.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to clear. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for read access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOEnableIrq6425

---

int DIOEnableIrq6425(int descriptor, enum DM6425HR_DIO Port, int Enable);

Description:

Enable or disable a bit-programmable digital I/O port's digital interrupts.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to modify interrupts for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
Enable:	Flag indicating how digital interrupts should be set. A value of 0 means disable digital interrupts. A nonzero value means enable digital interrupts.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOIsChipBytePortOutput6425

```
int int DIOIsChipBytePortOutput6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    int *port_output_p
);
```

Description:

Determine whether or not the given byte-programmable digital I/O port is set to output.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to check bit direction of. Valid values are DM6425HR_DIO1 and DM6425HR_DIO3.
port_output_p:	Address where port output flag should be stored. If the port is set to output, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.
	EINVAL	Port is not valid.
	EOPNOTSUPP	Port is valid but is not byte-programmable.

DIOIsChipIrq6425

```
int int DIOIsChipIrq6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    int *interrupt_generated_p
);
```

Description:

Determine whether or not the given bit-programmable digital I/O port has generated a digital I/O interrupt.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to check interrupt status of. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
interrupt_generated_p:	Address where interrupt generated flag should be stored. If the port has generated a digital interrupt, a nonzero value will be stored here. Otherwise, zero will be stored here.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for read access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOIsChipIRQEnabled6425

```
int DIOIsChipIRQEnabled6425(
   int descriptor,
   enum DM6425HR_DIO Port,
   int *irq_enabled_p
);
```

Description:

Determine whether or not digital interrupts are enabled for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to check interrupt enabled status for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
irq_enabled_p:	Address where IRQ enabled flag should be stored. If digital interrupts are enabled, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.
	EINVAL	Port is not valid.
	EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOIsChipIRQEventMode6425

```
int DIOIsChipIRQEventMode6425(
   int descriptor,
   enum DM6425HR_DIO Port,
   int *irq_event_p
);
```

Description:

Determine whether or not the given bit-programmable digital I/O port's digital interrupt mode is set to event.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to check IRQ mode of. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
irq_event_p:	Address where IRQ event mode flag should be stored. If digital interrupts are in event mode, a nonzero value will be stored here. Otherwise, 0 will be stored here (meaning digital interrupts are in match mode).

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.
	EINVAL	Port is not valid.
	EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOIsChipStrobe6425

```
int int DIOIsChipStrobe6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    int *strobe_occurred_p
);
```

Description:

Determine whether or not data has been strobed into the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to check strobe status of. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
strobe_occurred_p:	Address where strobe event flag should be stored. If data was strobed into the port, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for read access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOIsChipSystemClock6425

```
int DIOIsChipSystemClock6425(
   int descriptor,
   enum DM6425HR_DIO Port,
   int *system_clock_p
);
```

Description:

Determine whether or not the 8 MHz system clock is driving digital I/O sampling for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to get sampling clock for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
system_clock_p:	Address where system clock flag should be stored. If the 8 MHz system clock is driving digital sampling, a nonzero value will be stored here. Otherwise, 0 will be stored here (meaning that the User Timer/Counter is driving sampling).

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.
	EINVAL	Port is not valid.
	EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOLoadCompare6425

int int DIOLoadCompare6425(int descriptor, enum DM6425HR_DIO Port, u_int8_t Compare);

Description:

Load the Compare Register for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to set compare value for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
Compare:	The bit pattern to be matched.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for write access.
	EINVAL	Port is not valid.
	EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOLoadMask6425

int DIOLoadMask6425(int descriptor, enum DM6425HR_DIO Port, u_int8_t Mask);

Description:

Load the Mask Register for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to set mask for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
Mask:	The bit mask to be loaded.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIORead6425

int DIORead6425(int descriptor, enum DM6425HR_DIO Port, u_int8_t *digital_data_p);

Description:

Read 8 bits of data from board's specified digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Indicates which digital I/O port to read. Valid values are DM6425HR_DIO0, DM6425HR_DIO1, DM6425HR_DIO2, and DM6425HR_DIO3.
digital_data_p:	Address where data read should be stored.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.
EINVAL	Port is not valid.

DIOReadCompareRegister6425

```
int int DIOReadCompareRegister6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    u_int8_t *register_value_p
);
```

Description:

Read the contents of the given bit-programmable digital I/O port Compare Register.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to read compare value for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
register_value_p:	Address where register contents should be stored.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOReadStatus6425

```
int DIOReadStatus6425(
   int descriptor,
   enum DM6425HR_DIO Port,
   u_int8_t *status_p
);
```

Description:

Read the contents of the given digital I/O port's Digital Mode/Status Register. Other library functions make use of this routine to perform their work.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to read status of. Valid values are DM6425HR_DIO0, DM6425HR_DIO1, DM6425HR_DIO2, and DM6425HR_DIO3.
status_p:	Address where digital I/O status register contents should be stored.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.
	EINVAL	Port is not valid.

DIOSelectClock6425

```
int DIOSelectClock6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    enum DM6425HR_CLK_SEL Clock
);
```

Description:

Select the sample clock source for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to set clock source for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
Clock:	Indicates the clock source. Valid values are DM6425HR_CLOCK_TC and DM6425HR_USER_TC.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Clock is not valid.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOSelectIrqMode6425

```
int DIOSelectIrqMode6425(
   int descriptor,
   enum DM6425HR_DIO Port,
   enum DM6425HR_DIO_IRQ IrqMode
);
```

Description:

Set the advanced digital interrupt mode for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to set IRQ mode for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
IrqMode:	Indicates the digital interrupt mode. Valid values are DM6425HR_DIO_IRQ_EVENT and DM6425HR_DIO_IRQ_MATCH .

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EINVAL	IrqMode is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOSelectRegister6425

```
int DIOSelectRegister6425(
   int descriptor,
   enum DM6425HR_DIO Port,
   enum DM6425HR_REG_SEL Select
);
```

Description:

Set the mode of the given bit-programmable digital I/O port Clear/Direction/Mask/Compare Register. Other library functions make use of this routine to perform their work.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to set mode for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
Select:	Which mode to configure. Valid values are DM6425HR_REG_CLEAR, DM6425HR_REG_DIR, DM6425HR_REG_MASK, and DM6425HR_REG_CMP.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EINVAL	Select is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOSetBitPortDirection6425

```
int DIOSetBitPortDirection6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    u_int8_t Direction
);
```

Description:

Set the bit direction (input or output) for the given bit-programmable digital I/O port.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Port:	Digital I/O port to set bit direction for. Valid values are DM6425HR_DIO0 and DM6425HR_DIO2.
Direction:	Bit mask indicating bit direction for port bits. A 1 in the mask means set that bit to output. A 0 in the mask means set that bit to input.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not bit-programmable.

DIOSetBytePortDirection6425

```
int DIOSetBytePortDirection6425(
    int descriptor,
    enum DM6425HR_DIO Port,
    int Direction
);
```

Description:

Set the bit direction (input or output) for the given byte-programmable digital I/O port.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Port: Digital I/O port to set bit direction for. Valid values are DM6425HR_DIO1 and DM6425HR_DIO3.

Direction: Flag indicating bit direction. A value of 0 means set port bits to input. A nonzero value means set port bits to output.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.
EOPNOTSUPP	Port is valid but is not byte-programmable.

DIOWrite6425

---

int DIOWrite6425(int descriptor, enum DM6425HR_DIO Port, u_int8_t Data);

Description:

Write data to a board's selected digital I/O port.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Port: The port to write to. Valid values are DM6425HR_DIO0, DM6425HR_DIO1, DM6425HR_DIO2, and DM6425HR_DIO3.

Data: The data to write.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Port is not valid.

DisableIRQ6425

int DisableIRQ6425(int descriptor, enum DM6425HR_INT IRQChannel);

Description:

Disable the specified interrupt circuit on a board.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
IRQChannel:	Interrupt circuit to disable. Valid values are DM6425HR_INT1 and DM6425HR_INT2.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	IRQChannel is not valid.

DoneTimer6425

int DoneTimer6425(int descriptor);

Description:

Initialize a board's user timer/counter counter 0 and counter 1 for high speed to ensure immediate load. This function puts these two counters into mode 2.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.

Return Value:

0:	Success.
-1:	Failure. Please see the descriptions of the ClockDivisor6425(), ClockMode6425(), and SelectTimerCounter6425() functions or the ioctl(2) man page for information on possible values errno may have in this case.

EnableIRQ6425

int EnableIRQ6425(int descriptor, enum DM6425HR_INT IRQChannel);

Description:

Enable the specified interrupt circuit on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

IRQChannel: Interrupt circuit to enable. Valid values are DM6425HR_INT1 and DM6425HR_INT2.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL IRQChannel is not valid.

EnableADTriggerThreshold6425

int EnableADTriggerThreshold6425(int descriptor, u_int8_t enable);

Description:

Enable or disable A/D converter threshold triggering.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

enable: Flag to indicate whether A/D converter threshold triggering should be enabled. A value of zero means disable triggering. Any other value means enable triggering.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EnableTables6425

int EnableTables6425(int descriptor, int Enable_AD_Table, int Enable_Digital_Table);

Description:

Enable or disable the A/D and digital tables in the channel/gain scan memory on a board.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Enable_AD_Table:	Flag to indicate whether the A/D table should be enabled. A value of 0 means disable the A/D table. A nonzero value means enable the A/D table.
Enable_Digital_Table:	Flag to indicate whether the digital table should be enabled. A value of 0 means disable the digital table. A nonzero value means enable the digital table.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EOPNOTSUPP	The digital table is to be enabled but the A/D table is to be disabled.

GetAutoincData6425

```
int GetAutoincData6425(
   int descriptor,
   enum DM6425HR_STR_Regs from_register,
   enum DM6425HR_STR_TYPE type,
   void *buffer_p,
   size_t element_num
);
```

Description:

Initiate a streaming read from a board. Once can specify the board register to read from, what size data is to be transferred, and how many data elements to transfer.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
from_register:	Register from which the data should be read. Valid values are rSTR_AD_6425 and rSTR_DIN_FIFO_6425.
type:	Type/size of element to be transferred. Valid values are DM6425HR_STR_TYPE_BYTE and DM6425HR_STR_TYPE_WORD.
buffer_p:	Address of buffer in which to place the data read.
element_num:	How many data elements of type "type" should be read.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for read access.
EFAULT	buffer_p is not a valid user address.
EINVAL	from_register is not valid.
EINVAL	type is not valid.
EOPNOTSUPP	from_register is rSTR_AD_6425 and type is DM6425HR_STR_TYPE_BYTE.
EOPNOTSUPP	from_register is rSTR_DIN_FIFO_6425 and type is DM6425HR_STR_TYPE_WORD.

GetDmaData6425

```
int GetDmaData6425(
   int descriptor,
   void *dma_buffer_p,
   enum DM6425HR_DMA DMAChannel,
   size_t length,
   size_t offset,
   size_t *bytes_transferred_p
);
```

Description:

Copy data from specified DMA circuit's DMA buffer into user buffer.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
dma_buffer_p:	Address of user buffer.
DMAChannel:	DMA circuit to operate on. Valid values are DM6425HR_DMA1 and DM6425HR_DMA2.
length:	Number of bytes to transfer.
offset:	Offset in bytes from beginning of driver's DMA buffer where read should begin.
bytes_transferred_p:	Address where actual number of bytes transferred will be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for read access.
EFAULT	dma_buffer_p is not a valid user address.
EFAULT	dma_buffer_p is not large enough to hold the data.
EINVAL	DMAChannel is not valid.
EINVAL	No DMA channel was ever allocated to DMAChannel.
EINVAL	No DMA buffer was ever allocated to DMAChannel.
EINVAL	(length + offset) lies beyond the end of the driver's DMA buffer.

GetIRQCounter6425

```
int GetIRQCounter6425(
   int descriptor,
   enum DM6425HR_INT IRQChannel,
   unsigned long *counter_value_p
);
```

Description:

Get the number of interrupts that have occurred on a board's specified interrupt circuit.

Data returned from this function is interpreted differently depending upon usage of the force argument on the insmod command. If you specify a nonzero value for the force argument, the value returned represents the number of interrupts that occurred since the driver module was loaded. If you specify a zero value for the force argument or do not use force at all, the value returned represents the number of interrupts that occurred since the device file was opened.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
IRQChannel:	Interrupt circuit to read counter value from. Valid values are DM6425HR_INT1 and DM6425HR_INT2.
counter_value_p:	Address where counter value should be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for read access.
EINVAL	IRQChannel is not valid.
EINVAL	No IRQ was ever allocated to IRQChannel.

InitBoard6425

---

int InitBoard6425(int descriptor);

Description:

Initialize a board. This will 1) clear the board, 2) clear the A/D DMA done flag, 3) clear the channel/gain table, and 4) clear the A/D input FIFO.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.

InstallCallbackIRQHandler6425

```
int InstallCallbackIRQHandler6425(
   int descriptor,
   void (*callback)(void),
   enum DM6425HR_INT IRQChannel
);
```

Description:

Install a function which will be called whenever an interrupt occurs and the driver sends a signal to the process to indicate that the interrupt happened.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
callback:	Address of callback function.
IRQChannel:	Board interrupt circuit that signal should be attached to. Valid values are DM6425HR_INT1 and DM6425HR_INT2.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	callback is NULL.
EINVAL	IRQChannel is not valid.
ENOMEM	Library callback descriptor memory could not be allocated.

Please see the sigaction(2) man page, the sigprocmask(2) man page, or the ioctl(2) man page for information on other possible values errno may have in this case.

InstallDMA6425

```
int InstallDMA6425(int descriptor, enum DM6425HR_DMA DMAChannel);
```

Description:

Configure the specified DMA circuit on a board to be able to perform DMA.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
DMAChannel:	DMA circuit to configure. Valid values are DM6425HR_DMA1 and DM6425HR_DMA2.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for write access.
	EINVAL	DMAChannel is not valid.
	EINVAL	No DMA channel was ever allocated to DMAChannel.
	ENOMEM	DMA buffer memory allocation failed.

IsAboutTrigger6425

---

int IsAboutTrigger6425(int descriptor, int *ad_about_trigger_p);

Description:

Determine whether or not a board's A/D about trigger has occurred.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_about_trigger_p:	Address where A/D about trigger flag should be stored. If A/D about trigger has occurred, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.

IsADConverting6425

int IsADConverting6425(int descriptor, int *ad_converting_p);

Description:

Determine whether or not a board's A/D converter is converting.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_converting_p:	Address where A/D converting flag should be stored. If A/D conversion is occurring, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.

IsADDMADone6425

int IsADDMADone6425(int descriptor, int *ad_dma_done_p);

Description:

Determine whether or not a board's A/D DMA transfer is complete.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_dma_done_p	Address where A/D DMA done flag should be stored. If A/D DMA is finished, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.

IsADFIFOEmpty6425

int IsADFIFOEmpty6425(int descriptor, int *ad_fifo_empty_p);

Description:

Determine whether or not a board's A/D FIFO is empty.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_fifo_empty_p:	Address where FIFO empty flag should be stored. If the A/D FIFO is empty, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.

IsADFIFOFull6425

int IsADFIFOFull6425(int descriptor, int *ad_fifo_full_p);

Description:

Determine whether or not a board's A/D FIFO is full.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_fifo_empty_p:	Address where FIFO full flag should be stored. If the A/D FIFO is full, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for read access.

IsADHalted6425

int IsADHalted6425(int descriptor, int *ad_halted_p);

Description:

Determine whether or not a board's A/D conversion has been stopped because the sample buffer is full.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_halted_p:	Address where A/D halted flag should be stored. If A/D conversion has been stopped, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

IsBurstClockOn6425

int IsBurstClockOn6425(int descriptor, int *ad_burst_clock_on_p);

Description:

Determine whether or not a board's A/D burst clock gate is on.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_burst_clock_on_p:	Address where A/D burst clock flag should be stored. If A/D burst clock gate is on, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

IsDINFIFOEmpty6425

int IsDINFIFOEmpty6425(int descriptor, int *digital_fifo_empty_p);

Description:

Determine whether or not a board's digital input FIFO is empty.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
digital_fifo_empty_p:	Address where digital FIFO empty flag should be stored. If the digital FIFO is empty, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

IsDINFIFOFull6425

int IsDINFIFOFull6425(int descriptor, int *digital_fifo_full_p);

Description:

Determine whether or not a board's digital input FIFO is full.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
digital_fifo_full_p:	Address where digital FIFO full flag should be stored. If the digital FIFO is full, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

IsDINFIFOHalf6425

int IsDINFIFOHalf6425(int descriptor, int *digital_fifo_half_full_p);

Description:

Determine whether or not a board's digital input FIFO is half full.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
digital_fifo_half_full_p:	Address where digital FIFO half full flag should be stored. If the digital FIFO is half full, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

IsFirstADDMADone6425

int IsFirstADDMADone6425(int descriptor, int *ad_first_dma_done_p);

Description:

Determine whether or not a board's A/D first DMA transfer is complete when in dual channel DMA mode.

NOTE: The description of the ad_first_dma_done_p parameter below is based upon the hardware manual's description of the First DMA Flag in the Status Register. However, the manual erroneously provides an inverted interpretation of this flag. Since this function examines the First DMA Flag, the value stored in *ad_first_dma_done_p is inverted also. In reality, the value stored in this address will be zero if A/D first DMA transfer is finished and nonzero otherwise.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_first_dma_done_p:	Address where A/D first DMA done flag should be stored. If A/D first DMA is finished, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

IsPacerClockOn6425

---

int IsPacerClockOn6425(int descriptor, int *ad_pacer_clock_on_p);

Description:

Determine whether or not a board's A/D pacer clock gate is on.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_pacer_clock_on_p:	Address where A/D pacer clock flag should be stored. If A/D pacer clock gate is on, a nonzero value will be stored here. Otherwise, 0 will be stored here.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

LoadADSampleCounter6425

---

int LoadADSampleCounter6425(int descriptor, u_int16_t NumOfSamples);

Description:

Load a board's analog to digital sample counter.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
NumOfSamples:	Number of samples to take.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

LoadADTable6425

---

int LoadADTable6425(int descriptor, u_int16_t ADEntries, ADTableRow *ADTable_p);

Description:

Load a board's A/D table with the given number of entries.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ADEntries:	Number of entries in A/D table.
ADTable_p:	Address of memory containing A/D table to send to board.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	ADEntries is 0.
EINVAL	ADEntries is greater than 1024.
EINVAL	An entry in the table pointed to by ADTable_p has an invalid Channel, Gain, ADRange, or Se_Diff member variable value.

LoadControlRegister6425

---

int LoadControlRegister6425(int descriptor, u_int16_t value);

Description:

Load a value into a board's Control Register.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

value: Data to write into Control Register.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

LoadDAC6425

---

int LoadDAC6425(int descriptor, enum DM6425HR_DAC dac, u_int16_t Data);

Description:

Load a 12-bit binary value into one of the digital to analog converters on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

dac: D/A converter to load. Valid values are DM6425HR_DAC1 and DM6425HR_DAC2.

Data: Value to write to converter.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL dac is not valid.

LoadDigitalTable6425

---

int LoadDigitalTable6425(int descriptor, u_int16_t entries, u_int8_t *table_p);

Description:

Load a board's digital table with the given number of entries.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
entries:	Number of entries in digital table.
table_p:	Address of memory containing digital table to send to board. This memory is an array of unsigned bytes.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for write access.
	EINVAL	entries is 0.
	EINVAL	entries is greater than 1024.

LoadDINConfigRegister6425

---

int LoadDINConfigRegister6425(int descriptor, u_int16_t value);

Description:

Load a 16-bit value into a board's Digital Input FIFO Configuration Register.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
value:	Value to write into register.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for write access.

LoadIRQRegister6425

---

int LoadIRQRegister6425(int descriptor, u_int16_t value);

Description:

Load a 16-bit value into a board's Interrupt Register.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

value: Value to load into Interrupt Register.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

LoadTriggerRegister6425

---

int LoadTriggerRegister6425(int descriptor, u_int16_t value);

Description:

Load a 16-bit value into a board's Trigger Mode Register.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

value: Value to write into Trigger Mode Register.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

OpenBoard6425

---

int OpenBoard6425(int DeviceNumber);

Description:

Open a DM6425 device file.

Parameters:

DeviceNumber: Minor number of board device file.

Return Value:

>=0:	Success. The integer returned is the file descriptor from open() system call.
-1:	Failure. Please see the open(2) man page for information on possible values errno may have in this case.

ReadADData6425

---

int ReadADData6425(int descriptor, int16_t *ad_data_p);

Description:

Read the 12-bit A/D sample from a board's analog to digital FIFO.

NOTE: This function discards marker data from the A/D FIFO data. Calling ReadADData6425() and ReadADDataMarker6425() separately with the intention of getting data from a single sample will likely result in data being returned from two distinct samples.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
ad_data_p:	Address where A/D sample data read should be stored.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

ReadADDataMarker6425

---

int ReadADDataMarker6425(int descriptor, u_int8_t *data_marker_p);

Description:

Read the 3-bit data marker from a board's analog to digital FIFO .

NOTE: This function discards A/D sample data from the A/D FIFO data. Calling ReadADDataMarker6425() and ReadADData6425() separately with the intention of getting data from a single sample will likely result in data being returned from two distinct samples.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

data_marker_p: Address where data marker should be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

ReadADDataWithMarker6425

---

int ReadADDataWithMarker6425(int descriptor, int16_t *ad_fifo_p);

Description:

Read the entire 16-bit contents of a board's analog to digital FIFO . The most significant bit is the sign bit. The next 12 bits are the converted A/D data. The least significant 3 bits are the data marker.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

ad_fifo_p: Address where A/D FIFO contents should be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

ReadBoardIDData6425

---

int ReadBoardIdData6425(int descriptor, struct Dm6425BoardId *board_id_p);

Description:

Read a device's board ID data.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

board_id_p: Address where board ID data should be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

ReadDINFIFO6425

---

int ReadDINFIFO6425(int descriptor, u_int8_t *digital_data_p);

Description:

Read 8 bits of data from the port 0 digital input FIFO.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

digital_data_p: Address where data read should be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

ReadStatus6425

---

int ReadStatus6425(int descriptor, u_int16_t *status_p);

Description:

Read a board's Status Register. Other library functions make use of this routine to perform their work.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

status_p: Address where status should be stored.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for read access.

ReadTimerCounter6425

```
int ReadTimerCounter6425(
   int descriptor,
   enum DM6425HR_CLK_SEL Timer,
   enum DM6425HR_CLK Clock,
   u_int16_t *counter_value_p
);
```

Description:

Read the 16 bit contents of the desired timer/counter. The read is done as two 8-bit reads: least significant byte then most significant byte.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Timer: Indicates which timer/counter to use. Valid values are DM6425HR_CLOCK_TC and DM6425HR_USER_TC.

Clock: Indicates which counter to use. Valid values are DM6425HR_CLK0, DM6425HR_CLK1, and DM6425HR_CLK2.

counter_value_p: Address where timer contents should be stored.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Timer is not valid.
EINVAL	Clock is not valid.

RemoveIRQHandler6425

---

int RemoveIRQHandler6425(int descriptor, enum DM6425HR_INT IRQChannel);

Description:

Uninstall the function which was previously registered as an interrupt callback by InstallCallbackIRQHandler6425().

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
IRQChannel:	Board interrupt circuit that signal should be detached from. Valid values are DM6425HR_INT1 and DM6425HR_INT2.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	IRQChannel is not valid.
EINVAL	No IRQ was ever allocated to IRQChannel.

ResetChannelGainTable6425

---

int ResetChannelGainTable6425(int descriptor);

Description:

Reset a board's channel/gain table starting point to the beginning of the table.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

SelectTimerCounter6425

---

int SelectTimerCounter6425(int descriptor, enum DM6425HR_CLK_SEL Select);

Description:

Select which timer/counter on a board will be accessed when a subsequent operation is performed on the registers located at base I/O address + 16 through base I/O address + 22. Other library functions make use of this routine to perform their work.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Select: Indicate which timer/counter will be accessed. Valid values are DM6425HR_CLOCK_TC and DM6425HR_USER_TC.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL Select is not valid.

SetBurstClock6425

---

int SetBurstClock6425(int descriptor, double BurstRate, double *actual_p);

Description:

Set a board's burst clock rate.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
BurstRate:	Burst clock rate desired.
actual_p:	Address where the actual programmed frequency should be stored. If this function fails, the actual frequency is not updated.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.

SetBurstTrigger6425

---

int SetBurstTrigger6425(int descriptor, enum DM6425HR_BURST_TRIG Burst_Trigger);

Description:

Select a board's burst mode trigger.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Burst_Trigger:	What triggers a burst. Valid values are DM6425HR_BURST_TRIG_SOFTWARE, DM6425HR_BURST_TRIG_PACER, DM6425HR_BURST_TRIG_EXTERNAL, and DM6425HR_BURST_TRIG_DIGITAL.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Burst_Trigger is not valid.

SetBurstTriggerDigitalPort6425

```
int int SetBurstTriggerDigitalPort6425(
    int descriptor,
    enum DM6425HR_BURST_DIGITAL_PORT port
);
```

Description:

Select which digital I/O port serves as the interrupt source when the burst clock is triggered by digital interrupts.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
port:	Digital I/O port to use as interrupt source. Valid values are DM6425HR_BURST_PORT_0 and DM6425HR_BURST_PORT_2.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for write access.
	EINVAL	port is not valid.

SetChannelGain6425

```
int SetChannelGain6425(
   int descriptor,
   enum DM6425HR_AIN Channel,
   enum DM6425HR_GAIN Gain,
   enum DM6425HR_RANGE Range,
   enum DM6425HR_SE Se_Diff
);
```

Description:

Load a board's channel/gain latch.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Channel:	A/D channel. Valid values are DM6425HR_AIN1, DM6425HR_AIN2, DM6425HR_AIN3, DM6425HR_AIN4, DM6425HR_AIN5, DM6425HR_AIN6, DM6425HR_AIN7, DM6425HR_AIN8, DM6425HR_AIN9, DM6425HR_AIN10, DM6425HR_AIN11, DM6425HR_AIN12, DM6425HR_AIN13, DM6425HR_AIN14, DM6425HR_AIN15, DM6425HR_AIN16, DM6425HR_AIN17, DM6425HR_AIN18, DM6425HR_AIN19, DM6425HR_AIN20, DM6425HR_AIN21, DM6425HR_AIN22, DM6425HR_AIN23, DM6425HR_AIN24, DM6425HR_AIN25, DM6425HR_AIN26, DM6425HR_AIN27, DM6425HR_AIN28, DM6425HR_AIN29, DM6425HR_AIN30, DM6425HR_AIN31, and DM6425HR_AIN32.
Gain:	A/D gain. Valid values are DM6425HR_GAINx1, DM6425HR_GAINx2, DM6425HR_GAINx4, and DM6425HR_GAINx8.
Range:	A/D input voltage range and polarity. Valid values are DM6425HR_RANGE_SIGNED5, DM6425HR_RANGE_SIGNED10, and DM6425HR_RANGE_UNSIGNED10.
Se_Diff:	Select single-ended or differential mode. Valid values are DM6425HR_SE_SE and DM6425HR_SE_DIFF.

Return Value:

0:	Success.	
-1:	Failure with errno set as follows:	
	EACCES	descriptor refers to a file that is open but not for write access.
	EINVAL	Channel is not valid.
	EINVAL	Gain is not valid.
	EINVAL	Range is not valid.
	EINVAL	Se_Diff is not valid.

SetConversionSelect6425

int SetConversionSelect6425(int descriptor, enum DM6425HR_CONV Select);

Description:

Configure how a board's A/D conversion is done.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Select: Indicates how A/D conversions are controlled. Valid values are DM6425HR_CONV_SOFT_TRIGGER, DM6425HR_CONV_PACER_CLOCK, DM6425HR_CONV_BURST_CLOCK, and DM6425HR_CONV_DIGITAL_INT.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL Select is not valid.

SetDACRange6425

```
int SetDACRange6425(
    int descriptor,
    enum DM6425HR_DAC dac,
    enum DM6425HR_DAC_SETUP range
);
```

Description:

Set the range and polarity for one of the D/A converters on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

dac: D/A converter to set range/polarity for. Valid values are DM6425HR_DAC1, DM6425HR_DAC2, DM6425HR_DAC3, and DM6425HR_DAC4.

range: Polarity/range to set. Valid values are DAC_SETUP_BIPOLAR_5, DAC_SETUP_UNIPOLAR_5, DAC_SETUP_BIPOLAR_10, and DAC_SETUP_UNIPOLAR_10.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	dac is not valid.
EINVAL	range is not valid.

SetIRQ0Source6425

---

int SetIRQ0Source6425(int descriptor, enum DM6425HR_INTSRC IRQSource);

Description:

Select the interrupt source of the first interrupt circuit on a board.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
IRQSource:	Source of interrupt on first circuit. Valid values are IRQS_AD_SAMPLE_CNT_6425, IRQS_AD_START_CONVERT_6425, IRQS_AD_END_CONVERT_6425, IRQS_AD_WRITE_FIFO_6425, IRQS_AD_FIFO_HALF_6425, IRQS_AD_DMA_DONE_6425, IRQS_RESET_GAIN_TABLE_6425, IRQS_PAUSE_GAIN_TABLE_6425, IRQS_EXT_PACER_CLOCK_6425, IRQS_EXT_TRIGGER_6425, IRQS_DIGITAL_6425, IRQS_TC_COUNTER0_6425, IRQS_TC_COUNTER0_INVERTED_6425, IRQS_TC_COUNTER1_6425, IRQS_DIO_FIFO_HALF_6425, IRQS_DIO_WRITE_FIFO_6425, IRQS_DIGITAL_INT_PORT_2_6425, IRQS_ANALOG_THRESHOLD_6425, IRQS_EXTERNAL_INT_1_6425, and IRQS_EXTERNAL_INT_2_6425.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL IRQSource is not valid.

SetIRQ1Source6425

---

int SetIRQ1Source6425(int descriptor, enum DM6425HR_INTSRC IRQSource);

Description:

Select the interrupt source of the second interrupt circuit on a board.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

IRQSource: Source of interrupt on first circuit. Valid values are
IRQS_AD_SAMPLE_CNT_6425,
IRQS_AD_START_CONVERT_6425,
IRQS_AD_END_CONVERT_6425,
IRQS_AD_WRITE_FIFO_6425,
IRQS_AD_FIFO_HALF_6425,
IRQS_AD_DMA_DONE_6425,
IRQS_RESET_GAIN_TABLE_6425,
IRQS_PAUSE_GAIN_TABLE_6425,
IRQS_EXT_PACER_CLOCK_6425,
IRQS_EXT_TRIGGER_6425,
IRQS_DIGITAL_6425,
IRQS_TC_COUNTER0_6425,
IRQS_TC_COUNTER0_INVERTED_6425,
IRQS_TC_COUNTER1_6425,
IRQS_DIO_FIFO_HALF_6425,
IRQS_DIO_WRITE_FIFO_6425,
IRQS_DIGITAL_INT_PORT_2_6425,
IRQS_ANALOG_THRESHOLD_6425,
IRQS_EXTERNAL_INT_1_6425, and
IRQS_EXTERNAL_INT_2_6425.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	IRQSource is not valid.

SetPacerClock6425

int SetPacerClock6425(int descriptor, double clock, double *actual_p);

Description:

Set a board's pacer clock rate. This function decides whether to use a 16-bit or 32-bit clock depending upon the clock rate.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
clock:	Clock rate desired.
actual_p:	Address where the actual programmed frequency should be stored. If this function fails, the actual frequency is not updated.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	clock is greater than 8 MHz.
EINVAL	clock is 0.0.
EINVAL	clock is less than 0.0.

SetPacerClockSource6425

int SetPacerClockSource6425(int descriptor, enum DM6425HR_PACER_CLK Source);

Description:

Select the source of a board's pacer clock.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Source: Pacer clock source. Valid values are DM6425HR_PACER_CLK_INTERNAL and DM6425HR_PACER_CLK_EXTERNAL.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL Source is not valid.

SetPauseEnable6425

---

int SetPauseEnable6425(int descriptor, int Enable);

Description:

Enable or disable a board's A/D table pause bit.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Enable: Flag to indicate whether the A/D table pause bit should set. A value of 0 means enable the pause bit. A nonzero value means disable the pause bit.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

SetSampleCounterStop6425

---

int SetSampleCounterStop6425(int descriptor, int Disable);

Description:

Enable or disable a board's A/D Sample Counter Stop.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Disable: Flag to indicate whether the A/D sample counter should stop the pacer clock. A value of 0 means enable sample counter stop. A nonzero value means disable sample counter stop.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

SetStartTrigger6425

int SetStartTrigger6425(int descriptor, enum DM6425HR_START_TRIG Start_Trigger);

Description:

Configure how a board's pacer clock is started during A/D conversion.

Parameters:

descriptor: File descriptor from OpenBoard6425() call.

Start_Trigger: What starts the pacer clock. Valid values are DM6425HR_START_TRIG_SOFTWARE, DM6425HR_START_TRIG_EXTERNAL, DM6425HR_START_TRIG_DIGITAL_INT, DM6425HR_START_TRIG_USER_TC1, and DM6425HR_START_TRIG_GATE.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES descriptor refers to a file that is open but not for write access.

EINVAL Start_Trigger is not valid.

EOPNOTSUPP Start_Trigger is one of the three reserved bit patterns as defined in the hardware manual.

SetStopTrigger6425

int SetStopTrigger6425(int descriptor, enum DM6425HR_STOP_TRIG Stop_Trigger);

Description:

Configure how a board's pacer clock is stopped during A/D conversion.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Stop_Trigger:	What stops the pacer clock. Valid values are DM6425HR_STOP_TRIG_SOFTWARE, DM6425HR_STOP_TRIG_EXTERNAL, DM6425HR_STOP_TRIG_DIGITAL_INT, DM6425HR_STOP_TRIG_SAMPLE_CNT, DM6425HR_STOP_TRIG_ABOUT_SOFTWARE, DM6425HR_STOP_TRIG_ABOUT_EXTERNAL, DM6425HR_STOP_TRIG_ABOUT_DIGITAL, and DM6425HR_STOP_TRIG_ABOUT_USER_TC1.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Stop_Trigger is not valid.

SetTriggerPolarity6425

int SetTriggerPolarity6425(int descriptor, enum DM6425HR_POLAR Polarity);

Description:

Select which edge of an external pacer clock triggers a board's burst mode.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Polarity:	Which external pacer clock edge triggers burst mode. Valid values are DM6425HR_POLAR_POSITIVE and DM6425HR_POLAR_NEGATIVE.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Polarity is not valid.

SetTriggerRepeat6425

---

int SetTriggerRepeat6425(int descriptor, enum DM6425HR_REPEAT Repeat);

Description:

Select whether or not a trigger initiates multiple A/D conversion cycles.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Repeat:	Indicates whether or not A/D conversion cycles repeat. Valid values are DM6425HR_REPEAT_SINGLE and DM6425HR_REPEAT_REPEAT.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Repeat is not valid.

SetUserClock6425

```
int SetUserClock6425(
   int descriptor,
   enum DM6425HR_CLK Timer,
   double InputRate,
   double OutputRate,
   double *actual_p
);
```

Description:

Set up a board's user clock.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
Timer:	Indicates which counter to use. Valid values are DM6425HR_CLK0, DM6425HR_CLK1, and DM6425HR_CLK2.
InputRate	Input frequency to specified counter.
OutputRate:	Desired output rate from specified counter.
actual_p:	Address where the actual programmed frequency should be stored. If this function fails, the actual frequency is not updated.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	Timer is not valid.

StartConversion6425

---

int StartConversion6425(int descriptor);

Description:

Issue a Start Convert (software trigger) command to a board.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for read access.

StartDMA6425

int StartDMA6425(int descriptor, enum DM6425HR_DMA DMAChannel, size_t TransferBytes);

Description:

Start DMA on a board's specified DMA circuit.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
DMAChannel:	DMA circuit to operate on. Valid values are DM6425HR_DMA1 and DM6425HR_DMA2.
TransferBytes:	Number of bytes to transfer in a single DMA. This value must be even because the entities transferred are 16-bit signed converted A/D values.

Return Value:

0: Success.

-1: Failure with errno set as follows:

EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	DMAChannel is not valid.
EINVAL	No DMA channel was ever allocated to DMAChannel.
EINVAL	No DMA buffer was ever allocated to DMAChannel.
EINVAL	TransferBytes is odd.
EINVAL	TransferBytes is greater than the DMA buffer size.

StopDMA6425

int StopDMA6425(int descriptor, enum DM6425HR_DMA DMAChannel);

Description:

Stop DMA on a board's specified DMA circuit.

Parameters:

descriptor:	File descriptor from OpenBoard6425() call.
DMAChannel:	DMA circuit to operate on. Valid values are DM6425HR_DMA1 and DM6425HR_DMA2.

Return Value:

0:	Success.
-1:	Failure with errno set as follows:
EACCES	descriptor refers to a file that is open but not for write access.
EINVAL	DMAChannel is not valid.
EINVAL	No DMA channel was ever allocated to DMAChannel.

Example Programs Reference

Name	Remarks
dm6425-analog-trigger	Demonstrates how to use analog threshold triggering.
dm6425-auto-burst	Demonstrates how to use the pacer clock, burst clock, sample counter, and channel/gain table to scan groups of channels multiple times.
dm6425-auto-scan	Demonstrates using the sample counter and pacer clock to do multi-scan sampling.
dm6425-board-id	Demonstrates how to read a device's board ID data.
dm6425-dac	Demonstrates how to use the digital to analog converter.
dm6425-dac-simultaneous	Demonstrates how to simultaneously update all D/A converter outputs.
dm6425-digital-io	Demonstrates how to read and write the digital I/O ports.
dm6425-dma	Demonstrates using DMA transfers to acquire data.
dm6425-dual-dma	Demonstrates using both DMA circuits on a board to transfer data.
dm6425-event-interrupt	Demonstrates how to use event mode advanced digital interrupts.
dm6425-external-interrupt	Demonstrates how to use external interrupts.
dm6425-match-interrupt	Demonstrates how to use match mode advanced digital interrupts.
dm6425-multi-burst	Demonstrates how to perform an analog to digital conversion on multiple channels using the channel/gain table.
dm6425-sample-counter	Demonstrates how to use the sample counter to generate interrupts.
dm6425-soft-trigger	Demonstrates how to initiate an analog to digital conversion using a software trigger.
dm6425-speed-test	Demonstrates three different IRQ handling and data read methods: 1) using a callback routine invoked via the driver interrupt handler sending a signal to the process; data is read using streaming input, 2) polling the IRQ counter for an interrupt circuit; data is read one sample at a time, and 3) polling the IRQ counter for an interrupt circuit; data is read using streaming input.
dm6425-stream	Demonstrates using streaming input to read blocks of data from the digital input FIFO.
dm6425-strobe-interrupt	Demonstrates how to use strobe digital interrupts.
dm6425-test-lib-errors	Test program to validate library error checking. This program exercises the error checking code added to the library. All exported functions are verified. The following types of tests are performed: 1) a function fails when expected, 2) if a function fails as expected. errno is set as expected depending upon the cause of failure, 3) a function succeeds when expected, and 4) if a function can accept multiple valid inputs, it succeeds on all combinations of such inputs.
dm6425-throughput	Allows one to see how the driver works when a board is converting A/D data at a particular throughput rate and converted data is being read using a specified method. Data can be retrieved as follows: 1) reading one sample at a time, 2) using streaming input, 3) using single channel DMA, and 4) using dual channel DMA. In all cases, interrupts are used to signal availability of data. Every five seconds, the program prints status information.
dm6425-timers	Demonstrates how to program the 8254 programmable interval timers.
dm6425-user-timer	Demonstrates how to set up the User Timer to generate interrupts at a specified rate, poll the appropriate interrupt counter, and perform an

	A/D conversion when the counter changes.

Limited Warranty

RTD Embedded Technologies, Inc. warrants the hardware and software products it manufactures and produces to be free from defects in materials and workmanship for one year following the date of shipment from RTD Embedded Technologies, Inc. This warranty is limited to the original purchaser of product and is not transferable.

During the one year warranty period, RTD Embedded Technologies will repair or replace, at its option, any defective products or parts at no additional charge, provided that the product is returned, shipping prepaid, to RTD Embedded Technologies. All replaced parts and products become the property of RTD Embedded Technologies. Before returning any product for repair, customers are required to contact the factory for an RMA number.

THIS LIMITED WARRANTY DOES NOT EXTEND TO ANY PRODUCTS WHICH HAVE BEEN DAMAGED AS A RESULT OF ACCIDENT, MISUSE, ABUSE (such as: use of incorrect input voltages, improper or insufficient ventilation, failure to follow the operating instructions that are provided by RTD Embedded Technologies, "acts of God" or other contingencies beyond the control of RTD Embedded Technologies), OR AS A RESULT OF SERVICE OR MODIFICATION BY ANYONE OTHER THAN RTD Embedded Technologies. EXCEPT AS EXPRESSLY SET FORTH ABOVE, NO OTHER WARRANTIES ARE EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, ANY IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND RTD Embedded Technologies EXPRESSLY DISCLAIMS ALL WARRANTIES NOT STATED HEREIN. ALL IMPLIED WARRANTIES, INCLUDING IMPLIED WARRANTIES FOR MECHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED TO THE DURATION OF THIS WARRANTY. IN THE EVENT THE PRODUCT IS NOT FREE FROM DEFECTS AS WARRANTED ABOVE, THE PURCHASER'S SOLE REMEDY SHALL BE REPAIR OR REPLACEMENT AS PROVIDED ABOVE. UNDER NO CIRCUMSTANCES WILL RTD Embedded Technologies BE LIABLE TO THE PURCHASER OR ANY USER FOR ANY DAMAGES, INCLUDING ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, EXPENSES, LOST PROFITS, LOST SAVINGS, OR OTHER DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PRODUCT.

SOME STATES DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES FOR CONSUMER PRODUCTS, AND SOME STATES DO NOT ALLOW LIMITATIONS ON HOW LONG AN IMPLIED WARRANTY LASTS, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM STATE TO STATE.

RTD Embedded Technologies, Inc.
103 Innovation Boulevard
State College PA 16803-0906
USA
Our website: www.rtd.com