Data Acquisition Software Library

Daqlib

For daqdrv and

General Standards Daq Cards

16AISS8AO4

16AISS16AO2

Linux Version

Dr. M. C. Nelson

Sensor Realtime, LLC

daqlib

Rev. 070112

Notice and Disclaimer:

This document and the software described herein are provided with no claim or guarantee of safety, reliability or suitability for any purpose whatsoever.

Always test your software and systems exhaustively under safe, controlled circumstances before deployment.

Bug reports and feature requests are welcome.

Dr. M. C. Nelson
Sensor Realtime, LLC
12/26/2006

Contents

1 INTRODUCTION 1

2 INSTALLATION 1

3 PACKAGE ARCHITECTURE 2

4 PROGRAMMING INTERFACE 2

4.1 A SIMPLE EXAMPLE 2
4.2 RETURN VALUES 4
4.3 DAQIO 4
4.3.1 DATA STRUCTURES 4
4.3.2 FUNCTIONS 4
4.3.2.1 int daqReadRaw(DaqBoard *board, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg); 4
4.3.2.2 int daqWriteRaw(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg); 5
4.3.2.3 int daqWriteReadRaw(DaqBoard *b, unsigned int *uwrite, int nwrite, unsigned int *uread, int nread, int (*readyfunc)(void *), void *readyarg); 5
4.4 DAQTHRESHOLD 6
4.4.1 DATA STRUCTURES 6
4.4.2 FUNCTIONS 6
4.4.2.1 int daqReadThreshold(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, int (*thresholdfunc)(void *, int, void *), void *thresholdarg, int nlatent, int npretrig); 6
4.4.2.2 int daqReadThresholdLevel(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, float level, int direction, int channel, nlatency, int npretrig); 7
4.4.2.3 int daqReadThresholdEdge(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, float level, int direction, int channel, nlatency, int npretrig); 7

4.4.2.4 int daqReadThresholdTone(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, int channel, float level, float f, int nlatency, int npretrig); 8
4.5 DAQLIB **8**
4.5.1 DATA STRUCTURES 8
4.5.1.1 DaqBoard 8
4.5.2 FUNCTIONS 10
4.5.2.1 int daqOpen(DaqBoard *b, int n); 10
4.5.2.2 void daqClose(DaqBoard *b); 11
4.5.2.3 int daqReadStatus(DaqBoard *b); 11
4.5.2.4 int daqClearUser(DaqBoard *b); 11
4.5.2.5 int daqSetUser(DaqBoard *b); 11
4.5.2.6 DaqBoard *daqGetUser(int fd); 11
4.5.2.7 int daqReadInputSize(DaqBoard *b); 11
4.5.2.8 int daqReadOutputSize(DaqBoard *b); 11
4.5.2.9 int daqReadControls(DaqBoard *b); 11
4.5.2.10 int daqWriteControls(DaqBoard *b); 11
4.5.2.11 int daqTrigger(DaqBoard *b); 11
4.5.2.12 Int daqTriggerInput_(int fd) 12
4.5.2.13 Int daqTriggerOutput_(int fd) 12
4.5.2.14 Int daqTriggerInputOutput_(int fd) 12
4.5.2.15 int daqInit(DaqBoard *b); 12
4.5.2.16 int daqAddInputBuffer(DaqBoard *b, unsigned int *data, int ndata); 12
4.5.2.17 int daqReleaseInputBuffers(DaqBoard *b); 12
4.5.2.18 int daqStartInputDma(DaqBoard *b); 13
4.5.2.19 int daqCancelInputDma(DaqBoard *b); 13
4.5.2.20 int daqWaitInput(DaqBoard *b); 13
4.5.2.21 void *daqWaitInputBuffer(DaqBoard *b); 13
4.5.2.22 int daqAddInputBuffer_(int fd, unsigned int *data, int ndata); 13
4.5.2.23 int daqReleaseInputBuffers_(int fd); 13
4.5.2.24 int daqStartInputDma_(int fd); 13
4.5.2.25 int daqCancelInputDma_(int fd); 13
4.5.2.26 int daqAddOutputBuffer(DaqBoard *b, unsigned int *data, int ndata); 13
4.5.2.27 int daqReleaseOutputBuffers(DaqBoard *b); 14

4.5.2.28 int daqStartOutputDma(DaqBoard *b);....14
4.5.2.29 int daqCancelOutputDma(DaqBoard *b);....14
4.5.2.30 int daqWaitOutput(DaqBoard *b);....14
4.5.2.31 void *daqWaitOutputBuffer(DaqBoard *b);....14
4.5.2.32 int daqAddOutputBuffer_(int fd, unsigned int *data, int ndata);....14
4.5.2.33 int daqReleaseOutputBuffers_(int fd);....14
4.5.2.34 int daqStartOutputDma_(int fd);....15
4.5.2.35 int daqCancelOutputDma_(int fd);....15
4.6 DAQUTIL....15
4.6.1 FUNCTIONS....15
4.6.1.1 void *daqMalloc(void *p, unsigned int nsize);....15
4.6.1.2 void *daqMallocNoFill(void *p, unsigned int nsize);....15
4.6.1.3 void *daqMallocAligned(void *p, unsigned int nsize);....15
4.7 DAQREGS....15
4.7.1 LOCAL BOARD REGISTER FUNCTIONS....15
4.7.1.1 Programmed IO functions....15
4.7.2 PLX PCI INTERFACE FUNCTIONS....16
4.7.3 DIAGNOSTICS DUMP....16
4.8 GSCLIB....16
4.8.1 CLOCK DIVIDER....16
4.8.2 RANGE....16
4.8.3 OPEN, CLOSE, INIT....16
4.8.4 DIGITAL I/O....17
4.8.5 DIGITAL I/O....17
4.8.6 OUTPUT BUFFER....17
4.8.7 CHECK DATA SYNCH MARKERS.....17
4.8.8 DATA TRANSLATION BETWEEN RAW AND REAL WORLD VALUES.....17
4.8.9 REGISTER DUMP.....18

1 Introduction

The daqlib software library provides a simple programming interface for **daqdrv**, the data acquisition device driver for real-time data acquisition. The library supports event driven and streaming i/o with single or multi-buffering, status and control functions, synchronous and asynchronous DMA and PIO data transfers to and from user space buffers, and also provides register level access to the data acquisition card and its PCI interface controller. CPU requirements for daqlib are minimal.

Simple examples for a number of common scenarios are provided in the samples directory. Users are encouraged to peruse the sample programs as an adjunct to this document.

2 Installation

The daqlib and sample programs are provide with the daqdrv driver, in a single file daqkit{date}.tgz. The contents of the tar-zip file are:

./daqlib	
libdaqlib.a	The data acquisition software library.
gsclib.h daqregs.h daqio.h daqutil.h daqlib.h daqdrv.h daqprint.h daqthreshold.h fft.h	C-language include files
daqcmd testecho.sh testwrite.sh testread.sh	Command line utility and shell scripts to test and exercise the driver and library.
./samples	
exercise.c, readburst.c writewave.c echoburst.c readcircle.c readleveltrigger.c readtonetrigger.c readusertrigger.c Makefile	Sample programs and make file
./daqdriver	
daqdrv.ko	The data acquisition driver as a loadable module for kernel 2.6.21
daqdrvstart.sh	Shell script to start the driver
daqdrvstop.sh	Shell script to stop the driver
daqdrv.h	C language IOCTL codes and data structures.
Makefile daqdrvmain.c daqdrvmain.h gsc16aiss8ao4.c gsc16aiss8ao4.h gsc16aiss8ao4regs.h gsc16aiss16ao2.c gsc16aiss16ao2.h gsc16aiss16ao2regs.h plx_regs.h	Source code files needed to rebuild the data acquisition driver for a different kernel.
daqlib.pdf daqdrv.pdf	Software documents.

The files can be unzipped to a directory of the user's choosing. The daqlib directory should be added to the search lists for include files and libraries (see the documentation for gcc and make or the sample makefile in the samples directory). The driver has to be started before the examples will run. See the daqdrv manual for instructions on building the driver for different kernel versions.

3 Package Architecture

The daqlib software package comprises several components. The **daqio** component provides high level read and write functions. The **daqthreshold** component provides a customizable trigger capability. The **daqlib** component provides supporting routines for buffer management and device management. The **gsclib** component provides board level functions and conversions between raw and floating point and transposed data formats. Utility and memory allocation functions are provided in **daqutil**, and register level functions are in **daqregs.**

4 Programming Interface

4.1 A simple example

The following example sets some controls, reads the analog input, and dumps the data to stdout. The source code file and a compiled executable are found in the samples directory.

```
#include "daqutil.h"
#include "daqio.h"
#include "daqprint.h"
#include "gsclib.h"
#include <stdio.h>
#include <stdlib.h>

int main( )
{
  DaqBoard board = { 0 };
  unsigned int *buffer = NULL;
  int retv = 0;

  retv = daqOpen( &board, 0 );
  if ( retv ) { perror( "daqOpen"); exit( 1 ); }
```

```
// Set input sampling rate to 100 kHz , internal clock
board.ctl.ictl.ndiv          = gscClockDivider( board.ctl.cfg.clock, 100.E3 );
board.ctl.ictl.clkmaster     = 1;

// Set single ended input, range +/- 5 Volts
board.ctl.ictl.nmode         = 1;
board.ctl.ictl.nrange        = gscRange( 5.f );      // VFS = 5 volts

// Set 8 input channels, burst length 100,000 samples
board.ctl.ictl.nchans        = 8;
board.ctl.ictl.nburst        = 100000;

// Internal software trigger
board.ctl.trigmaster         = 1;

// Load the controls
retv = daqWriteControls( &board );
if ( retv ) { perror( "daqWriteControls"); exit( 1 ); }

// Allocate a data buffer (alignment is optional)
buffer = daqMallocAligned( buffer,  board.ctl.ictl.ndata*sizeof(unsigned
                    int) );
if ( !buffer ) { printf( "malloc failed\n");  exit( 1 ); }

// Start the DMA,  trigger the input burst and wait for completion
retv = daqReadRaw( &board, buffer, board.ctl.ictl.ndata,
                        (int(*)(void*))daqTrigger, &board );
if ( retv ) { perror("daqReadRaw"); exit( 1 ); }

// Dump the data as real voltage values
printRawFlt( stdout, buffer, board.ctl.ictl.ndata, board.ctl.ictl.nchans,
        board.ctl.ictl.vfs );
free( buffer );
exit( 0 );
}
```

4.2 Return values

Generally, routines return 0 for success, and negative values on error, unless otherwise noted.

4.3 Daqio

The daqio component provides high level functions equivalent to read and write. The read and write routines allow the user to provide a trigger or start function for the data acquisition.

The daqio component also provides start, wait, and stop versions of the read and write functions.

4.3.1 Data Structures

The daqio component uses the DaqBoard structure type defined by the daqlib component.

4.3.2 Functions

4.3.2.1 int daqReadRaw(DaqBoard *board, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg);

Function: Transfers raw data to the user space buffer **udata** of length **ndata** 32 bit words, from the analog input FIFO.

Prerequisites: The card device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below).

The user space buffer ***udata** of length **ndata** words, is mapped by the driver and locked into memory. The DMA engine is then started and waits for the data to be available. If **readyfunc** is not null, it is invoked at this point with the argument **readyarg**. The routine then waits for the transfer to complete, stops the DMA engine, and unmaps and releases the buffer.

The function can be performed incrementally by using the following functions.

*4.3.2.1.1 int daqReadRawStart(DaqBoard *board, unsigned int *udata, int ndata);*

Function: Maps and locks the buffer, and starts the DMA engine

*4.3.2.1.2 int daqReadWait(DaqBoard *b);*

Function: Waits for the transfer to complete

*4.3.2.1.3 int daqReadRawStop(DaqBoard *b);*

Function: Stops the DMA engine and unmaps and releases the buffer.

4.3.2.2 int daqWriteRaw(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg);

Function: Transfers raw data from the user space buffer **udata** length **ndata** 32 bit words, to the analog input FIFO.

Prerequisites: The device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below).

The user space buffer ***udata** of length **ndata** words, is mapped by the driver and locked into memory. The DMA engine is then started and waits for space to be available. If **readyfunc** is not null, it is invoked at this point with the argument **readyarg**. The routine then waits for the transfer to complete, stops the DMA engine, and unmaps and release the buffer.

The function can be performed incrementally by using the following functions.

*4.3.2.2.1 int daqWriteRawStart(DaqBoard *b, unsigned int *udata, int ndata);*

Function: Maps and locks the buffer, and starts the DMA engine

*4.3.2.2.2 int daqWriteWait(DaqBoard *b);*

Function: Waits for the transfer to complete

*4.3.2.2.3 int daqWriteRawStop(DaqBoard *b);*

Function: Stops the DMA engine and unmaps and releases the buffer.

4.3.2.3 int daqWriteReadRaw(DaqBoard *b, unsigned int *uwrite, int nwrite, unsigned int *uread, int nread, int (*readyfunc)(void *), void *readyarg);

Function: Transfers raw data from the user space buffer **uwrite** length **nwrite** 32 bit words, to the analog output FIFO on card **board**, and to the buffer **uread**, **nread**, from analog input FIFO.

Prerequisites: The card device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below).

The user space buffers are mapped by the driver and locked into memory. The two DMA engines are then started and wait for space and/or data to be available. If **readyfunc** is not null, it is invoked at this point with the argument **readyarg**. The routine then waits for the transfers to complete, stops the DMA engines, and unmaps and releases the buffers.

4.4 DaqThreshold

The daqthreshold component provides high level functions for data collection threshold triggered by either, an analog level, an edge, the presence of a specific frequency or tone, or a user supplied function. The parameters include a latency and pretrigger length. The latency sets the number of sample clocks of data to examine at one time for the threshold. The pretrigger length is the number of clock cycles of data prior to the threshold trigger. The builtin threshold functions have been tested on 1 GHz processor, for data rates up to 1 MHz x 8 channels.

4.4.1 Data Structures

The daqthreshold component uses the DaqBoard structure type defined by the daqlib component. The built-in threshold functions internal data structures declared in daqtrigger.h. Users may provide their own threshold functions and structures.

4.4.2 Functions

4.4.2.1 int daqReadThreshold(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, int (*thresholdfunc)(void *, int, void *), void *thresholdarg, int nlatent, int npretrig);

Function: Continuously scans the analog input stream until a threshold condition is met and then transfers pre and post trig data to the user buffer.

Prerequisites: The card device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below). Generally the input burst length should be set to 0, so that the input can be scanned for as long as necessary.

The function creates an internal set of buffers linked in a ring. Each buffer is of length equal to **nlatent** sample clocks of data. The **readyfunc,** if not null, is invoked with the argument **readyarg**, to start the ring buffered input. The incoming data is then scanned by the threshold function,

Found_a_signal = (* thresholdfunc)(buffer, nwords, thresholdarg);

The thresholdfunc returns a nonzero value when the threshold condition is met. The routine then copies the previous **npretrig** sample clocks of data to **udata**, and continues to transfer data to the user buffer until **ndata** words have been transferred. The routine then stops the DMA and releases and frees the internal buffers.

The threshold triggered read can be performed incrementally by using the following functions.

4.4.2.1.1 *int daqReadThresholdStart(DaqBoard *b, int nlatent, int npretrig);*

Function: Creates, maps and locks the ring buffers, and starts the DMA engine

4.4.2.1.2 *int daqReadThresholdWait(DaqBoard *b, unsigned int *udata, int ndata, int (*thresholdfunc)(void *, int, void *), void *thresholdarg);*

Function: Scans the input stream until the threshold condition is satisfied and then copies the data to the user buffer.

4.4.2.1.3 *int daqReadThresholdStop(DaqBoard *b);*

Function: Stops the dma engine, releases and frees the ring buffers

4.4.2.2 int daqReadThresholdLevel(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, float level, int direction, int channel, nlatency, int npretrig);

Function: Continuously scans the analog input stream until the signal in the specified channel is above the specified level (or below, direction < 0), and then transfers pre and post trig data to the user buffer.

Prerequisites: The card device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below). Generally the input burst length should be set to 0, so that the input can be scanned for as long as necessary.

This function provides a level triggered read using a built-in thresholding function.

4.4.2.3 int daqReadThresholdEdge(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, float level, int direction, int channel, nlatency, int npretrig);

Function: Continuously scans the analog input stream until the signal in the specified channel crosses the specified level, and then transfers pre and post trig data to the user buffer.

Prerequisites: The card device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below). Generally the input burst length should be set to 0, so that the input can be scanned for as long as necessary.

This function provides a level triggered read using a built-in thresholding function.

4.4.2.4 int daqReadThresholdTone(DaqBoard *b, unsigned int *udata, int ndata, int (*readyfunc)(void *), void *readyarg, int channel, float level, float f, int nlatency, int npretrig);

Function: Continuously scans the analog input stream until a tone at the specified frequency is detected above the specified power level and then transfers pre and post trig data to the user buffer. The power level is calcuated as the PSD, normalized by the window width.

Prerequisites: The card device has been opened with **daqOpen()** (see daqlib, below) and the controls have been set with **daqWriteControls()** (see daqlib, below). Generally the input burst length should be set to 0, so that the input can be scanned for as long as necessary.

This function provides a tone (frequency selected) triggered read.

4.5 daqlib

4.5.1 Data Structures

The daqlib component manages and references the following data structures. Refer to the hardware manuals for more detailed explanations of the controls.

4.5.1.1 DaqBoard

Daqboard is populated by daqOpen(), and then referenced by all of the high level functions in daqlib and daqio.

```
typedef struct daqboard {

    DAQDRV_Status      status;        // Status defined by daqdrv.h

    DAQDRV_Ctl         ctl;           // Controls defined by daqdrv.h

    sem_t              isem;          // Analog Input DMA internals
    int                iwait;
    unsigned int       istatus;
    unsigned int       icounter;
    unsigned int       ioverflow;

    DaqBuff            *ibufferlist;
    DaqBuff            *ibuffertail;
    int                ibuffers;

    sem_t              osem;          // Analog Output DMA internals
    int                owait;
    unsigned int       ostatus;
```

```
        unsigned int        ocounter;
        unsigned int        ooverflow;

        DaqBuff             *obufferlist;
        DaqBuff             *obuffertail;
        int                 obuffers;

        DaqBuff             *iring0;
        DaqBuff             *iring1;

        DaqBuff             *oring0;
        DaqBuff             *oring1;

        Int                 fd;            // File descriptor for the device

    } DaqBoard;
```

4.5.1.1.1 DAQDRV_Status

The DAQDRV_Status structure returns high level status information from the device driver.

```
    typedef struct daqdrv_status {
        unsigned int status;               // high level driver status
        unsigned int intstatus;            // Primary status register
        unsigned int dma0counter;          // Completed input transfers
        unsigned int dma1counter;          // Completed output transfers
        unsigned int dma0overflow;         // Input FIFO overflows
        unsigned int dma1overflow;         // Output FIFO underflows
        unsigned int userdata;             // Points to DaqBoard for this fd
        unsigned int useraddr0;            // Last completed input buffer
        unsigned int useraddr1;            // Last completed output buffer
        unsigned int usersize0;            // Size in bytes, input buffer
        unsigned int usersize1;            // Size in bytes, output buffer
    } DAQDRV_Status;
```

4.5.1.1.2 DAQDRV_Ctl

The following structure contains descriptions of the board configuration, the input and output control settings and a control for internal software triggering or external triggering.

```
    typedef struct daq_ctl {
        DaqBoardConfig      cfg;           // board assembly (read-only)
        DaqAioCtl           ictl;          // analog input controls
        DaqAioCtl           octl;          // analog output controls
        Int                 trigmaster;    // SW trigger (1), external (0)
    } DAQDRV_Ctl;
```

4.5.1.1.2.1 DaqBoardConfig

The following structure describes the hardware configuration.

```
typedef struct daqboardconfig {
    int         version;            // Firmware version
    int         inchans;            // Analog input channels
    int         outchans;           // Analog output channels
    float       clock;              // Master clock rate
} DaqBoardConfig;
```

4.5.1.1.2.2 DaqAioCtl

The DaqAioCtl structure contains control settings for the analog input and analog output functions of the hardware. The control functions settings are in the comment fields below. The controls are described in more detail in the hardware manual. Routines for translating some of the control settings are included in gsclib and gscregs components.

```
typedef struct daqaioctl {
    float vfs;              // Volts Full Scale
    float clockrate;        // Sample rate/channel in Hz
    int   ndiv;             // clock divider
    int   clkmaster;        // 1=internal clock , 0=external clock
    int   nmode;            //  for input , 0=single ended, 1=differential, etc.
    int   nrange;           //  input range setting (0=2.5V,1=5V,2=10V)
    int   nchans;           //   number of active channels (0 - nmax)
    int   nburst;           //  burst length (samples/channel)
    int   nthreshold;       //  fifo threshold level
    int   ndata;            // data requirement ( nchans * nburst )
} DaqAioCtl;
```

4.5.1.1.3 DaqBuff

The DaqBuff structure contains a pointer to a user space buffer and a size in number of 32 bit words.

```
typedef struct daqbuff {
    unsigned int *p;
    int  n;
    void *next;
} DaqBuff;
```

4.5.2 Functions

4.5.2.1 int daqOpen(DaqBoard *b, int n);

Function: Open the device, initialize the device, read the control and status settings, and setup to receive notifications from the device driver.

The memory pointed to by DaqBoard *b must continue to exist in the program until the device is closed by daqClose(). The pointer is saved in the device driver.

Data acquisition cards installed in the computer, are numbered from 0 thru 3.

4.5.2.2 void daqClose(DaqBoard *b);

Function: Close the device (see gscClose_()).

4.5.2.3 int daqReadStatus(DaqBoard *b);

Function: Read the status from the device driver into the status data in DaqBoard *b;

4.5.2.4 int daqClearUser(DaqBoard *b);

Function: Clear the userdata field in the device driver and update the status data.

This function should not be called by programs that use the DMA functions provided in the daqio and dalqlib components.

4.5.2.5 int daqSetUser(DaqBoard *b);

Function: Set the user data in the device driver to contain the pointer DaqBoard *b.

This function is called by daqOpen() in setting up the dma management mechanism.

4.5.2.6 DaqBoard *daqGetUser(int fd);

Function: Get the address of the DaqBoard structure associated with the file descriptor fd.

4.5.2.7 int daqReadInputSize(DaqBoard *b);

Function: Returns the input buffer size, i.e. the amount of data in the input FIFO.

4.5.2.8 int daqReadOutputSize(DaqBoard *b);

Function: Returns the output buffer size, i.e. the amount of data in the output FIFO.

4.5.2.9 int daqReadControls(DaqBoard *b);

Function: Reads the control settings from the device into DaqBoard *b.

4.5.2.10 int daqWriteControls(DaqBoard *b);

Function: Loads the hardware controls from the settings in DaqBoards *b.

4.5.2.11 int daqTrigger(DaqBoard *b);

Function: Sets the input and output burst triggers if enabled in b->ctl.trigmaster, or toggles the first digital I/O line to provide a hardware trigger signal.

Prerequisites: For this routine, if trigmaster is 0, the board is operated with external triggering and the first digital I/O line is assumed to be connected to the trigger input.

For current hardware such as the 16AISS16AO2, the software triggering for analog input and output is simultaneous.

For earlier hardware the burst start bits are located in different registers. When operated in software triggered mode (trigmaster=1), the driver starts the input and then the output, and the offset is generally 4 to 8 usec. The recommended method to achieve simultaneous triggering in the earlier hardware is to set the controls to trigmaster=0 and drive the trigger input from the first digital i/o line.

4.5.2.12 Int daqTriggerInput_(int fd)

Function: Low level routine to trigger an analog input burst.

4.5.2.13 Int daqTriggerOutput_(int fd)

Function: Low level routine to trigger an analog output burst.

4.5.2.14 Int daqTriggerInputOutput_(int fd)

Function: Low level routine to simultaneously trigger an analog input burst and an analog output burst (see the comments above).

4.5.2.15 int daqInit(DaqBoard *b);

Function: Initialize the device by toggling the INITIALIZE bit in the Board Control Register

4.5.2.16 int daqAddInputBuffer(DaqBoard *b, unsigned int *data, int ndata);

Function: Add a buffer to the device drive input DMA engine.

The buffer is mapped and locked into memory until released. Multiple buffers can be added to the input DMA engine. The buffers should not overlap. See daqutil for routines for allocating page aligned buffers (page alignment is not strictly necessary).

If one buffer is added to the DMA engine, the transfer occurs one time after the DMA engine is started. If two or more buffers are added, the DMA engine cycles through the buffers automatically as data is avaialble. A signal is generated at the completion of each buffer.

4.5.2.17 int daqReleaseInputBuffers(DaqBoard *b);

Function: Releases all of the input buffers.

Buffers must be released before they are free'd.

4.5.2.18 int daqStartInputDma(DaqBoard *b);

Function: Start the input DMA engine

Prerequisites: At least one buffer must be added before starting the DMA.

The DMA transfer runs asynchronously as data becomes available. The transfer can be restarted by calling this function after the transfer completes.

4.5.2.19 int daqCancelInputDma(DaqBoard *b);

Function: Cancel the pending input DMA transfer

4.5.2.20 int daqWaitInput(DaqBoard *b);

Function: Wait for the pending input DMA transfer to complete. Returns 1 for DMA done, other values for error or overflow.

4.5.2.21 void *daqWaitInputBuffer(DaqBoard *b);

Function: Wait for the pending input DMA transfer and return a pointer to the completed user space buffer.

4.5.2.22 int daqAddInputBuffer_(int fd, unsigned int *data, int ndata);

Function: Low level routine that adds an input buffer to the driver.

4.5.2.23 int daqReleaseInputBuffers_(int fd);

Function: Low level routine that releases all input buffers from the driver.

4.5.2.24 int daqStartInputDma_(int fd);

Function: Low level routine that starts an input DMA transfer.

4.5.2.25 int daqCancelInputDma_(int fd);

Function: Low level routine that cancels an input DMA transfer

4.5.2.26 int daqAddOutputBuffer(DaqBoard *b, unsigned int *data, int ndata);

Function: Add a buffer to the device drive output DMA engine.

The buffer is mapped and locked into memory until released. Multiple buffers can be added to the output DMA engine. The buffers should not overlap. See daqutil for routines for allocating page aligned buffers (page alignment is not strictly necessary, but some users may prefer to us page aligned buffers for the DMA engine).

If one buffer is added to the DMA engine, the transfer occurs one time after the DMA engine is started. If two or more buffers are added, the DMA engine cycles through the buffers, as space becomes available in the output FIFO.

4.5.2.27 int daqReleaseOutputBuffers(DaqBoard *b);

Function: Releases all of the output buffers.

Buffers must be released before they are free'd.

4.5.2.28 int daqStartOutputDma(DaqBoard *b);

Function: Start the output DMA engine

Prerequisites: At least one buffer must be added before starting the DMA.

The DMA transfer runs asynchronously as space becomes available in the output FIFO. The transfer can restarted by calling this function after the transfer completes.

4.5.2.29 int daqCancelOutputDma(DaqBoard *b);

Function: Cancel the pending output DMA transfer

4.5.2.30 int daqWaitOutput(DaqBoard *b);

Function: Wait for the pending output DMA transfer to complete. Returns 1 for DMA done, other values for error or over/underflow.

4.5.2.31 void *daqWaitOutputBuffer(DaqBoard *b);

Function: Wait for the pending output DMA transfer and return a pointer to the completed user space buffer.

4.5.2.32 int daqAddOutputBuffer_(int fd, unsigned int *data, int ndata);

Function: Low level routine that adds a buffer to the driver.

4.5.2.33 int daqReleaseOutputBuffers_(int fd);

Function: Low level routine that releases all output buffers from the driver.

4.5.2.34 int daqStartOutputDma_(int fd);

Function: Low level routine that starts an output DMA transfer.

4.5.2.35 int daqCancelOutputDma_(int fd);

Function: Low level routine that cancels an output DMA transfer

4.6 daqutil

The daqutil component provides memory management functions used with other components.

4.6.1 Functions

4.6.1.1 void *daqMalloc(void *p, unsigned int nsize);

Function: The memory *p is freed if not null, and reallocated to the specified size and filled with zeros.

4.6.1.2 void *daqMallocNoFill(void *p, unsigned int nsize);

Function: The memory *p is freed if not null, and reallocated to the specified size.

4.6.1.3 void *daqMallocAligned(void *p, unsigned int nsize);

Function: The memory *p is freed if not null, and reallocated to the specified size on a page aligned boundary. The size is rounded up to the next complete page.

4.7 Daqregs

The daqregs component provides low level register access functions.

4.7.1 Local Board Register Functions

The following functions read and/or write the local board registers. Offsets are specified in bytes from the base of the local register space.

int daqRegisterRead(int fd, unsigned int offset, unsigned int *value);

int daqRegisterWrite(int fd, unsigned int offset, unsigned int value);

int daqRegisterAnd(int fd, unsigned int offset, unsigned int value);

int daqRegisterOr(int fd, unsigned int offset, unsigned int value);

4.7.1.1 Programmed IO functions

The following functions provide PIO transfers to and from the output and input FIFOs. The functions return only when the transfer is completed.

```
int daqFifoReadPio( int fd, unsigned int *data, int ndata );

int daqFifoWritePio( int fd, unsigned int *data, int ndata );
```

4.7.2 PLX PCI Interface Functions

The following functions read and/or write the PLX PCI adapter local registers. Offsets are specified in bytes from the base of the device local register space.

```
int daqPlxRead( int fd, unsigned int offset, unsigned int *value );

int daqPlxWrite( int fd, unsigned int offset, unsigned int value );

int daqPlxAnd( int fd, unsigned int offset, unsigned int value );

int daqPlxOr( int fd, unsigned int offset, unsigned int value );
```

4.7.3 Diagnostics Dump

The following functions dump the registers in simple format or with register specific formatting.

```
int daqDump( int fd );
int daqDumpFormatted( int fd );
```

4.8 Gsclib

The gsclib component provides low level board functions and board specific data conversions.

4.8.1 Clock Divider

The following functions translate between sampling clock divider and the real world sampling rate.

```
float gscClockRate( float clockHz, int ndiv );

int gscClockDivider( float clockHz, float samplerate );
```

4.8.2 Range

The following functions translate between the range setting and the real world full scale voltage

```
int gscRange( float vfs );

float gscVfs( int range );
```

4.8.3 Open, Close, Init

The following functions open and close the device and initialize the hardware.

```
fd = gscOpen_(int nbrd );  // returns file descriptor

void gscClose_( int fd );
```

int gscInit_(int fd);

4.8.4 Digital I/O

The following functions operate the digital I/O interface. The byte argument can be 0 or 1 to select the low or high byte of the digital I/O interface.

int gscDioReset_(int fd);

int gscDioSet_(int fd, int byte, unsigned int val);

int gscDioClear_(int fd, int byte, unsigned int val);

int gscDioRead_(int fd, unsigned int *uval);

4.8.5 Digital I/O

The following functions trigger burst inputs and/or outputs. For the hardware trigger functions, the first digital I/O line is assumed to be connected to the trigger input.

int gscSwInputTrigger_(int fd);
int gscHwTrigger_(int fd);
int gscSwOutputTrigger_(int fd);

4.8.6 Output Buffer

The following function sets the output buffer for circular operation.

int gscSetCircularOutput_(int fd);

The following function sets the end of frame marker in a data set. This should be done before loading it into the output buffer. The higher level routines do this automatically.

void gscMarkEndofFrame(unsigned int *udata, int ndata);

4.8.7 Check data synch markers.

The following function checks the raw data for synchronization and end of frame markers.

int gscChkRaw(unsigned int *raw, int ndata, int ncols);

4.8.8 Data translation between raw and real world values.

The following functions translate between the raw data and real world voltages.

void gscRawtoFloat(float *f, unsigned int *u, int ndata, float vfs);

void gscFloattoRaw(unsigned int *u, float *f, int ndata, float vfs);

The transpose forms translate between column major raw data and row major real world data.

void gscRawtoFloatTranspose(float *f, unsigned int *u, int ndata, int ncols, float vfs);

void gscFloattoRawTranspose(unsigned int *u, float *f, int ndata, int ncols, float vfs);

Single values can be translated by the following functions.

unsigned int raw_float(float r, float vfs)

float float_raw(unsigned int u, float vfs)

4.8.9 Register Dump.

The following function reads the board registers and produces a register-by-register specific formatted dump.

int gscDumpRegisters(unsigned int fd);