Universal Driver Software User Manual

DS-MPE-DAQ0804

PCIe Minicard Data Acquisition Module

For Version 7.0.0 and later

Revision A.0 March 2015

Revision	Date	Comment
A.0	3/17/2015	Initial release

FOR TECHNICAL SUPPORT PLEASE CONTACT:

support@diamondsystems.com

Diamond Systems Corporation
555 Ellis Street
Mountain View, CA 94043 USA
Tel 1-650-810-2500
Fax 1-650-810-2525
www.diamondsystems.com

CONTENTS

1. INTRODUCTION

This user manual contains all essential information about the Universal Driver 7.0 DS-MPE-DAQ0804 demo applications, programming guidelines and usage instructions. This manual also includes the Universal Driver API descriptions with usage examples.

2. HARDWARE OVERVIEW

2.1 Description

DS-MPE-DAQ0804 is a rugged data acquisition PCIe MiniCard module consisting of both analog and configurable digital I/O. The module is ideal for add-on data acquisition I/O expansion in embedded and OEM applications. The PCIe MiniCard offers 8 single ended or four differential 16-bit analog inputs with an aggregate sample rate of 100KHz maximum, 2048 sample A/D FIFO, four 16-bit analog outputs, and 14 digital I/O lines. The buffered digital I/O lines can be optionally configured as either pulse width modulators or counter/timers.

Diamond's Universal Driver 7.0 software provides driver support for all functions. The DS-MPE-DAQ0804 product comes with the PCIe MiniCard data acquisition I/O module, the CK-DAQ0804 cable kit, and a hardware kit with jumpers and screws.

2.2 Specifications

- 8 single ended / 4 differential 16-bit analog inputs
- 100KHz maximum aggregate sample rate
- 2048 sample A/D FIFO with programmable threshold
- 4 analog input ranges: +/-10V, +/-5V, 0-10V, 0-5V
- 4 16-bit analog outputs
- 2 analog output ranges: 0-5V, 0-2.5V
- 14 digital I/O lines, configurable as PWMs or counter/timers:
    - 4 24-bit pulse width modulators
    - 8 32-bit counter/timers
- +3.3VDC input power
- Latching connectors for increased ruggedness
- Support for Windows 7, XP, and Linux 3.2.x
- Universal Driver support for all functions
- PCIe MiniCard full size form factor (50.95mm x 30mm)
- Operating temperature of -40°C to +85°C

3. GENERAL PROGRAMMING GUIDELINES

3.1 Initialization and Exit Function Calls

All the demo applications begin with the following functions and should be called in sequence to initialize the Universal Driver and the board. These functions should be called prior to calling any other DS-MPE-DAQ0804 board specific functions.

1. dscInit ( ), this function initializes the Universal Driver
2. DAQ0804InitBoard(), this function initializes the DS-MPE-DAQ0804 module
3. DSCGetBoardInfo(), this function collects the board information from the Universal Driver and returns the boardinfo structure to be used in the board specific functions

At the termination of any demo application, the user should call the dscfree() function to close the file handles which are opened by the dscInit() function.

These function calls are important in initializing and freeing resources used by the driver. Following is an example of the framework for an application using the driver:

```
#include "DSCUD_demo_def.h"
#include “mpedaq0804.h”

ERRPARAMS errorParams; //structure for returning error code and error string
DSCCBP dsccbp; //structure containing PCI board settings
BoardInfo *bi=NULL; //Structure containing board base address

int main()
{
        if ( (dscInit ( DSC_VERSION ) != DE_NONE) )
        {
                dscGetLastError ( &errorParams );
                printf ( "dscInit error: %s %s\n", dscGetErrorString ( errorParams.ErrCode ),
                errorParams.errstring );
                return 0;
        }
                dsccbp.boardtype = DSC_ MPEDAQ0804;
                dsccbp.pci_slot  = 0;
         if(DAQ0804InitBoard ( &dsccbp  ) !=DE_NONE)
        {
                dscGetLastError(&errorParams);
                printf("DAQ0804InitBoard error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                errorParams.errstring );
                return 0;
        }
        bi = DSCGetBoardInfo ( dsccbp.boardnum );
        /* Application code goes here */
        dscFree ( );
        return 0;
}
```

In the above example, DSC_VERSION, DSC_MPEDAQ0804, and DE_NONE are macros defined in the included header file, *dscud.h file.*

3.2 Error Handling

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

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

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

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

4. UNIVERSAL DRIVER API DESCRIPTION

4.1 DAQ0804ADSetSettings

Function Definition
BYTE DAQ0804ADSetSettings(BoardInfo* bi, DAQ0804ADSETTINGS* settings);

Function Description
This function configures the A/D input range, channel register, and scan settings.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804ADSETTINGS	Structure with following member variables, Range - 0-1; 0 = 5V, 1 = 10V Polarity - 0-1; 0 = bipolar, 1 = unipolar Sedi - 0-1; 0 = single-ended, 1 = differential Sign - 0-1; in Diff mode only: 0 = even channel is high; 1 = odd channel is high Lowch - low channel, 0-7 for SE mode or 0-3 for Diff mode Highch - high channel, 0-7 for SE mode or 0-3 for Diff mode ADClock - 0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit 19; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1 ScanEnable - 0 = disable, 1 = enable scan mode ScanInterval - 0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable; used only if ScanEnable = 1 ProgInt - 100-255, used only if ScanInterval = 3; interval is this value times 100ns

Return Value
Error code or 0

Usage Example
To configure channel zero in 0-5v with single ended and scan disabled,

```
DAQ0804ADSETTINGS dscadsettings;
dscadsettings.Polarity = 1;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = 0;
dscadsettings.Lowch = 0;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 0;
DAQ0804ADSetSettings ( bi, &dscadsettings );
```

4.2 DAQ0804ADSetTiming

Function Definition

BYTE DAQ0804ADSetTiming(BoardInfo* bi, DAQ0804ADSETTINGS* settings);

Function Description

This function configures the A/D clock source and scan settings.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804ADSETTINGS	Structure with following member variables, Range - 0-1; 0 = 5V, 1 = 10V Polarity - 0-1; 0 = bipolar, 1 = unipolar Sedi - 0-1; 0 = single-ended, 1 = differential Sign - 0-1; in Diff mode only: 0 = even channel is high; 1 = odd channel is high Lowch - low channel, 0-7 for SE mode or 0-3 for Diff mode Highch - high channel, 0-7 for SE mode or 0-3 for Diff mode ADClock - 0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit 19; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1 ScanEnable - 0 = disable, 1 = enable scan mode ScanInterval - 0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable; used only if ScanEnable = 1 ProgInt - 100-255, used only if ScanInterval = 3; interval is this value times 100ns

Return Value

Error code or 0.

Usage Example

To configure channels 0-7 in 0-5v with single ended, and programmable scan interval 100, or 10usec:

```
 DAQ0804ADSETTINGS  dscadsettings;
dscadsettings.Polarity = 1;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = 0;
dscadsettings.Lowch = 0;
dscadsettings.ADClock = 0;
dscadsettings, ScanInterval = 3;
dscadsettings; ProgInt = 100;
DAQ0804ADSetTiming(bi, &dscadsettings);
```

4.3 DAQ0804ADSetChannelRange

Function Definition

BYTE DAQ0804ADSetChannelRange(BoardInfo* bi, DAQ0804ADSETTINGS* settings);

Function Description

This function configures the A/D input range. All other settings remain the same. The user is expected to know whether the board is set for a single-ended or differential mode and the odd/sign is set for differential channel polarity.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804ADSETTINGS	Structure with following member variables, Range - 0-1; 0 = 5V, 1 = 10V Polarity - 0-1; 0 = bipolar, 1 = unipolar Sedi - 0-1; 0 = single-ended, 1 = differential Sign - 0-1; in Diff mode only: 0 = even channel is high; 1 = odd channel is high Lowch - low channel, 0-7 for SE mode or 0-3 for Diff mode Highch - high channel, 0-7 for SE mode or 0-3 for Diff mode ADClock - 0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit 19; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1 ScanEnable - 0 = disable, 1 = enable scan mode ScanInterval - 0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable; used only if ScanEnable = 1 ProgInt - 100-255, used only if ScanInterval = 3; interval is this value times 100ns

Return Value

Error code or 0.

Usage Example

To configure channel 0 to 3 in 0-5v with single ended and scan disabled;

```
DAQ0804ADSETTINGS dscadsettings;
dscadsettings.Lowch = 0;
dscadsettings.Highch = 3;
DAQ0804ADSetChannelRange(bi,&dscadsettings);
```

4.4 DAQ0804ADSetChannel

Function Definition

BYTE DAQ0804ADSetChannel(BoardInfo* bi, int Channel);

Function Description

This function configures the A/D circuit for a single channel. All other settings remain the same. The user is expected to know whether the board is set for single-ended or differential mode and the odd/sign setting for differential channel polarity. This function is identical to SetRange except that the same channel is used for the low and high settings, so that the A/D will continuously sample the same channel on successive A/D conversions.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Channel	Input channel, 0-7 for SE mode or 0-3 for Diff mode

Return Value

Error code or 0.

Usage Example

To set the channel number to 2,

```
int channel ;
Channel = 2;
DAQ0804ADSetChannel(bi,Channel);
```

4.5 DAQ0804ADTrigger

Function Definition

BYTE DAQ0804ADTrigger(BoardInfo* bi, unsigned* Sample);

Function Description

This function executes one A/D conversion or scan using the current board settings. It does not perform any configuration of the board but rather uses the current settings. If the board is configured for sample mode (SCANEN = 0), then one A/D conversion will be performed and stored in the sample buffer. If the board is configured for scan mode (SCANEN = 1), then one scan of all channels between 'Highch' and 'Lowch' will be performed and all samples will be stored in the sample buffer.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Sample	pointer to an array of size 8 to hold the return values

Return Value

Error code or 0.

Usage Example

To read samples from 0th channel,

```
unsigned long sample;       // sample reading
DAQ0804ADSETTINGS* dscadsettings;
dscadsettings.Polarity = 1;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = 0;
dscadsettings.Lowch = 0;
dscadsettings.ADClock = 0;
ScanEnable = 0;
DAQ0804ADSetTiming(bi, dscadsettings);
DAQ0804ADTrigger ( bi, &sample );
printf("The samples from channel 0 : 0x%X\n",sample);
```

4.6 DAQ0804ADConvert

Function Definition

BYTE DAQ0804ADConvert(BoardInfo* bi, DAQ0804ADSETTINGS* settings, unsigned* Sample);

Function Description

This function configures the A/D input range, channel register, and scan settings as requested by the user, and then sets the timing to be non-turbo, single-channel (non-scan), and software trigger. Then it takes a single A/D sample of a single channel, triggers the A/D, and reads back the A/D sample value.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804ADSETTINGS	Structure with following member variables, Range - 0-1; 0 = 5V, 1 = 10V Polarity - 0-1; 0 = bipolar, 1 = unipolar Sedi - 0-1; 0 = single-ended, 1 = differential Sign - 0-1; in Diff mode only: 0 = even channel is high; 1 = odd channel is high Lowch - low channel, 0-7 for SE mode or 0-3 for Diff mode Highch - high channel, 0-7 for SE mode or 0-3 for Diff mode ADClock - 0-3: 0 = command bit ADSTART = 1; 1 = falling edge of DIO bit 19; 2 = rising edge of output of counter 0; 3 = rising edge of output of counter 1 ScanEnable - 0 = disable, 1 = enable scan mode ScanInterval - 0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable; used only if ScanEnable = 1 ProgInt - 100-255, used only if ScanInterval = 3; interval is this value times 100ns
Sample	Return value; may be interpreted as unsigned or signed int depending on the input range

Return Value

Error code or 0.

Usage Example

To read a single sample value from channel 0, by configuring the A/D input range as 0-5v and single ended,

```
DAQ0804ADSETTINGS  dscadsettings;
unsigned int sample;       // sample reading
dscadsettings.Polarity = 1;
dscadsettings.Range = 0;
dscadsettings.Sedi = 0;
dscadsettings.Highch = 0;
dscadsettings.Lowch = 0;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 0;
DAQ0804ADConvert ( bi,&dscadsettings, &sample );
                printf("The samples from channel 0 : 0x%X\n",sample);
```

4.7 DAQ0804ADInt

Function Definition

BYTE DAQ0804ADInt(BoardInfo* bi, DAQ0804ADINT* DAQ0804adint);

Function Description

This function enables A/D interrupt operation using the current analog input settings. It configures the FIFO but leaves all other settings alone.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804ADINT	Structure with following member variables, FIFOEnable - 0 = disable; interrupt occurs after each sample / scan; 1 = enable; interrupt occurs on FIFO threshold flag FIFOThreshold - 0-2048, indicates level at which FIFO will generate an interrupt (if FIFOEnable = 1) Cycle - 0 = one shot operation; 1 = continuous operation NumConversions - if Cycle = 0 this is the number of samples / scans to acquire; if Cycle = 1 this is the size of the circular buffer in samples / scans ADBuffer - pointer to A/D buffer to hold the samples

Return Value

Error code or 0.

Usage Example

To enable the A/D interrupt operation,

```
DAQ0804ADINT  dscIntSettings;
dscIntSettings.NumConversions = 1000;
dscIntSettings.Cycle = 1;
dscIntSettings.FIFOEnable = 1;
dscIntSettings.FIFOThreshold = 1000;
dscIntSettings.ADBuffer = (SWORD*) malloc (sizeof(SWORD)*
dscIntSettings.NumConversions );
DAQ0804ADInt(bi,&dscIntSettings );
```

4.8 DAQ0804ADIntStatus

Function Definition

BYTE DAQ0804ADIntStatus(BoardInfo* bi, DAQ0804ADINTSTATUS* intstatus);

Function Description

This function returns the interrupt routine status including running or not running, number of conversions completed, cycle mode, FIFO status, and FIFO flags.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804ADINTSTATUS	Structure with following member variables, OpStatus - 0 = not running, 1 = running NumConversions - Number of conversions since interrupts started Cycle - 0 = one-shot operation, 1 = continuous operation FIFODepth - Current FIFO depth pointer OF, FF, TF, EF - FIFO flags

Return Value

Error code or 0.

Usage Example

To read the interrupt routine status,

```
DAQ0804ADINTSTATUS intstatus;
DAQ0804ADIntStatus(bi,&intstatus)
printf("No of A/D conversions completed %d\n",intstatus.NumConversions);
```

4.9 DAQ0804ADIntPause

Function Definition

BYTE DAQ0804ADIntPause(BoardInfo* bi);

Function Description

This function pauses A/D interrupts by turning off the interrupt enable and stopping the A/D clock. This holds the A/D channel counter and FIFO at their current positions.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value

Error code or 0.

Usage Example

To pauses the A/D interrupts by turning off the interrupt enable and stopping the A/D clock,

```
DAQ0804ADIntPause(bi);
```

4.10 DAQ0804ADIntResume

Function Definition
BYTE DAQ0804ADIntResume(BoardInfo* bi);

Function Description
This function resumes A/D interrupts from the point at which they were paused.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To resumes the A/D interrupts from the point at which they were paused,

```
DAQ0804ADIntResume(bi);
```

4.11 DAQ0804ADIntCancel

Function Definition
BYTE DAQ0804ADIntCancel(BoardInfo* bi);

Function Description
This function stops A/D interrupts by turning off the interrupt enable, stopping the A/D clock, and removing the A/D interrupt handler.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To stops A/D interrupts by turning off the interrupt,

```
DAQ0804ADIntCancel(bi);
```

4.12 DAQ0804DASetSettings

Function Definition
BYTE DAQ0804DASetSettings(BoardInfo* bi, int Range, int Sim);

Function Description
This function is designed to work with the AD5686R D/A converter. This function sets the D/A output ranges, the data format, and the update mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Range	0 = 0-2.5V, 1 = 0-5V
Sim	0 = Individual DAC update mode, 1 = simultaneous update mode

Return Value
Error code or 0.

Usage Example
To set the range as 0 – 5v with simultaneous update mode,

```
int Range;
int sim;
Range = 1;
Sim = 1;
DAQ0804DASetSettings(bi,Range,sim);
```

4.13 DAQ0804DAConvert

Function Definition
BYTE DAQ0804DAConvert(BoardInfo* bi, int Channel, unsigned int DACode);

Function Description
This function outputs a value to a single D/A channel. The output range must be previously set with DASetSettings().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Channel	0-3
DACode	0-65535

Return Value
Error code or 0.

Usage Example
To set channel zero with 5V,

```
Channel = 0;
DACode = 65535;
DAQ0804DAConvert(bi,channel,DACode);
```

4.14 DAQ0804DAConvertScan

Function Definition

BYTE DAQ0804DAConvertScan(BoardInfo* bi, int* ChannelSelect, int* DACodes);

Function Description

This function outputs multiple values to multiple D/A channels. The output ranges must be previously set with DASetSettings().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
ChannelSelect[4]	Array of 4 flags: 0 = don't update channel n, 1 = update channel n
DACodes[4]	Array of 4 D/A values, 0-65535; values where ChannelSelect[n] = 0 are Ignored

Return Value

Error code or 0.

Usage Example

To update channel 0,1 with dacode 65535 ,32768 respectively and not change any other channel,

```
ChannelSelect = (int*)malloc( sizeof( int ) * 4 );
 DACodes = (unsigned int*)malloc( sizeof( unsigned int ) * 4 );
ChannelSelect[0] = 1;
DACodes[0] = 65535;
ChannelSelect[1] = 1;
DACodes[1] = 32768;
DAQ0804DAConvertScan( bi,ChannelSelect, DACodes );
```

4.15 DAQ0804DAFunction

Function Definition
BYTE DAQ0804DAFunction(BoardInfo* bi, unsigned DAData, int DACommand);

Function Description
This function enables the programmer to control the D/A chip directly to implement commands that are not supported by other Universal Driver functions.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAData	16-bit value in straight binary 0000-FFFF
DACommand	8-bit value in straight binary 0-FF

Return Value
Error code or 0.

Usage Example
To perform D/A convert operation by using DAQ0804DAFunction,

```
DACode = 65535;
Channel = 2;
DAcommand =0x10 + (1<<channel) ; //DA update command
DAQ0804DAFunction(bi,DACode,DAcommand);
```

4.16 DAQ0804DAUpdate

Function Definition
BYTE DAQ0804DAUpdate(BoardInfo* bi) ;

Function Description
This function is used to update the D/A when it is set for simultaneous mode (DASIM = 1) and DAConvertScan() function is not being used.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
To update D/A channel 0 and 1 with 65535 and 32768 respectively,

```
Range = 1;
Sim = 1;
DAQ0804DASetSettings(bi,Range,sim);
DAQ0804DAConvert(bi,0,65535);
DAQ0804DAConvert(bi,1,32768);
DAQ0804DAUpdate(bi);
```

4.17 DAQ0804WaveformBufferLoad

Function Definition

BYTE DAQ0804WaveformBufferLoad(BoardInfo* bi, DAQ0804WAVEFORM DAQ0804waveform);

Function Description

This function configures a D/A waveform by copying the waveform data to the board waveform buffer and programming the number of frames into the board.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804WAVEFORM	Structure with following member variables, Waveform - pointer to array of 16-bit unsigned data Frames - total number of frames in the array Framesize - number of channels to be driven by one clock = frame size Channels - list of channels to be driven by the waveform generator

Return Value

Error code or 0.

Usage Example

To generate square wave on channel zero with four DA values ranges from 0- 5V,

```
DAQ0804WAVEFORM DAQ0804waveform;
DAQ0804waveform. Waveform = (int *) malloc (sizeof(int ) *4);
DAQ0804waveform.Waveform[0] = 0;
DAQ0804waveform.Waveform[1] = 0;
DAQ0804waveform.Waveform[2] = 65535;
DAQ0804waveform.Waveform[3] = 65535;
DAQ0804waveform.Frames      =  4;
DAQ0804waveform.FrameSize   = 1;
DAQ0804waveform.Channels [0]=0;
Range = 1;
Sim = 1;
DAQ0804DASetSettings(bi,Range,sim);
DAQ0804WaveformBufferLoad(bi,& DAQ0804waveform);
```

4.18 DAQ0804WaveformConfig

Function Definition

BYTE DAQ0804WaveformConfig(BoardInfo* bi, DAQ0804WAVEFORM DAQ0804waveform);

Function Description

This function configures the operating parameters of the waveform generator, including the clock source, the output frequency if being controlled by a timer, and one-shot or continuous mode. The values in the buffer need to be loaded with DAQ0804WaveformBufferLoad() before starting the waveform generator with DAQ0804WaveformStart().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804WAVEFORM	Structure with following member variables, Frames - total number of frames in the array Clock - 0 = software increment; 1 = counter/timer 0 output; 2 = counter/timer 1 output; 3 = DIO pin 20 Rate - frame update rate, Hz (only used if Clock = 1 or 2) Cycle - 0 = one-shot operation; 1 = repetitive operation

Return Value

Error code or 0.

Usage Example

To configure the waveform generator with 100Hz frequency,

```
DAQ0804WAVEFORM DAQ0804waveform;
DAQ0804waveform.Frames    = 500;
DAQ0804waveform. Clock        = 1;
DAQ0804waveform. Rate       = 100;
DAQ0804waveform. Cycle  = 1;
DAQ0804WaveformConfig(bi,&DAQ0804waveform);
```

4.19 DAQ0804WaveformStart

Function Definition

BYTE DAQ0804WaveformStart(BoardInfo* bi);

Function Description

This function starts or restarts the waveform generator running based on its current configuration.

Prerequisites: DAQ0804WaveformConfig() and DAQ0804WaveformBufferLoad() must have been run previously in order to start the defined waveform.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value

Error code or 0.

Usage Example

```
DAQ0804WaveformStart(bi);
```

4.20 DAQ0804WaveformPause

Function Definition

BYTE DAQ0804WaveformPause(BoardInfo* bi);

Function Description

This function stops the waveform generator. It can be restarted with DAQ0804WaveformStart().

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value

Error code or 0.

Usage Example

```
DAQ0804WaveformPause(bi);
```

4.21 DAQ0804WaveformReset

Function Definition
BYTE DAQ0804WaveformReset(BoardInfo* bi);

Function Description
This function resets the waveform generator and stops all waveform output.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
DAQ0804WaveformReset(bi);
```

4.22 DAQ0804WaveformInc

Function Definition
BYTE DAQ0804WaveformInc(BoardInfo* bi);

Function Description
This function increments the waveform generator by one frame. The current frame of data is output to the selected channels associated to each DAC output code in the buffer frame. This function must be used if "software increment" was selected with DAQ0804WaveformConfig; best used in a conditional loop.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example
```
DAQ0804WaveformInc(bi);
```

4.23 DAQ0804DIOConfig

Function Definition
BYTE DAQ0804DIOConfig(BoardInfo* bi, int Bit, int Dir);

Function Description
This function sets the digital I/O port directions for a single bit.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Bit	DIO line number, 0-20
Dir	0 = input, 1 = output

Return Value
Error code or 0.

Usage Example
To set bit 0 as input mode,

```
Bit = 0;
Dir = 0;
DAQ0804DIOConfig(bi,Bit,Dir);
```

4.24 DAQ0804DIOConfigAll

Function Definition
BYTE DAQ0804DIOConfigAll(BoardInfo* bi, int* Config[]);

Function Description
This function sets the digital I/O port direction for all DIO lines at once.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Config	Array of 3 8-bit configuration values

Return Value
Error code 0.

Usage Example
To set the all digital I/O port direction to input mode,

```
int *config;
config = (int *)malloc(sizeof(int)* 3);
config[0] = 0;
config[1] = 0;
config[2] = 0;
DAQ0804DIOConfigAll(bi,config);
```

4.25 DAQ0804DIOOutputByte

Function Definition
BYTE DAQ0804DIOOutputByte(BoardInfo* bi, int Port, int Data);

Function Description
This function outputs the specified data to the specified port's output register.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = DIO lines 0-7, 1 = DIO lines 8-15, 2 = DIO lines 16-20
Data	8 or 5 bit value to write to the port; for port C only the low 5 bits are used

Return Value
Error code or 0.

Usage Example
To set Port 0 output as 0x77,

```
port=0;
Data=0x77;
DAQ0804DIOOutputByte(bi,port,Data);
```

4.26 DAQ0804DIOInputByte

Function Definition
BYTE DAQ0804DIOInputByte(BoardInfo* bi, int Port, byte* Data);

Function Description
This function reads in the data from the specified port and returns it in the location specified by the pointer passed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Port	0 = DIO lines 0-7, 1 = DIO lines 8-15, 2 = DIO lines 16-20
Data	pointer to receive the data read from the port

Return Value
Error code or 0.

Usage Example
To read port 0 input values and display on the screen,

```
Port=0;
DAQ0804DIOInputByte (bi, Port, &Data);
printf("The PORT 0: 0x%x",Data);
```

4.27 DAQ0804DIOOutputBit

Function Definition
BYTE DAQ0804DIOOutputBit(BoardInfo* bi, int Bit, int Value);

Function Description
This function outputs the desired value to a single digital I/O line. The other bits remain at their current state.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Bit	DIO line number, 0-20
Value	0 or 1

Return Value
Error code or 0.

Usage Example
To set Port 0, bit 6 to 1,

```
Bit=6;
Value=1;
DAQ0804DIOOutputBit ( bi,  Bit,  Value);
```

4.28 DAQ0804DIOInputBit

Function Definition
BYTE DAQ0804DIOInputBit(BoardInfo* bi, int Bit, int* Value);

Function Description
This function reads the specified bit and returns it in the location specified by the pointer passed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Bit	DIO line number, 0-20
Value	Pointer to receive the bit data; return data is always 0 or 1

Return Value
Error code or 0.

Usage Example
To read Port 0, bit 7 and display on the screen,

```
Bit = 7;
DAQ0804DIOInputBit(bi,Bit,&Value);
```

4.29 DAQ0804CounterSetRate

Function Definition

BYTE DAQ0804CounterSetRate(BoardInfo* bi, DAQ0804COUNTER *Ctr);

Function Description

This function programs a counter for timer mode with down counting and continuous operation (reload enabled). The output may be used for waveform generator control, interrupt generation, or for a general programmable frequency output pulse train. The output may also be enabled on a DIO pin. The counter is started immediately. This function is a simplified version of CounterConfig() optimized for the most popular application of rate generator.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804COUNTER	Structure with following member variables, Ctrno - Counter number, 0-7 CtrData - Initial load data, 32-bit straight binary CtrClk - Clock source, 0-3 (see Hardware user manual for usage) CtrCountDir - 0 = down counting, 1 = up counting CtrReload - 0 = one-shot counting, 1 = auto-reload (repetitive counting, only works in count down mode) CtrCmd - Counter command, 0-15 (see Hardware user manual for available commands) CtrOutEn - 1 = enable output onto corresponding I/O pin; 0 = disable output CtrOutPol - 1 = output pulses high, 0 = output pulses low; only used if CtrOutEn = 1 CtrCmdData - Auxiliary data for counter command, 0-3 (see Hardware user manual for usage) Rate - Desired output rate, Hz ctrOutWidth - 0-3 0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks , only used if CtrOutEn = 1 and CtrClock = 2 or 3

Return Value

Error code or 0.

Usage Example

To configure counter 0 with 100Hz, output enabled, polarity high and output pulse width of 1000 clocks,

```
DAQ0804COUNTER counter;
counter.CtrNo = 0;
counter.Rate = 100;
counter.CtrOutEn     = 1;
counter.CtrOutPol    = 1;
counter.ctrOutWidth  =3;
DAQ0804CounterSetRate( bi,&counter);
```

4.30 DAQ0804CounterConfig

Function Definition

BYTE DAQ0804CounterConfig(BoardInfo* bi, DAQ0804CTR *Ctr);

Function Description

This function programs a counter for up or down counting and starts the counter running. A DIO pin may be selected as the input source. If a DIO pin is not selected as the input, and the counter is counting in down mode, the output may be enabled on the DIO pin. Using a DIO pin for input or output causes the FPGA to automatically set the DIO pin direction as needed.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804COUNTER	Structure with following member variables, Ctrno - Counter number, 0-7 CtrData - Initial load data, 32-bit straight binary CtrClk - Clock source, 0-3 (see Hardware user manual for usage) CtrCountDir - 0 = down counting, 1 = up counting CtrReload - 0 = one-shot counting, 1 = auto-reload (repetitive counting, only works in count down mode) CtrCmd - Counter command, 0-15 (see Hardware user manual for available commands) CtrOutEn - 1 = enable output onto corresponding I/O pin; 0 = disable output CtrOutPol - 1 = output pulses high, 0 = output pulses low; only used if CtrOutEn = 1 CtrCmdData - Auxiliary data for counter command, 0-3 (see Hardware user manual for usage) Rate - Desired output rate, Hz ctrOutWidth - 0-3 0 = 1 clocks, 1 = 10 clocks, 2 = 100 clocks, 3 = 1000 clocks , only used if CtrOutEn = 1 and CtrClock = 2 or 3

Return Value

Error code or 0.

Usage Example

To configure counter 0 with 100Hz, counter direction down, auto reload and output disabled,

```
DAQ0804COUNTER Ctr;
Ctr.Ctrno = 0;
Ctr.CtrClk = 2; //50MHz
Ctr.CtrData = 50000000/100;
Ctr.CtrCountDir = 0;
Ctr.CtrReload   = 1;
Ctr.CtrOutEn    = 0;
DAQ0804CounterConfig(bi,&Ctr);
```

4.31 DAQ0804CounterRead

Function Definition

BYTE DAQ0804CounterRead(BoardInfo* bi, int CtrNo, unsigned long * CtrData);

Function Description

This function latches a counter and reads the value.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
CtrNo	Counter number, 0-7
CtrData	pointer to storage location for return data, unsigned 32-bit value

Return Value

Error code or 0.

Usage Example

To read current value of counter 0 when it is running and display on the screen,

```
unsigned long CtrData;
ctrNo = 0;
DAQ0804CounterRead ( bi, CtrNo, &CtrData);
printf("The counter value %d ",CtrData);
```

4.32 DAQ0804CounterFunction

Function Definition

BYTE DAQ0804CounterFunction(BoardInfo* bi, DAQ0804CTR* Ctr);

Function Description

This function can be used to program any desired function into a counter, except reading

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804CTR	Structure with following member variables, Ctrno - Counter number, 0-7 CtrData - Initial load data, 32-bit straight binary CtrCmd - Counter command, 0-15 (see Hardware user manual for available commands) CtrCmdData - Auxiliary data for counter command, 0-3 (see Hardware user manual for usage)

Return Value

Error code or 0.

Usage Example

To reset the counter 0,

```
DAQ0804CTR Ctr;
Ctr.Ctrno          = 0;
Ctr.CtrCmd         = 0xF; // reset;
Ctr.CtrCmdData  = 0; // reset only selected counter;
Ctr.CtrData        = 0xF; // reset
DAQ0804CounterFunction(bi,&Ctr);
```

4.33 DAQ0804PWMConfig

Function Definition

BYTE DAQ0804PWMConfig(BoardInfo* bi, DAQ0804PWM* PWM);

Function Description

This function configures a PWM for operation.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
PWM	Structure with following member variables, Num - PWM number, 0-3 Rate - Output frequency in Hz Duty - Initial duty cycle, 0.0-100.0 Polarity - 0 = pulse high, 1 = pulse low OutputEnable - 0 = disable output, 1 = enable output on DIO pin Run - 0 = don't start PWM, 1 = start PWM

Return Value

Error code or 0.

Usage Example

To configure PWM 0 with 100 Hz, 50% duty cycle and output enabled,

```
DAQ0804PWM PWM;
PWM.Num = 0;
PWM.Rate = 100;
PWM.Duty = 50.0;
PWM.Polarity = 1;
PWM.OutputEnable   = 1;
DAQ0804PWMConfig(bi,&PWM);
```

4.34 DAQ0804PWMStart

Function Definition

BYTE DAQ0804PWMStart(BoardInfo* bi, int Num);

Function Description

This function starts a PWM running.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	PWM number, 0-3

Return Value

Error code or 0.

Usage Example

To start PWM 0,

```
Num = 0;
DAQ0804PWMStart(bi,Num);
```

4.35 DAQ0804PWMStop

Function Definition
BYTE DAQ0804PWMStop(BoardInfo* bi, int Num);

Function Description
This function stops a PWM.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Num	PWM number, 0-3

Return Value
Error code or 0.

Usage Example
To stop PWM 0,

```
Num = 0;
DAQ0804PWMStop(bi,Num);
```

4.36 DAQ0804PWMCommand

Function Definition
BYTE DAQ0804PWMCommand(BoardInfo* bi, DAQ0804PWM* DAQ0804pwm);

Function Description
This function is used to modify a PWM configuration.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804pwm	Structure with following member variables, Num - PWM number, 0-3 Command - 0-15 = PWM command CmdData - 0 or 1 for auxiliary PWM command data (used for certain commands) Divisor - 24-bit value for use with period and duty cycle commands

Return Value
Error code or 0.

Usage Example
To stop all PWMs,

```
DAQ0804PWM DAQ0804pwm;
DAQ0804pwm.Command = 0x00; //Stop PWM
DAQ0804pwm.CmdData = 0x00; //Stop all PWM
DAQ0804PWMCommand(bi,& DAQ0804pwm);
```

4.37 DAQ0804UserInterruptConfig

Function Definition

BYTE DAQ0804UserInterruptConfig(BoardInfo* bi, DAQ0804USERINT* DAQ0804userint);

Function Description

This function installs a pointer to a user function that runs when interrupts occur.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804USERINT	Structure with following member variables, IntFunc - pointer to user function to run when interrupts occur Mode - 0 = alone, 1 = before standard function, 2 = after standard function Source - Selects interrupt source: 0 = A/D, 1 = unused, 2 = counter 2 output,3 = counter 3 output, 4 = digital I/O Enable - 0 = disable interrupts, 1 = enable interrupts BitSelect - 0-20 selects which DIO line to use to trigger interrupts; only used if Source = 4 Edge - 1 = rising edge, 0 = falling edge; only used if Source = 4

Return Value

Error code or 0.

Usage Example

To install a function to be called whenever interrupt occurs,

```
void intfunction()
{
    printf("My function called ");
}
void main()
{
        DAQ0804USERINT DAQ0804userint;
        DAQ0804userint.IntFunc = intfunction;
        DAQ0804userint.Mode = 0;
        DAQ0804userint.Source = 2; // Requires counter 2 already setup
        DAQ0804userint.Enable = 1;
        DAQ0804UserInterruptConfig(bi,& DAQ0804userint);
}
```

4.38 DAQ0804UserInterruptRun

Function Definition
BYTE DAQ0804UserInterruptRun(BoardInfo* bi, int Source, int Bit, int Edge);

Function Description
This function is used to start user interrupts when they are running in Alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Source	Identifies interrupt source: 2 = counter 2 output, 3 = counter 3 output, 4 = digital input
Bit	0-20 selects which DIO bit will drive interrupts
Edge	0 = falling edge, 1 = rising edge

Return Value
Error code 0.

Usage Example
To start the user interrupt,

```
Source = 4;
Bit = 0;
Edge = 0;
DAQ0804UserInterruptRun(bi, Source, Bit, Edge );
```

4.39 DAQ0804UserInterruptCancel

Function Definition
BYTE DAQ0804UserInterruptCancel(BoardInfo* bi, int Source);

Function Description
This function is used to cancel user interrupts when they are running in Alone mode.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Source	Identifies interrupt source: 2 = counter 2 output, 3 = counter 3 output, 4 = digital input

Return Value
Error code or 0.

Usage Example
To cancel the user interrupts,

```
Source = 2;
DAQ0804UserInterruptCancel(bi,Source);
```

4.40 DAQ0804InitBoard

Function Definition
BYTE DAQ0804InitBoard(DSCCB* dsccb);

Function Description
This function initializes the board.

Function Parameters

Name	Description
dsccb	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example

```
DSCCBP dsccbp; // structure containing board settings
dsccbp.boardtype = DSC_MPEDAQ0804;
dsccbp.pci_slot  = 0;
DAQ0804InitBoard ( &dsccbp );
```

4.41 DAQ0804FreeBoard

Function Definition
BYTE DAQ0804FreeBoard(DSCB board);

Function Description
This function stops any active interrupt processes and frees the interrupt resources assigned to a board.

Function Parameters

Name	Description
board	The handle of the board to operate on

Return Value
Error code or 0.

Usage Example

```
DAQ0804FreeBoard(dsccbp.boardnum);
```

4.42 DAQ0804LED

Function Definition

BYTE DAQ0804LED(BoardInfo* bi, int enable);

Function Description

This function turns the on-board LED ON or OFF.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
Enable	0 = turn off, 1 = turn on

Return Value

Error code or 0.

Usage Example

To turnoff blue LED on the board,

```
Enable=0;
DAQ0804LED(bi, Enable);
```

4.43 DAQ0804FIFOStatus

Function Definition

BYTE DAQ0804FIFOStatus(BoardInfo* bi, DAQ0804FIFO* Fifo);

Function Description

This function returns the current FIFO threshold, depth and status flags in the parameter array.

Function Parameters

Name	Description
BoardInfo	The handle of the board to operate on
DAQ0804FIFO	Structure with following member variables, Threshold - Current FIFO programmed threshold Depth - Current FIFO depth pointer Enable - 0 = FIFO is not currently enabled, 1 = FIFO is currently enabled OF - 0 = no overflow, 1 = FIFO overflow (attempt to write into FIFO when FIFO was full) FF - 0 = FIFO not full, 1 = FIFO is full TF - 0 = number of A/D samples in FIFO is less than the programmed threshold, 1 = number of A/D samples in FIFO is equal to or greater than the programmed threshold EF - 0 = FIFO has unread data in it, 1 = FIFO is empty

Return Value

Error code or 0.

Usage Example

To read the current FIFO threshold, OF, FF, TF, EF and display on the screen,

```
DAQ0804FIFO* Fifo;
DAQ0804FIFOStatus(bi,&Fifo);
printf("OF : %d\n",Fifo.OF);
printf("FF : %d\n",Fifo.FF);
printf("TF : %d\n",Fifo.TF);
printf("EF : %d\n",Fifo.EF);
printf("Depth : %d\n",Fifo.Depth);
```

5. UNIVERSAL DRIVER APPLICATION DESCRIPTION

The Universal Driver supports the following functions on the DS-MPE-DAQ0804:

- DA Convert
- DA Convert Scan
- DA Waveform
- DIO
- Counter Function
- Counter Set Rate
- PWM
- User Interrupt
- AD Sample
- AD Sample Scan
- AD Trigger
- AD Interrupt
- LED

5.1 DA Convert

This function outputs a value to a single D/A channel. The output range can be selected from user input. When a D/A conversion is performed, it takes a digital value and sends it out to the specified analog output channel as a voltage. This output code can be translated to one of the output voltages described in the hardware specification.

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

5.2 DA Convert Scan

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

5.3 DA Waveform

This function generates a desired waveform on selected DA channel. It configures a D/A channel by copying the waveform values to the board's waveform buffer. The waveform generator replays a loaded waveform from the internal FPGA memory. The digital signal is converted into an analog output signal with a defined offset and amplitude based on the D/A values sent. Any waveform can be replayed including a previously acquired waveform or a calculated or simulated waveform.

5.4 DIO

Digital I/O supports four types of direct digital I/O operation: input bit, input byte, output bit, and output byte.

5.5 Counter Function

Generally the counter is used as rate generator. The counter also can be configured in count-up or count-down direction. The counter function can be used to program any desired function into a counter except reading, which is accomplished with the counter reading function.

5.6 Counter Set Rate

This function programs a counter for timer mode with down counting and continuous operation (reload enabled). The output may be used for waveform generator control, interrupt generation, or for a general programmable frequency output pulse train. The output is enabled by using a DIO pin.

5.7 PWM

This application generates pulse width modulation (PWM) signals. PWM signals are a method for generating analog signals using a digital source. A PWM signal consists of two main components that define its behavior: duty cycle and frequency. The duty cycle describes the amount of time the signal is in a high (on) state as a percentage of the total time it takes to complete one cycle. The frequency determines how fast the PWM completes a cycle (i.e. 1000Hz would be 1000 cycles per second), and therefore how fast it switches between high and low states. By cycling a digital signal off and on at a fast enough rate, and with a certain duty cycle, the output will appear to behave like a constant voltage analog signal when providing power to devices. This program gets duty cycle and frequency value from the user and generates PWM signals.

5.8 User Interrupt

The User Interrupt feature of the Universal Driver enables the user to run personal code when a hardware interrupt is generated by an I/O board. This is useful for applications that require special operations to be performed in conjunction with the interrupt or applications where the code has to be run at regular fixed intervals. Universal Driver includes example programs for each board with user interrupt capability to illustrate how to use this feature. This application gets interrupt frequency as user input and calls the user function periodically at the rate of interrupt frequency.

5.9 AD Sample

Performs a single A/D conversion on the currently selected channel and it will return a digital reading of an analog voltage signal applied to the currently selected analog input channel. The A/D board uses a device called an analog-to-digital (A/D) converter to convert the real-world analog voltage transducer signal (temperature, pressure, tank level, speed, etc.) into a digital value that the digital computer electronics can process. This digital value can be translated back to the input voltage using the conversion formulas provided in the board's hardware user manual.

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

5.10 AD Sample Scan

A/D scan is similar to an A/D sample except that it performs several conversions on a specified range of input channels with each function call.

5.11 AD Trigger

An A/D trigger is similar to A/D sample. It performs a single A/D conversion on the currently selected channel and it will return a digital reading of an analog voltage signal applied to the currently selected analog input channel. This is useful for fast re-triggering of the A/D converter with the same settings.

5.12 AD Interrupt

The A/D interrupt performs A/D scans using interrupt-based I/O with one scan per A/D clock tick. Note that calling this function only starts the interrupt operations in a separate system thread. The function call does not result in an atomic transaction with immediate results. Rather, it starts a real-time process that terminates only when the number of conversions reaches the maximum specified (one-shot mode) or if a call to DAQ0804ADIntCancel() is made.

The behavior of A/D interrupt operations depends on three parameters. Some affect the behavior of the hardware, and some affect the behavior of the interrupt service routine (ISR) software function. Together they determine the overall characteristics of the interrupt operation.

5.13 LED

This function is used to turn the on-board LED on or off.

6. UNIVERSAL DRIVER APPLICATION USAGE INSTRUCTIONS

The following section details about how to use the Universal Driver applications and to confirm the output which may be obtained through the application.

6.1 DA Convert

This application gets input from the user as follows:

- Enter Channel Number(0 – 3,Default 0):
- Enter output range (0-1; 0 = 0-2.5V,1 = 0-5V,default 1):
- Enter output code (0-65535, default:32768):

To set channel 0 with 5v using range 1, run DA Convert application and provide input as follows:

Channel Number=0
Range=1
Output code=65535

Measure the voltage on channel 0 using a multi-meter. It should show 5V, the application generated expected voltage.

6.2 DA Concert Scan

This application gets input from the user as follows:

- Enter output range (0 = 0-2.5V,1 = 0-5V):
- Enter Sim value (0 =Individual Update 1=Simultaneous update):
- Enter the channel enable flag for channel 0
  (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 0 (0-65535):
- Enter the channel enable flag for channel 1
  (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 1 (0-65535):
- Enter the channel enable flag for channel 2
  (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 2 (0-65535):
- Enter the channel enable flag for channel 3
  (0 for FALSE, 1 for TRUE; default: 1):
- Enter the output code for channel 3 (0-65535):

To set all channels with 5v using range 1, run DA Convert scan application and provide input as follows:

- Enter output range (0 = 0-2.5V,1 = 0-5V): 1
- Enter Sim value (0 =Individual Update 1=Simultaneous update): 1
- Enter the channel enable flag for channel 0
  (0 for FALSE, 1 for TRUE; default: 1): 1
- Enter the output code for channel 0 (0-65535): 65535
- Enter the channel enable flag for channel 1
  (0 for FALSE, 1 for TRUE; default: 1): 1
- Enter the output code for channel 1 (0-65535): 65535
- Enter the channel enable flag for channel 2
  (0 for FALSE, 1 for TRUE; default: 1): 1
- Enter the output code for channel 2 (0-65535): 65535
- Enter the channel enable flag for channel 3

(0 for FALSE, 1 for TRUE; default: 1): 1

- Enter the output code for channel 3 (0-65535): 65535

Measure the voltage on all channels using a multi-meter. Each should show 5V, the application generated expected voltage.

6.3 DA Waveform

This application gets input from the user as follows:

- Enter DA-OutPut Channel Number (0-3):
- Enter Range value (0-1; 0 = 0-2.5V, 1 = 0-5V):
- Enter waveform size (1-1023) :
- Enter Clock source (0-3 , 0 -Manual, 1-Counter 0 output, 2-Counter 1 output , 3-

  External trigger on DIO pin 20):

- Select waveform type (0-Sine wave 1-Sawtooth Wave) :
- Enter waveform frequency :
- Enter waveform mode(0-1, 0-One shot 1-Repeat mode):

To generate a sine wave form on channel zero with 100Hz frequency and range from 0-5V, run the wave form application and provide input as follows:

Channel Number=0

Range=1

Waveform size = 500

Clock source = 1

Waveform type = 0; //sine wave

Waveform frequency = 100

Waveform mode = 1

Place an oscilloscope probe on D/A channel zero and set voltage division to 1V range and second division to 1ms. It should show a sine wave with 100Hz frequency, the application generated expected wave form.

6.4 DIO

The DIO application provides various operations on a DIO channel such as input byte, output byte, input bit, output bit, and DIO loopback. This section describes the input byte and output byte DIO operations. The DIO port must be configured in either input or output mode based on the DIO operation to be performed.

Output Byte:

- Select Write a Byte to port option from main menu
- Enter value 0-255 or q to quit

To set Port 0 all the pins to high except pin 3 and pin 7, run the DIO application and provide input as follows:

- Select Write a Byte to port option from main menu :4
- Enter value 0-255 or q to quit: 119

The Byte value 119 is sent to port 0.

Measure the voltage on all pins of Port 0 using a multi-meter. They should show 3.3v on all the pins except pin 3 and pin 7, the application generated expected voltages.

Input Byte:

- Select Read a Byte from port option from main menu:
- Enter port number (0-2):
- Press ENTER key to stop reading

Provide 3.3V to Port 0 pin 0 from VCC. It should read and display 0x01. To see the output, run the DIO application and provide input as follows:

- Select Read a Byte from port option from main menu:1
- Enter port number (0-2):0

The application should show 0x01 on the screen.

6.5 Counter Function

This application gets input from the user as follows:

- Enter counter number (0-7):
- Enter Counter Direction (0 = down counting, 1 = up counting):
- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz):
- Enter Output Enable (0=disable, 1=Enable):
- Enter Output Polarity (0=negative, 1= positive):
- Enter Counter Frequency :
- Press any key to stop counting.

To generate a 100Hz rate generator using counter 0, run the Counter Function and provide input as follows:

- Enter counter number (0-7): 0
- Enter Counter Direction (0 = down counting, 1 = up counting): 0
- Select Clock source (0=External ,2=Internal clock 50MHz,3=Internal clock 1MHz): 2
- Enter Output Enable (0=disable, 1=Enable): 1
- Enter Output Polarity (0=negative, 1= positive): 1
- Enter Counter Frequency :100

Place a oscilloscope probe on counter 0 output pin (DIO 6th pin is output pin for counter0) and set the voltage division to 1V range and second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key and the application stops the counter output

6.6 Counter Set Rate

This application gets input from the user as follows:

- Enter counter number (0-7):
- Enter Counter frequency rate (1-50MHz):
- Enter Output Enable (0=disable, 1=Enable):
- Enter Output Polarity (0=negative, 1= positive):
- Press any key to stop counting

To generate a 100Hz rate generator using counter 0, run the Counter Set Rate application and provide input as follows:

- Enter counter number (0-7):0
- Enter Counter frequency rate (1-50MHz):100
- Enter Output Enable (0=disable, 1=Enable):1
- Enter Output Polarity (0=negative, 1= positive):1

Place an oscilloscope probe on counter 0 output pin (DIO 6th pin is output pin for counter0) and set the voltage division to 1V range and second division to 1ms. It should show a periodic pulse with 100Hz frequency, the application generated expected rate.

- Press any key and the application stops the counter output

6.7 PWM

This application gets input from the user as follows:

- Select PWM no (0-3):
- Select Output Frequency (1-50MHz):
- Select Duty cycle value (1-100):
- Select Polarity (0 = pulse high, 1 = pulse low):
- Waits for key press
- Output is generated
- If any key is pressed, application Stops PWM output.

To generate a 100Hz PWM waveform with 50% duty cycle on PWM channel 0, run the PWM application and provide input as follows:

- Select PWM no (0-3): 0
- Select Output Frequency (1-50MHz): 100
- Select Duty cycle value (1-100): 50
- Select Polarity (0 = pulse high, 1 = pulse low): 0
- Press any key to start PWM

Place an oscilloscope probe on PWM channel 0 output pin (DIO 14th pin is output pin for PWM0) and set the voltage division to 1V range and second division to 1ms. It should show a PWM wave form with 50% duty cycle and 100Hz frequency, the application generated expected rate.

- Press any key and the application stops the PWM output

6.8 User Interrupt

This application gets input from the user as follows:

- Enter Interrupt source (2-4; 2 = counter 2 output, 3 = counter 3 output, 4 = digital I/O):
- Enter Interrupt source frequency rate (1-50MHZ):
- It calls user function based interrupt rate
- Press any key to cancel the application:

This application installs a function where a count value is incremented by one whenever the function gets called. To confirm the user function is getting called as per the interrupt rate, run the UserInt application and provide input as follows:

- Enter Interrupt source (2-4; 2 = counter 2 output, 3 = counter 3 output, 4 = digital I/O):2
- Enter Interrupt source frequency rate (1-50MHZ): 100

It calls the user function based on the interrupt rate and prints the count value every second. Since it is configured for 100Hz it displays the count value as follows:

UserInt count value =0

UserInt count value =100

UserInt count value =200

UserInt count value =300

Press any key to cancel the interrupt.

6.9 AD Sample

This application gets input from the user as follows:

- Enter channel number(0-7):
- Enter A/D polarity (0 for bipolar, 1 for unipolar):
- Enter A/D full scale value (0 = 5V, 1 = 10V):
- Enter input mode (0 = single-ended, 1 = differential):
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit 19):

To perform this operation for channel 0 with a 5v range, run the ADSample application and provide an external voltage (5v) to channel 0, then provide inputs as follows:

- Enter channel number (0-7):0
- Enter A/D polarity (0 for bipolar, 1 for unipolar):1
- Enter A/D full scale value (0 = 5V, 1 = 10V):0
- Enter input mode (0 = single-ended, 1 = differential):0
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit 19):0

It will perform the sampling operation and display the output as follows:

Sample readout: 32768, Actual voltage: 5.0v.

6.10 AD Sample Scan

This application gets input from the user as follows:

- Enter the low channel (0-7, default:0):
- Enter the High channel (0-7, default:7):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):
- Press any key to stop scanning A/D channel

To perform the the AD Scan application for channel 0-4 with bipolar 5v range, run the ADScan application and provide the inputs as follows:

- Enter the low channel(0-7, default:0):0
- Enter the High channel (0-7, default:7):4
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):0
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):0
- Enter input mode (0 = single-ended, 1 = differential; default 0):0
- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):0

The application scans AD channels from 0 to 4 and displays the AD sample values.

6.11 AD Trigger

This application gets input from the user as follows:

- Enter channel number (0-7, default: 0):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit 19), default 0):

To perform this operation for channel 0 with 5v range, run the ADSample application and provide an external voltage (5v) to channel 0 and inputs as follows:

- Enter channel number (0-7, default: 0):0
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):1
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):0
- Enter input mode (0 = single-ended, 1 = differential; default 0):0
- Enter Clock source (0 = Manual, 1 = falling edge of external trigger (DIO bit 19), default 0):0

It will perform the sampling operation and display the output as follows:

Sample readout: 32768, Actual voltage: 5.0v.

6.12 AD Interrupt

This application gets input from the user as follows:

- Enter low channel value (0-7, default: 0):
- Enter High channel value(0-7, default:7):
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):
- Enter input mode (0 = single-ended, 1 = differential; default 0):
- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):
- Enter A/D clock source (1-3, 1= External trigger 2=Counter 0, 3=Counter 1 default=2) :
- Enter A/D sampling Rate (Default 1000) :
- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1000):
- Enter the cycle flag (0 = one shot operation, 1 = continuous operation, default 1):
- Enter FIFO Enable bit value (0 = disable, 1 = enable, default: 1) :
- Enter FIFO Threshold value (0-2048 default: 1000) :
- Press Space bar key to Pause/Resume A/D Int or Press any other key to stop A/D Interrupt

To perform the AD Interrupt application for channel 0-4 with bipolar 5V range and sampling rate of 1000Hz, run the A/D interrupt application and provide inputs as follows:

- Enter low channel value (0-7, default: 0):0
- Enter High channel value(0-7, default:7):4
- Enter A/D polarity (0 for bipolar, 1 for unipolar, default: 0):0
- Enter A/D full scale value (0 = 5V, 1 = 10V; default 0):0
- Enter input mode (0 = single-ended, 1 = differential; default 0):0

- Enter Scan Interval (0 = 10us, 1 = 12.5us, 2 = 20us, 3 = programmable, default 0):0
- Enter A/D clock source (1-3, 1= External trigger 2=Counter 0, 3=Counter 1 default=2) :2
- Enter A/D sampling Rate (Default 1000) :1000
- Enter number of A/D conversions (must be multiple of FIFO threshold value default 1000):1000
- Enter the cycle flag (0 = one shot operation, 1 = continuous operation, default 1):1
- Enter FIFO Enable bit value (0 = disable, 1 = enable, default: 1) :1
- Enter FIFO Threshold value (0-2048 default: 1000) :1000

The application scans AD channels from 0 to 4 at 1000Hz interrupt rate and displays the AD sample values.

6.13 LED

This application gets input from user as follows:

- Press space bar key to toggle LED or Press 'q' to quit

To toggle the on board LED, press the space bar key to disable the LED. Press the same key again to enable the LED.

7. COMMON TASK REFERENCE

7.1 Data Acquisition Feature Overview

I/O connectors

The DS-MPE-DAQ0804 provides two I/O connectors for the attachment of all user I/O signals. The connectors are JST number BM20B-GHDS-G-TF. Diamond's I/O cable number 6980501 (2 per board) is used to connect the user signals to these connectors. This cable comes in a kit of two as part number CK-DAQ02. Each cable has a common 2mm pitch IDC connector at the opposite end which may be plugged into a custom board or may be cut off and the wiring then used as needed. Unused A/D signals should be grounded to avoid floating values. If the cable wiring is cut, care should be taken to avoid shorting any unused wires to any other voltages in the system in order to prevent possible damage to the board or incorrect analog I/O readings.

Analog inputs

There are 8 analog input channels, numbered 0-7, located on connector J3 pins 7-14. In single-ended mode, each pin is an individual channel. The voltage on each channel is measured relative to the reference analog ground, which must be connected to any of pins 5, 6, 15, and 16. In differential mode, each consecutive pair of pins forms a high-low pair; for example pins 7-8, or channels 0-1, are treated as the high and low inputs of a single channel. The input voltage is measured as the difference between these two pins. On the DS-MPE-DAQ0804, the definition of which pin is high and which pin is low is selectable by the user with the "sign" parameter.

The maximum difference between any voltage on any analog input pin and the analog ground pins (common mode voltage) is 10V. In single-ended mode, as long as the input signal is within the range of +/-10V, the A/D will be able to measure the signal accurately. In differential mode, both inputs must be within this range. If the input voltages exceed this range, errors will occur, since the A/D will treat anything outside this range as the limit voltage, i.e. either +10V or -10V. This limit is not precise and should not be used in any normal operation. Furthermore any long term excursion by an input voltage beyond +/-10V may cause damage to the A/D chip or the on-board analog power supply.

Analog outputs

There are 4 analog output channels, numbered 0-3, located on connector J3 pins 1-4. Analog outputs operate in single-ended mode only and are always referenced to the analog ground, which must be connected to any of pins 5, 6, 15, and 16.

Waveform generator

The board contains a 4-channel analog waveform generator using the analog outputs. One to four channels may operate simultaneously. The waveforms are stored in an on-board data buffer that holds 2048 samples. Any number of samples can be used up to the 2048 limit. If more than one channel is being used, then each channel's waveform must have the same number of samples, and the maximum length of each waveform is 2048 divided by the number of channels in use.

The waveform generator can be clocked in multiple ways, including by software command, external digital signal on I/O connector J3 pin 19 (digital I/O port 20), or on-board counter/timer 0 or 1. On each clock, one "frame" of data will be output, consisting of one data point for each channel being used.

The maximum frame output rate depends on the number of channels in use and is shown in the table below:

Channels	Max frame rate
1	
2	
3	
4	930Hz

If one channel per frame is being used, an arbitrary sequence of channel/data pairs can be entered by the user. When running, the buffer can be updated arbitrarily in real time by writing to the desired address in the buffer, and the buffer can be reset to the start instead of requiring it to run all the way through to the end. The buffer is never cleared, instead it can be overwritten with new data as desired, and the user is responsible for maintaining congruence between the data in the buffer and its operation.

Digital I/O signals

There are 21 digital I/O signals, numbered 0-20. Signals 0-17 are on I/O connector J2 pins 1-18, and signals 18-20 are on I/O connector J3 pins 17-19. Digital I/O signals use 3.3V signaling only. Each signal's direction is independently programmable. On system startup or reset, all signals are automatically set to input mode.

All digital I/O signals have programmable pull-up/down resistors and are divided into two groups for this purpose. Group A includes I/O signals 0-15, and Group B includes I/O signals 16-20. All signals within each group have the same pull direction. The default configuration is for all DIO signals to be pulled low.

Counter/timers

The board provides 8 32-bit counter/timers. Counter mode means the circuit will count external events that are connected via one of the digital I/O lines. Timer mode means the circuit will generate output pulses at a user-specified rate. The pulse width is programmable for both polarity (high or low pulse) and width (1, 10, 100, or 1000 clock pulses). The clock for timer mode is provided internally from an on-board 50MHz oscillator. This oscillator is divided by 50 to provide a 1MHz clock for very low pulse rates. The available range of output rates is 50MHz / 20ns (50MHz / 1) to .0002328Hz / 4295 sec (1MHz / 2^32).

The counter/timers use the digital I/O lines for their I/O signals. When a counter/timer is programmed to use external clock or output, the associated digital I/O line is taken over for the counter/timer and its direction is set as needed to support the selected function. The I/O pin may be assigned as either an input (clock) to the counter/timer or an output. The I/O pin assignment is as follows:

Connector J2 Pin	DIO Bit	Counter/Timer
7	6	0
8	7	1
9	8	2
10	9	3
11	10	4
12	11	5
13	12	6
14	13	7

Pulse Width Modulators (PWMs)

The board offers 4 24-bit PWMs. Each PWM may be programmed for output frequency, duty cycle, and output polarity. Duty cycle is defined as the percentage of time the output signal will have the indicated polarity during each period. For example, a 1KHz output frequency (1ms period) with 20% duty cycle and positive output polarity will exhibit a repetitive waveform that is high for 0.2ms at the start of the period

and low for 0.8ms during the remainder of the period. Each PWM contains 2 counters. Counter C0 controls the output frequency, and counter C1 controls the duty cycle.

The PWMs use the digital I/O lines for their output signals. When a PWM is running and its output is enabled, the associated digital I/O line is taken over to be used as the output for the PWM, and its direction is forced to output. The I/O pin assignment is as follows:

Connector J2 Pin	DIO Bit	PWM
15	14	0
16	15	1
17	16	2
18	17	3

7.2 Data Acquisition Software Task Reference

This section describes the various data acquisition tasks that may be performed with the DS-MPE-DAQ0804 and gives step by step instructions on how to use them using the Universal Driver functions. Tasks include:

- Program entry / exit sequence
- A/D conversions
- A/D interrupts
- D/A conversions
- Waveform generator
- Digital I/O
- Counter/timer operation
- PWM operation
- User interrupts

Program Entry/Exit sequence

1. Each program that uses the Universal Driver must first call function dscInit only once.
2. The DAQ0804InitBoard() function must be called prior to any other function involving the DS-MPE-DAQ0804.
3. At the termination of the program the programmer may use DAQ0804FreeBoard(), but this is not required. This function is normally used in a development environment where the program is being repeatedly modified and rerun.
4. Function dscFree() will release any other resources used by the driver upon exit of the program.

A/D conversion operation

Each A/D conversion is triggered by a clock event. The clock event may be a software command, an external digital signal on I/O connector J3 pin 18 (digital I/O bit 19), or the output of either counter 0 or counter 1. Generally, low-speed conversions and “on-demand” conversions are triggered by software or by an external signal, and high speed conversions (where a precise time interval is required between samples) are driven by a counter/timer. If an external signal is used, it is connected to I/O connector J3 pin 18.

High-speed conversions are generally controlled with an interrupt-based A/D function in order to reduce the software overhead. The application program sets up the A/D circuit as needed and then calls the A/D interrupt function. The sampling and data transfer to memory occur in a background process. The application can monitor the progress of A/D conversions and retrieve data from the memory buffer as needed.

A/D conversions fall into two basic modes; sample and scan. In sample mode, a single channel is sampled on each clock. If the user is sampling multiple channels, then each clock will cause a single A/D conversion to occur on the currently selected channel, and then the channel counter will increment to the next channel in the list to be ready for the next clock. In scan mode, each clock causes one A/D conversion to occur on each channel in the user-selected channel range. The channel range can consist of 1 to 8 channels and must be consecutive, for example 0-3 or 4-7. Note that a scan operation on a single channel is equivalent to a sample operation on that channel.

In A/D scan operations, the time interval between samples (scan interval) is selectable from a range of options, as described in the Function Descriptions. Three fixed intervals and one user-programmable interval are provided. Generally the programmer will want to use the fastest available scan interval to complete all samples as closely together in time as possible. In cases where the input signals are using high input ranges such as 0-10V or +/-10V, using a longer interval may result in higher accuracy, since the input circuit has more time to swing from the current channel to the next. Most of Diamond's A/D boards are designed to provide accuracy close to the specified performance using the smallest scan interval for A/D input ranges of 0-5V or less.

Full A/D conversion sequence

The full sequence of operations is the same for A/D sample and A/D scan operations:

1. DAQ0804ADSetSettings() is used to select the input type, input voltage range, and input channel range.
2. DAQ0804ADSetTiming() is used to select the A/D clock source and the scan mode if desired.
3. If software A/D clocking is selected, then DAQ0804ADTrigger() is used to generate A/D conversions. The function must be called once for each A/D sample. The board will auto-increment within the selected input channel range, starting with the lowest numbered channel in the range. When the highest numbered channel has been sampled, the board will reset to the lowest numbered channel. The function will return the A/D value from the currently sampled channel in A/D counts. This number must be converted to a voltage using the formulas described in the DS-MPE-DAQ0804 hardware user manual. The programmer may also convert this number to whatever engineering units are appropriate for the application.
4. If external clocking or counter/timer clocking is selected, the A/D conversions will start as soon as the selected clock source becomes active. In this case, the A/D FIFO will start to fill up with samples, and the program should use the A/D interrupt functions described below to acquire the data from the FIFO.

Quick mode for a single A/D conversion

For software-triggered A/D conversions, the entire operation can be executed with a single function, DAQ0804ADConvert(). The function returns the A/D value from the selected channel using the selected input type and input range.

Shortcuts

To change the input channel range but retain all other existing settings, use the following sequence:

1. DAQ0804ADSetChannelRange() to select the new channel range
2. DAQ0804ADTrigger() to execute single A/D conversions, one channel at a time

To select a single channel but retain all other existing settings, use the following sequence:

1. DAQ0804ADSetChannel() to select the single channel
2. DAQ0804ADTrigger() to execute single A/D conversions

A/D Interrupt Operations

For high speed or externally clocked A/D conversions, interrupts should be used. This method installs an interrupt handler as a background task to read data from the board and store it in the program's data buffer. The A/D conversions can be triggered by a software command, a falling edge on I/O connector J3 pin 18 (digital I/O bit 19), or the output of either counter 0 or counter 1.

A/D data is stored in a FIFO on the board, incrementing the FIFO depth counter each time. When the depth in the FIFO reaches the user-selected threshold, an interrupt will occur, and the interrupt handler will read out the data in the FIFO, decrementing the FIFO depth counter. This process occurs repeatedly until stopped.

The program must select the FIFO threshold based on two opposing parameters: The waiting time until the first data is available (threshold divided by sample rate) and the desired interrupt rate (sample rate divided by threshold). Generally the FIFO threshold should be selected to avoid exceeding a 1KHz interrupt rate. Higher interrupt rates consume more processor time, so applications may wish to reduce the interrupt rate even lower, to 100-200Hz, by using a deeper threshold if the sample rate permits it.

Note that A/D data will only be transferred from the board to the user's data buffer when the FIFO reaches the selected threshold. This means that once starting the interrupt operation, the program will have no data until the first interrupt occurs.

Generally the sample rate for interrupt operations is high, so the initial waiting time is not significant, and the desired interrupt rate will drive the selection of the FIFO threshold value.

Interrupt-based A/D conversions can be done in two modes, one-shot or continuous. One-shot means that a fixed number of A/D conversions is done and then the operation automatically stops. Continuous means that the A/D sampling continues until the program stops the operation.

The sequence of operations for interrupt-based A/D conversions is as follows:

1. DAQ0804ADSetSettings() is used to select the input type, input voltage range, and input channel range.
2. DAQ0804ADSetTiming() is used to select the clock source.
3. DAQ0804ADInt() is used to install the interrupt handler.
4. If a counter/timer is being used to control A/D timing, then DAQ0804CounterSetRate() is used to program the counter/timer for the desired sample rate.
5. DAQ0804ADSetTiming() is used to select the A/D clock source (counter/timer or external signal) and the scan mode if desired. A/D conversions will start as soon as the selected clock source becomes active, and the data will automatically be transferred to the user-specified data buffer based on the selected FIFO threshold.
6. To monitor A/D operations, use DAQ0804ADIntStatus().

The interrupt operation can be monitored with the DAQ0804ADIntStatus() function. This function will report whether the interrupt operation is running, how many samples have been taken since the start, the current FIFO depth, and the type of operation – single or recycle.

D/A conversion operation

This section discusses single D/A conversions on one or more channels. For waveform generator operations, see the separate D/A waveform generator section.

D/A conversions can be performed on one or more channels at a time. When operating on multiple channels, the program has the option of selecting single channel update or multi-channel simultaneous update. Simultaneous update is useful in certain applications where two or more parameters need to change simultaneously, for example when driving a laser from point (x1, y1) to point (x2, y2). In this type

of application it is obviously preferable to move the laser from point 1 to point 2 in a straight line rather than in two orthogonal lines, one in the X direction and one in the Y direction.

For single channel output, use the following sequence:

1. DAQ0804DASetSettings() to select the output range and simultaneous update mode if desired
2. DAQ0804DAConvert() to update a single channel with the given value

For multi-channel output with individual channel update:

1. DAQ0804DASetSettings() to select the output range; set Sim = 0
2. DAQ0804DAConvertScan() to update the selected channels with the given data

For multi-channel output with simultaneous update:

1. DAQ0804DASetSettings() to select the output range; set Sim = 1
2. DAQ0804DAConvertScan() to load the given data into the selected channels
3. DAQ0804DAUpdate() to update all channels at the same time

Waveform generator

Using the waveform generator involves a series of operations:

1. Create the waveform data buffer and download it to the board
2. Configure the waveform generator
3. Start (and restart) the waveform generator
4. Pause or reset the waveform generator

To create the waveform buffer, first compute the waveform or load the data from a file. The data is in binary form using numbers in the range 0 – 65535 (0 – 2^16-1). If more than one channel will be used for waveform generation, each waveform must be the same length, and the data for all channels must be interleaved in the buffer, so that each consecutive group of data values represents one “frame” of data. For example, a 2-channel waveform generator using D/A channels 0 and 1 would have its data buffer organized like this:

Buffer address	Channel
0	0
1	1
2	0
3	1
4	0
5	1
…	…

The total number of samples cannot exceed 2048 for a single channel or 2048 / <number of channels> for multiple channels. The term <number of channels> is also referred to as the frame size.

Once the data buffer is built, use DAQ0804WaveformBufferLoad() to download the entire buffer to the board at one time. The buffer must not be larger than 2048 samples, and the formula <frame size> x <number of frames> must be less than or equal to 2048.

DAQ0804WaveformDataLoad() can be used to update a single data point in the waveform buffer at any time, including while the waveform generator is running.

Once the buffer is downloaded, use DAQ0804WaveformConfig() to configure the clock source, clock rate (for internal clocking), and one-shot or continuous operation. In one-shot operation, the waveform generator will make a single pass through the data buffer and output the data one time, then automatically stop. In continuous operation, the waveform generator will output the waveform(s) repeatedly until stopped with a software command.

To start the waveform, use DAQ0804WaveformStart(). After this function is called, the waveform generator will run in response to clocks from the selected source.

To pause the waveform generator at any time in its current position, use DAQ0804WaveformPause(). The waveform may be restarted in its current position by using DAQ0804WaveformStart() again.

To reset the waveform generator, use DAQ0804WaveformReset(). This stops the waveform generator function, however the data buffer still retains its data. After the reset function is called, to restart the waveform generator function DAQ0804WaveformConfig() followed by DAQ0804WaveformStart() must be called.

If software increment is selected, the waveform is incremented with the function DAQ0804WaveformInc(). Each time this function is called, the waveform generator will output one frame of data, i.e. one data value to each channel in use.

Digital I/O operation

Digital I/O operation is relatively simple. First configure the DIO port direction with one of the below functions:

BYTE DAQ0804DIOConfig() configures a single bit for input or output.
BYTE DAQ0804DIOConfigAll() configures all 21 bits at once.

Then execute whichever I/O function is desired. Byte read/write enables 5 or 8 bits of digital I/O to be updated at once. Bit operation enables a single bit to be updated.

BYTE DAQ0804DIOOutputByte() outputs 8 bits of data
BYTE DAQ0804DIOInputByte(BoardInfo* bi, int Port, byte* Data);
BYTE DAQ0804DIOOutputBit(BoardInfo* bi, int Bit, int Value);
BYTE DAQ0804DIOInputBit(BoardInfo* bi, int Bit, int* Value);

To configure the digital I/O pull-up/down resistors, use DAQ0804DIOConfig(). The programmed value is stored in a small flash device on the board, so that the board will retain the latest configuration the next time it is powered up.

Counter/timer operation

The counter/timers are configured using a series of commands to control individual features. Driver functions provide shortcuts to quickly configure the counters for common counting and timing operations. For non-standard or specialized operations, the individual commands can be used to configure the counter/timers exactly as desired.

Simplified Programming

To program a counter/timer as a rate generator with a specific frequency, use DAQ0804CounterSetRate(). This function is also used to set the sampling rate for interrupt-based A/D conversions. The counter is programmed for down counting, and an external clock is selected. The counter output may optionally be enabled onto a digital I/O pin, with programmable polarity and pulse width.

To program a counter/timer for counting operation, use the following functions:

1. Use DAQ0804CounterConfig() to configure the counter for either up or down counting and start the counter running. A Digital I/O pin may be selected for either the input or the output (but not both). This function is typically used to count external events.
2. Use DAQ0804CounterRead() to read the current contents of the counter. This function can be used repeatedly to monitor the operation. This is normally used with event counting.
3. When the counting function is no longer needed, use DAQ0804CounterReset() to reset the counter and return any assigned Digital I/O pin to normal digital I/O operation.

Detailed programming

To program a counter/timer using individual commands, use DAQ0804CounterFunction(). This function must be used multiple times to execute each command needed to configure the counter. All commands take effect immediately upon execution. The typical command sequences for the most common operations are provided below. See the full list of counter/timer commands in the appendix.

For a rate generator:

Command	Function
15	Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired.
1	Load counter with desired divisor to select the desired output pulse rate. The output rate is the selected clock frequency divided by the divisor.
2	Select the count direction. For a rate generator the direction should be down.
6	Select clock source. Normally an internal clock (50MHz or 1MHz) will be selected.
7	Enable auto-reload. This means that the counter will operate continuously.

If the rate generator output is desired, use the following two commands:

8	Enable counter output on the associated digital I/O pin. The desired output polarity is also selected with this command.
9	Select the desired output pulse width.

Finally, enable the counter/timer with the following command:

4	Start the counter/timer running. (This function is also used to stop the counter/timer.)

When the rate generator is no longer needed, either of the following commands can be used:

4	Stop the counter/timer running. The existing settings are maintained so the counter can be restarted later if desired. If it was assigned for the output pulse, the digital I/O line is still tied to the counter/timer and cannot be used for normal digital I/O operations.
15	Reset the counter/timer. This stops the counter/timer and releases the digital I/O line back to normal digital I/O operation.

Alternatively, the function DAQ0804CounterReset() can be used to reset the counter/timer.

For an event counter:

15	Reset the counter/timer. This function should be used first to reset any previous configurations to ensure the counter/timer operates exactly as desired. This will reset the counter data register to 0.
2	Select the count direction. For an event counter the direction should be up.

6 Select clock source. Normally the associated digital I/O pin will be selected to enable counting external pulses.

7 Enable or disable auto-reload. If auto-reload is enabled, the counter will operate continuously, meaning that when it reaches 2^32-1 it will roll over to zero. In most cases auto-reload will be disabled for event counting.

4 Start the counter/timer running. (This function is also used to stop the counter/timer).

While the counter is operating, its current count can be read by using the DAQ0804CounterRead() function. When the counting function is no longer needed, the function DAQ0804CounterReset() can be used to reset the counter/timer.

PWM operation

The PWMs are configured using a series of commands to control individual features. Driver functions provide shortcuts to quickly configure them for common operations. For non-standard or specialized operations, the individual commands can be used to configure the PWMs exactly as desired.

To configure and start a PWM:

1. DAQ0804PWMConfig() configures the selected PWM for output frequency, duty cycle, and polarity. The PWM may optionally be started as well.
2. DAQ0804PWMStart() can be used to start the PWM running if the config function did not start it.

To stop a PWM:

DAQ0804PWMStop() stops a PWM from running. The output is driven to the inactive state. For a PWM with positive output polarity, the output will go low.

To restart a PWM that has been stopped:, use DAQ0804PWMStart().

To reset a PWM and return its assigned digital I/O output pin to normal operation, use DAQ0804PWMReset().

To implement special functions, such as changing the duty cycle or frequency of a PWM while it is running, use DAQ0804PWMCommand(). This function must be executed multiple times, once for each command, to carry out the desired configuration. The available commands are listed in the appendix. All commands take effect immediately upon execution.

User interrupts

Universal Driver enables the installation of user-defined code to be run when an interrupt occurs. The interrupt can be triggered by a variety of sources. The interrupt can run as the only procedure when the interrupt occurs (standalone or alone mode) or it can run before or after the driver's built-in interrupt function (before and after modes). The available modes depend on the source of the interrupt:

Source	Source number	Modes supported
A/D interrupts	0	1 before, 2 after
D/A interrupts	1	Not supported on DS-MPE-DAQ0804
Counter/timer interrupts	2, 3	0 Alone
Digital input interrupts	4	0 Alone

User interrupts are very easy to use. Just 3 steps are required: Configure, run, and stop.

Configure:

DAQ0804UserInterruptConfig() selects the source for the user interrupts and also installs a pointer to the user's code to run when the interrupt occurs. If user interrupts are being run in Before or After mode, this function must be called before the function that initiates the standard interrupt function (e.g. before the sequence described in the A/D interrupts section).

Run (alone mode):

1. If a counter/timer is being used to drive interrupts, then configure it with DAQ0804CounterSetRate().
2. If a digital input is being used to drive interrupts, it is configured with DAQ0804UserInterruptConfig().
3. Use DAQ0804UserInterruptRun() to start the interrupt operation.

Run (before / after modes):
Call the standard interrupt setup function, such as DAQ0804ADInt().

Stop (alone mode):
Use DAQ0804UserInterruptCancel() to stop user interrupts.

Stop (before / after modes):
Use the standard interrupt cancel function such as DAQ0804ADIntCancel().

LED control

The DS-MPE-DAQ0804 contains a blue LED that is user-programmable. This can be used as a visual indication that the board is responding to commands. Turn the LED on and off, use DAQ0804LED().

7.3 Performing D/A Conversion

Description

When a D/A conversion is performed, essentially a digital value is taken and converted to a voltage value that is sent out to the specified analog output. This output code can be translated to an output voltage using one of the equations described in the hardware manual.

The Universal Driver function for performing a D/A conversion is DAQ0804DAConvert().

Step-By-Step Instructions

Call DAQ0804DAConvert()and pass the channel number and output code value. This generates a D/A conversion on the selected channel that will output the new voltage that is set with the output code.

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

Example of usage for D/A conversion

```
BoardInfo *bi;
```

```
int channel;
int range;
int dasim;
unsigned long int DACode;

channel = 0;
range = 0;  /* 0 - 2.5V  */
dasim = 0;
DACode = 65535;
/* Step 1 */
if (DAQ0804DASetSettings ( bi,range,dasim ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf("DAQ0804DASetSettings error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
/* Step 2 */
if( (DAQ0804DAConvert ( bi, channel, DACode ) ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf("DAQ0804DAConvert error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
```

7.4 Performing D/A Conversion Scan

Description

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

The Universal Driver function for performing the D/A conversion scan is DAQ0804DAConvertScan().

Step-By-Step Instructions

Pass the values for ChannelSelect and DACodes pointer to this function.

Call DAQ0804DAConvertScan() and pass it a pointer to the scan array values - this will generate a D/A conversion for each channel that is set to enable.

Example of Usage for D/A Conversion Scan

To update channel 0 and 2 with DA code 65535 and 32768respectively and the rest of the channels not to be changed from existing voltage level:

```
ChannelSelect = (int*)malloc( sizeof( int ) * 4 );
DACodes = (unsigned int*)malloc( sizeof( unsigned int ) * 4 );
ChannelSelect[0] = 1;
DACodes[0] = 65535;

ChannelSelect[2] = 1;
DACodes[2] = 32768;
if ( (DAQ0804DAConvertScan( bi,ChannelSelect, DACodes ) ) != DE_NONE )
```

```
{
        dscGetLastError(&errorParams);
        printf("DAQ0804DAConvertScan error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
```

7.5 Performing Digital IO Operations

Description

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

The six Universal Driver functions described here are DAQ0804DIOConfig() , DAQ0804DIOConfigAll(), DAQ0804DIOOutputByte(), DAQ0804DIOInputByte(), DAQ0804DIOOutputBit(), DAQ0804DIOInputBit().

Step-By-Step Instructions

If a digital input is being performed, create and initialize a pointer to hold the returned value of type byte*. Some boards have digital I/O ports with fixed directions, while others have ports with programmable directions. Make sure the port is set to the required direction, input or output. Use function DAQ0804DIOConfigAll() to set the direction if necessary or use function DAQ0804DIOConfig() to set the direction for a single bit.

Call the selected digital I/O function. Pass it an int port value, and either a pointer to or a constant byte digital value. If performing bit operations, then pass in an int value telling the driver which particular bit (0-7) of the DIO port which is wished to be operated.

Example of usage for digital I/O operation

```
...
BoardInfo *bi;
int *config;
int port;
BYTE input_byte;              // the value ranges from 0 to 255
BYTE  output_byte;
int digital_value;
config = (int *)malloc(sizeof(int)* 3);

/* 1. Configure Port 0 in output mode */
        config[0] = 0;  //0 = Input, 1 = Output
        config[1] = 1;  //0 = Input, 1 = Output
        config[2] = 0;  //0 = Input, 1 = Output
        if(result = DAQ0804DIOConfigAll(bi,config) != 0)
          return result;

/* 2. input bit - read bit 6 (port 0 is in input mode) */
        port = 0;
        bit = 6; //DIO line 0-7 are port 0
        if ( (DAQ0804DIOInputBit ( bi,bit, &digital_value ) != DE_NONE) )
        {
                dscGetLastError(&errorParams);
                printf("DAQ0804DIOInputBit error: %s %s\n",
                dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

/* 3. input byte - read all 8 bits of port 0 (port 0 is in input mode) */
        port = 0;
        if( (DAQ0804DIOInputByte ( bi, port, &input_byte ) != DE_NONE) )
```

```
        {
                dscGetLastError(&errorParams);
                printf("DAQ0804DIOInputByte error: %s %s\n",
                dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

/* 4. output bit – set the bit 6 of port 1 (assumes port 1 is in output mode) */
        bit = 14; //port 0(8bits) + port1 6th bit
        digital_val = 1;
        if ( (DAQ0804DIOOutputBit(bi,bit,digital_val) != DE_NONE) )
        {
                dscGetLastError(&errorParams);
                printf("DAQ0804DIOOutputBit error: %s %s\n",
                dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

/* 5. output byte – set the port 1  to “0xFF”(assumes port 1 is in output mode) */
        Port = 1;
        output_byte = 0xFF;
        if( (DAQ0804DIOOutputByte( bi, port, output_byte ) != DE_NONE) )
        {
                dscGetLastError(&errorParams);
                printf("DAQ0804DIOOutputByte error: %s %s\n",
                dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }
```

7.6 Performing PWM Operations

Description

The PWM operation generates a PWM signal with desired frequency and duty cycle value. The following functions are used for PWM operation: DAQ0804PWMConfig() , DAQ0804PWMStart() and DAQ0804PWMStop().

Step-By-Step Instructions

Create DAQ0804PWM structure variable to hold PWM settings and initialize PWM structure variables, then call DAQ0804PWMConfig() to configure the PWM, call the DAQ0804PWMStart() to start the PWM and finally call DAQ0804PWMStop()to stop the running PWM signal.

Example of usage for PWM operation

```
DAQ0804PWM pwm; // structure to hold the PWM settings
pwm.Num = 0;  //select PWM channel
pwm.Rate = 100; // Select Output Frequency
pwm.Duty = 50; // Select Duty cycle value
pwm.Polarity = 0; // Select Polarity value
pwm.OutputEnable = 1; //Enable PWM output
pwm.Run = 0;

//The following function configures PWM circuit
if(DAQ0804PWMConfig(bi,&pwm) !=DE_NONE)
{
      dscGetLastError(&errorParams);
      printf("DAQ0804PWMConfig error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring );
      return 0;
            }

//The following function start the PWM
if(DAQ0804PWMStart(bi,pwm.Num) !=DE_NONE)
{
      dscGetLastError(&errorParams);
      printf("DAQ0804PWMStart error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring );
      return 0;
            }
      printf("Press any key to stop PWM \n");
      while( !kbhit())
           {
           }

//The following function stop the PWM
if(DAQ0804PWMStop(bi,pwm.Num) !=DE_NONE)
     {
            dscGetLastError(&errorParams);
            printf("DAQ0804PWMStop error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring );
            return 0;}
```

7.7 Performing Counter Function Operations

Description:

Generally the Counter is used as rate generator, Also the counter can be configured in count-up or count-down direction. The following functions are used for counter function operator DAQ0804CounterConfig() and DAQ0804CounterFunction().

Step-By-Step instructions

Create DAQ0804COUNTER structure variable to hold the counter settings and initialize the counter structure variables then call DAQ0804CounterConfig() to configure the counter and finally DAQ0804CounterFunction()to stop the running counter.

Example of usage for counter operations

```
DAQ0804COUNTER Ctr;
Ctr. CtrNo = 0;
Ctr. CtrClk = 2; //50MHz
Ctr. CtrData = 50000000/100;
Ctr. CtrCountDir = 0;
Ctr. CtrReload     = 1;
Ctr.CtrOutEn = 1;
counter.CtrOutPol = 1;
//The following function configures the counter with desired rate
if(DAQ0804CounterConfig(bi,&Ctr) !=DE_NONE)
{
       dscGetLastError(&errorParams);
       printf("DAQ0804CounterConfig error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
       errorParams.errstring );
                      return 0;
}
printf("Press any key to stop counting \n");
while( !kbhit())
{
       dscSleep(1000);
       DAQ0804CounterRead(bi,counter.CtrNo ,&counter.CtrData);
       printf("Counter Data %ld \r",counter.CtrData);
       fflush(stdout);
}
counter.CtrCmd = 0xF;
counter.CtrCmdData = 0;
if(DAQ0804CounterFunction(bi,&counter) !=DE_NONE)
{
        dscGetLastError(&errorParams);
        printf("DAQ0804CounterFunction error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
```

7.8 Performing Counter Set Rate Operation

Description:

Generally the counter is used as rate generator, but also the counter can be configured in count-up or count-down direction. The following functions are used for counter set rate operation DAQ0804CounterSetRate(), DAQ0804CounterRead() and DAQ0804CounterFunction().

Step-By-Step instructions

Create DAQ0804COUNTER structure variable to hold Counter settings and initialize counter structure variables then call DAQ0804CounterSetRate() to programs a counter for timer mode with down counting and continuous operation (reload enabled) , then call DAQ0804CounterRead() to reads the counter value and finally call DAQ0804CounterFunction() to stop the running counter.

Example of usage for counter set rate operations

```
DAQ0804COUNTER counter;
counter.CtrNo = 0;
Rate = 1000;   // Counter frequency
counter.CtrOutEn = 1;
counter.CtrOutPol = 1;

if(DAQ0804CounterSetRate(bi,counter.CtrNo,Rate,counter.CtrOutEn,counter.CtrOutPol) !=DE_NONE)
{
        dscGetLastError(&errorParams);
        printf("DAQ0804CounterSetRate error: %s %s\n",
        dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
        return 0;
}
printf("Press any key to stop counting \n");
while( !kbhit())
{
        dscSleep(1000);
        DAQ0804CounterRead(bi,counter.CtrNo,&ctrData);
        printf("Counter Data %ld \r",ctrData);
        fflush(stdout);
}
counter.CtrCmd = 0xF;
counter.CtrCmdData = 0;
if(DAQ0804CounterFunction(bi,&counter) !=DE_NONE)
{
         dscGetLastError(&errorParams);
         printf("DAQ0804CounterFunction error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
         errorParams.errstring );
         return 0;
}
```

7.9 Performing User Interrupt Operations

Description:

The user Interrupt application configures counter 0 to generate the desired interrupt rate and at the same interrupt rate, a registered user interrupt function is called. To do a user Interrupt application the following Universal Driver functions are used DAQ0804UserInterruptConfig(), DAQ0804UserInterruptRun() and DAQ0804UserInterruptCancel().

Step-By-Step instructions

Create DAQ0804USERINT structure variable to hold Interrupt settings and initialize Interrupt structure variables, then call DAQ0804UserInterruptConfig() to configure the counter, call the DAQ0804UserInterruptRun() to run the user interrupt and finally call DAQ0804UserInterruptCancel() to stop the user interrupt.

Example of usage for user interrupt operations

```
int count =0;
Void intfunction()
{
        Count ++;
}
Void main()
{
     DAQ0804USERINT DAQ0804userint;
     DAQ0804userint. IntFunc = intfunction;
     DAQ0804userint. Mode = 0;
     DAQ0804userint. Source = 2;
     DAQ0804userint.Enable = 1;

   if(DAQ0804UserInterruptConfig(bi,& DAQ0804userint) !=DE_NONE)
  {
      dscGetLastError ( &errorParams );
       printf ( "DAQ0804UserInterruptConfig error: %s %s\n", dscGetErrorString ( errorParams.ErrCode ),
       errorParams.errstring );
       return 0;
 }
 Bit = 0;
 Edge = 0;
 if(DAQ0804UserInterruptRun(bi, DAQ0804userint. Source,Bit,Edge) !=DE_NONE)
{
        dscGetLastError ( &errorParams );
        printf ( "DAQ0804UserInterruptRun error: %s %s\n", dscGetErrorString (
        errorParams.ErrCode ), errorParams.errstring );
                    return 0;
}
        printf("Press any key to terminate the application \n");
        while( !kbhit())
{
         dscSleep(1000);
         printf("Count value %d \r",count);
         fflush(stdout);
}
 DAQ0804userint.Enable= 0;
 if(DAQ0804UserInterruptCancel(bi, DAQ0804userint.Source) !=DE_NONE)
 {
```

```
        dscGetLastError ( &errorParams );
        printf ( "DAQ0804UserInterruptCancel error: %s %s\n", dscGetErrorString (
        errorParams.ErrCode ), errorParams.errstring );
                return 0;
 }

}
```

7.10 Generating D/A Waveform

Description:

This function generates a desired waveform on selected DA channel. It configures a D/A channel by copying the waveform values to the board waveform buffer. The following Universal driver function are used to generate a waveform: DAQ0804WaveformReset(), DAQ0804WaveformConfig(), DAQ0804WaveformBufferLoad(), DAQ0804WaveformStart(), DAQ0804WaveformPause(), and DAQ0804WaveformReset().

Step-By-Step instructions

Create DAQ0804WAVEFORM structure variable to hold waveform settings and initialize waveform structure variables then call DAQ0804WaveformReset() to reset the D/A waveform buffer, call DAQ0804WaveformConfig() to configures the operating parameters of the waveform generator, call DAQ0804WaveformBufferLoad() to loads D/A conversion value into the D/A waveform buffer, call DAQ0804WaveformStart() to starts the waveform generator and finally call DAQ0804WaveformPause() and DAQ0804WaveformReset() to stop the waveform generator.

Example of usage for D/A waveform generator

```
DAQ0804WAVEFORM waveform;
waveform.Frames = 500;
waveform.Clock =1;
waveform.FrameSize=1;
waveform.Cycle =1;
frequency = 100;
waveform.Rate = waveform.Frames
channel = 0;
range = 1; //0-5v
if( DAQ0804WaveformReset(bi) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ("RMM1616WaveformReset error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
if( DAQ0804WaveformConfig(bi, &waveform ) != DE_NONE )
{
         dscGetLastError(&errorParams);
         printf ( "RMM1616WaveformConfig error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
         errorParams.errstring );
         return 0;
}
/* Loading D/A conversion values to waveform buffer */
waveform.Waveform =(unsigned int*) malloc(waveform.Frames * sizeof(unsigned int));
waveform.Channels=(int *) malloc(4 * sizeof(int *));
```

```
for(index=0;index <waveform.Frames;index++)
{
/* generating sine wave */
waveform.Waveform[index]= (int) ((sin( index * (360.0/waveform.Frames) * PI/180) + 1) * ( 0xFFFF/2)) ;
}
for( index=0;index<4;index++)
{
        waveform.Channels[index]= channel;
 }
if ( DAQ0804DASetSettings ( bi,range, 0) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ("DAQ0804DASetSettings error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
//the following loads the D/A conversion value into the D/A buffer.
if( DAQ0804WaveformBufferLoad(bi,&waveform) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ("DAQ0804WaveformBufferLoad error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
//start D/A conversions
printf("Starting DA wave form generator \n");
if( DAQ0804WaveformStart(bi) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ("DAQ0804WaveformStart error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
                return 0;
}
printf("Press any key to stop wave form Generator \n");
fgets(input_buffer,20,stdin);
DAQ0804WaveformPause(bi);
DAQ0804WaveformReset(bi);
free(waveform.Waveform);
free(waveform.Channels);
```

7.11 Performing A/D Sample

Description:

When performing an A/D conversion, you are getting a digital reading of an analog voltage signal applied to one of the A/D board's analog input channels. The following Universal Driver function is used for A/D conversion: DAQ0804ADConvert().

Step-By-Step instructions

Create DAQ0804ADSETTINGS structure variable to hold A/D settings and initialize structure variables then call DAQ0804ADConvert() to configures the A/D settings as requested by the user, and It then takes a single A/D sample value of a single channel.

Example of usage for A/D sample:

```
DAQ0804ADSETTINGS dscadsettings;
unsigned int sample;       // sample reading
dscadsettings.Highch = 0;
dscadsettings.Lowch = 0;
dscadsettings.Polarity = 0;
dscadsettings.Range = 0; // 5V
dscadsettings.Sedi = 0;
dscadsettings.ScanEnable = 0;
dscadsettings.ADClock = 0;
if ( (DAQ0804ADConvert ( bi,&dscadsettings, &sample ) ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ( "DAQ0804ADConvert error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
```

7.12 Performing A/D Sample Scan

Description:

A/D sample scan is similar to an A/D sample except that it performs several conversions on a specified range of input channels with each function call. The Universal Driver functions for performing an A/D scan are DAQ0804ADSetSettings(),DAQ0804ADSetTiming(),DAQ0804ADTrigger().

Step-By-Step instructions

Create DAQ0804ADSETTINGS structure variable to hold A/D settings and initialize structure variables then call DAQ0804ADSetSettings() configures the A/D settings as requested by the user, call DAQ0804ADSetTiming() to configures the turbo mode, A/D clock source, and scan settings and finally call DAQ0804ADTrigger() to executes A/D conversion scan using the current board settings.

Example of usage for A/D sample scan:

```
DAQ0804ADSETTINGS dscadsettings;
unsigned int sample[8];       // sample reading
dscadsettings.Lowch=0;
dscadsettings.Highch=7;
dscadsettings.Polarity = 0;
dscadsettings.Range = 0; //5v
dscadsettings.Sedi = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 1;
if ( ( DAQ0804ADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ("DAQ0804ADSetSettings error: %s %s\n",
        dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
        return 0;
}
if ( (DAQ0804ADSetTiming(bi,&dscadsettings) ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ( "DAQ0804ADSetTiming error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
while(!kbhit() )
{
        if ( (DAQ0804ADTrigger ( bi, sample ) ) != DE_NONE )
                {
                        dscGetLastError(&errorParams);
                        printf ( "DAQ0804ADTrigger error: %s %s\n",
                           dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                        return 0;
                }
}
```

7.13 Performing A/D Trigger

Description:

A/D Trigger is similar to A/D sample. When performing an A/D conversion, you are getting a digital reading of an analog voltage signal applied to one of the A/D board's analog input channels.

The following Universal Driver functions are used for A/D trigger: DAQ0804ADSetSettings(), DAQ0804ADSetTiming(), and DAQ0804ADTrigger().

Step-By-Step instructions:

Create DAQ0804ADSETTINGS structure variable to hold A/D settings and initialize structure variables then call DAQ0804ADSetSettings() to configures the A/D settings as requested by the user, call DAQ0804ADSetTiming() to configure the turbo mode, A/D clock source, and scan settings and finally call DAQ0804ADTrigger() to executes one A/D conversion using the current board settings.

Example of usage for A/D trigger:

```
DAQ0804ADSETTINGS dscadsettings;
unsigned int sample;       // sample reading
dscadsettings.Lowch=0;
dscadsettings.Highch=0;
dscadsettings.Polarity = 0;
dscadsettings.Range = 0; //5v
dscadsettings.Sedi = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 0;
dscadsettings.ScanEnable = 0;
if ( ( DAQ0804ADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
      dscGetLastError(&errorParams);
      printf ("DAQ0804ADSetSettings error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring );
      return 0;
}
if ( (DAQ0804ADSetTiming(bi,&dscadsettings) ) != DE_NONE )
{
      dscGetLastError(&errorParams);
      printf ( "DAQ0804ADSetTiming error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
      errorParams.errstring );
      return 0;
}
if ( (DAQ0804ADTrigger ( bi, &sample ) ) != DE_NONE )
{                   dscGetLastError(&errorParams);

                    printf ( "DAQ0804ADTrigger error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
                    errorParams.errstring );
                                return 0;

}
```

7.14 Performing A/D Interrupts

Description:

The AD interrupt application Performs A/D scans using interrupt-based I/O with one scan per A/D clock tick. The following Universal driver functions are used for A/D interrupt:_DAQ0804ADSetSettings(), DAQ0804ADSetTiming(), DAQ0804ADInt() , DAQ0804ADIntStatus(), and DAQ0804ADIntCancel() .

Step-By-Step instructions:

Create DAQ0804ADSETTINGS structure variable to hold A/D settings, create DAQ0804ADINT structure variable to hold A/D conversion interrupt settings, create DAQ0804ADINTSTATUS structure variable to hold A/D conversion interrupt status. Initialize A/D settings structure variables then call DAQ0804ADSetSettings() configures the A/D settings as requested by the user, call DAQ0804ADSetTiming() to configures the turbo mode, A/D clock source, and scan settings. Initialize A/D interrupt status structure variable then call DAQ0804ADInt() to enables A/D interrupt operation using the current analog input settings and call DAQ0804ADIntStatus() to get the interrupt routine status including, running / not running, number of conversions completed, cycle mode, FIFO status, and FIFO flags, call DAQ0804ADIntResume() to resume the interrupt, call DAQ0804ADIntPause() to pause the interrupt and finally call DAQ0804ADIntCancel() to stop the A/D interrupt.

Example of usage for A/D interrupt:

```
DAQ0804ADSETTINGS dscadsettings; // structure containing A/D conversion settings
DAQ0804ADINT    dscIntSettings;//  structure containing A/D conversion interrupt settings
DAQ0804ADINTSTATUS intstatus;// structure containing A/D conversion interrupt status
int sample[8];       // sample reading
int key = 0;
int Pause = 0;
/* Initializing A/D settings */
dscadsettings.Lowch = 0;
dscadsettings.Highch = 7;
dscadsettings.Polarity = 0; //bipolarity
dscadsettings.Range = 0;  //5v
dscadsettings.Sedi = 0;
dscadsettings.ScanEnable = 0;
dscadsettings.ScanInterval = 0;
dscadsettings.ADClock = 2;
if ( ( DAQ0804ADSetSettings ( bi, &dscadsettings ) ) != DE_NONE )
{
          dscGetLastError(&errorParams);
          printf ("DAQ0804ADSetSettings error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
          errorParams.errstring );
          return 0;
}
if ( (DAQ0804ADSetTiming(bi,&dscadsettings) ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ( "DAQ0804ADSetTiming error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
      return 0;
}
/* Initializing Interrupt settings */
dscIntSettings.NumConversions = 1000;
dscIntSettings.Cycle = 1;
dscIntSettings.FIFOEnable = 1;
dscIntSettings.FIFOThreshold = 1000;
```

```
dscIntSettings.ADBuffer = (SWORD*) malloc (sizeof(SWORD)* dscIntSettings.NumConversions );
if( DAQ0804ADInt(bi,&dscIntSettings ) !=DE_NONE)
{
        dscGetLastError(&errorParams);
        printf ( "DAQ0804ADInt error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
if( DAQ0804CounterSetRate(bi,dscadsettings.ADClock-2,rate,0,0) !=DE_NONE)
{
        dscGetLastError(&errorParams);
        printf ( "DAQ0804ADInt error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
if ( (DAQ0804ADSetTiming(bi,&dscadsettings) ) != DE_NONE )
{
        dscGetLastError(&errorParams);
        printf ( "DAQ0804ADSetTiming error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
        errorParams.errstring );
        return 0;
}
dscSleep(1000);
printf("\nPress Space bar key to Pause/Resume A/D Int or Press any other key to stop A/D Interrupt \n");
while (1)
{
    key = kbhit();
   if(key==0)
  {
      if(DAQ0804ADIntStatus(bi,&intstatus) !=DE_NONE)
      {
            dscGetLastError(&errorParams);
            printf ( "DAQ0804ADIntStatus error: %s %s\n", dscGetErrorString(errorParams.ErrCode),
            errorParams.errstring );
            return 0;
     }
     printf("No of A/D conversions completed %d\n",intstatus.NumConversions);
    dscSleep(1000);
    if( (dscIntSettings.Cycle == 0) && (intstatus.NumConversions >=dscIntSettings.NumConversions) )
                                        break;
 }
 else
 {
       key = getchar();
       if(key ==32 )
       {
            if(Pause )
           {
                    if(DAQ0804ADIntResume(bi) !=DE_NONE)
                    {
                        dscGetLastError(&errorParams);
                        printf ( "DAQ0804ADIntResume error: %s %s\n",
                            dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                         return 0;
                    }
                    Pause = 0;
            }
```

```
            else
                {

                        if(DAQ0804ADIntPause(bi) !=DE_NONE)
                                {
                                    dscGetLastError(&errorParams);
                                    printf ( "DAQ0804ADIntPause error: %s %s\n",
                                    dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                                        return 0;
                                 }
                                    Pause = 1 ;
                            }
                    }
                    else
                    {
                        break;
                    }
}
}
If(DAQ0804ADIntCancel(bi)!=DE_NONE)
{
                        dscGetLastError(&errorParams);
                        printf ( "DAQ0804ADIntPause error: %s %s\n",
                                    dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                        return 0;
}
```

7.15 Performing LED Operations

Description:

The application toggles the on board LED whenever the space bar key pressed. The DAQ0804LED() function is used to toggle on board LED.

Example of Usage for LED toggle:

```
Int key=0
Int LEDOn = 1;
printf("\nPress space bar key to toggle LED or Press 'q' to quit  \n");
while(1)
{
                key =getch();

                 if (key == 32)
                {
                        if(LEDOn)
                        {
                                DAQ0804LED(bi,0);
                                LEDOn = 0;
                        }
                        else
                        {
                                DAQ0804LED(bi,1);
                                LEDOn = 1;
                        }
                }
                else if(key == 'q')
                {
                        break;
                }
}
```

8. APPENDIX: REFERENCE INFORMATION

Counter/timer commands

The counter/timers are programmed with a series of commands. Each command is a 4 bit value. A command may have an associated option. A series of commands is required to configure a counter/timer for operation. See the counter/timer usage instructions in the manual for more information.

0 Clear the selected counter. If count direction is up the counter register is cleared to 0. If count direction is down the counter register is set to the reload value. All other counter settings are preserved. If the counter is running it continues running.
1 Load the selected counter with data in registers 0-3. This is used for down counting operations only.
2 Select count direction, up or down
3 Enable / disable external gate. This command is not implemented on the DS-MPE-DAQ0804.
4 Enable / disable counting
5 Latch selected counter. A counter must be latched before its contents can be read. Latching can occur while the counter is counting. The latched data can be read with the DAQ0804CounterRead() function.
6 Select counter clock source according to the table below:

Option	Clock source
0	External input pin, active low
1	Reserved
2	Internal clock 50MHz
3	Internal clock 1MHz

If an external DIO pin is selected as the counter input, the DIO pin's direction is automatically set for input mode. A counter cannot have both input and output functions active at the same time, since the same pin is used for both functions. If both are selected, the input function will prevail.

A counter must be enabled for the external input function to override the normal DIO operation. When one or more counters are reset with command 15, any I/O pins tied to the counter or counters are released to normal DIO operation.

7 Enable / Disable Auto-Reload When auto-reload is enabled, then when the counter is counting down and it reaches 1, on the next clock pulse it will reload its initial value and keep counting. Otherwise on the next clock pulse it will count down to 0 and stop.

8 Enable counter output and select the output pulse polarity. The initial logic level of the output pin will be the inactive state (the opposite state of the selected polarity). The output pulse always comes at the end of each clock period. The counter outputs are enabled on DIO pins according to the following table. Enabling a counter output automatically sets the corresponding DIO pin's direction to output. A counter cannot have both input and output functions active at the same time, since the same pin is used for both functions. If both are selected, the input function will prevail, and the requested output function will be ignored.

Connector J2 Pin	DIO Bit	Counter/Timer
7	6	0
8	7	1
9	8	2
10	9	3
11	10	4
12	11	5
13	12	6
14	13	7

9 Select counter output pulse width. Only has effect when counter output is enabled with command 1000 and clock source selected with command 6 is internal 50MHz or 1MHz. The counter output pulse width is defined according to the table below.

If the selected pulse width is equal to or greater than the clock period, the output will stay at its active state indefinitely.

If external clock is selected, the output pulse is always 1 clock wide, meaning that it will transition to active on the terminal count clock pulse and then transition to inactive on the next clock pulse.

Option	Output pulse width
0	1 clock (default if command is not executed)
1	10 clocks
2	100 clocks
3	1000 clocks

15 Reset one or all counters. When any counter is reset, all its registers are cleared to zero, and any DIO lines assigned to that counter for input or output are released to normal DIO operation. A command of 0xFF will reset all counters.

Each counter's output operates as follows: when disabled or during normal counting operation, the output is 0. When count direction is up, the output is always 0. When count direction is down, then when the counter reaches the selected pulse width, the output will go high. When the counter reaches 1, it will reload to its initial value on the next clock pulse. Thus a counter value of n will result in a divide by n output pulse rate. If a counter latch command is requested during this process, the command will be delayed until the reload is completed.

PWM commands

The PWMs are programmed with a series of commands. A command may have an associated parameter, referred to as PWMCD in the descriptions below. A series of commands is required to configure a PWM for operation. See the PWM usage instructions in the manual for more information.

0 Stop all PWMs / selected PWM
0 = stop all PWMs (opposite polarity for "all" compared to other PWM commands)
1 = stop single PWM

When a PWM is stopped, its output returns to its inactive state, and the registers are reloaded with their initial values. If the PWM is subsequently restarted, it will start at the beginning of its waveform, i.e. the start of the active output pulse.

1 Load counter C0 (period counter) or C1 (duty cycle counter)

0 = load C0 / period counter
1 = load C1 = duty cycle counter

2 Set output polarity. The pulse occurs at the start of the period.

0 = pulse high
1 = pulse low

3 Enable/disable pulse output

0 = disable pulse output; output = opposite of polarity setting from command 2
1 = enable pulse output

4 Reset all PWMs / selected PWM

0 = reset PWM selected with PWM2-0
1 = reset all PWMs

When a PWM is reset, it stops running, and any DIO line assigned to that PWM for output is released to normal DIO operation. The direction of the DIO line will revert to its value prior to the PWM operation.

5 Enable/disable PWM outputs on DIO port F

0 = disable output
1 = enable output on DIO pin; this forces the DIO pin to output mode

6 Select clock source for a PWM

0 = 50MHz
1 = 1MHz

7 Start all PWMs / selected PWM

0 = start PWM selected with PWM2-0
1 = start all PWMs

If a PWM output is not enabled, its output is forced to the inactive state, which is defined as the opposite of the value selected with command 2. The PWM may continue to run even though its output is disabled.

PWM outputs may be made available on I/O pins according to the table below using command 5. When a PWM output is enabled, the corresponding DIO pin is forced to output mode. To make the pulse appear on the output pin, command 3 must additionally be executed, otherwise the output will be held in inactive mode (the opposite of the selected polarity for the PWM output).

Connector J2 Pin	DIO Bit	PWM
15	14	0
16	15	1
17	16	2
18	17	3