16AO4MF

16-bit, 4 channel, 400K S/S/Ch Analog Output

PMC-16AO4MF
PMC-16AO4MFS

API Library
Reference Manual

Manual Revision: July 9, 2018
Driver Release Version 3.3.79.x.0

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: www.generalstandards.com
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com

Preface

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation
8302A Whitesburg Dr.
Huntsville, Alabama 35802
Phone: (256) 880-8787
FAX: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AO4MF API Library and, to a lesser extent, the underlying device driver. The API Library software provides the interface between "Application Software" and the device driver. The driver software provides the interface between the API Library and the actual 16AO4MF hardware. The API Library and device driver interfaces are primarily IOCTL based.

1.2. Acronyms

The following is a list of commonly occurring acronyms used throughout this document.

Acronyms	Description
API	Application Programming Interface
DMA	Direct Memory Access
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card

1.3. Definitions

The following is a list of commonly occurring terms used throughout this document.

Term	Definition
…	This is used as a shortcut for the 16AO4MF install directory path or one of its subdirectories.
16AO4MF	This is used as a general reference to any board supported by this driver and API Library.
API Library	This is the library that provides application level access to the 16AO4MF device driver.
Application	This is the user mode process, which runs in user space with user mode privileges.
Driver	This refers to the device driver.
INtime	This refers to the “INtime for Windows” real-time extension for Microsoft Windows. Refer to the *16AO4MF INtime for Windows Driver User Manual*.
Library	This is usually a general reference to the API Library.
Linux	This refers to the Linux operating system. Refer to the *16AO4MF Linux Driver User Manual*.

1.4. Software Overview

An architectural representation of the software is given in Figure 1.

1.4.1. API Library

The API Library is the application level software by which applications are able to communicate with 16AO4MF hardware. The library forms a layer between the application and the device driver. With the library, applications are able to open and close access to a device and, while open, perform I/O control operations, read data from the board and write data to the board. The interface implemented by the API Library is based on the C programming language (section 3, page 10).

1.4.2. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AO4MF hardware. Depending on the OS, the driver may be a user space application, a kernel mode process, or something in between.

The software interface to the device driver is analogous to that of the API Library, and is based on the board's functionality.

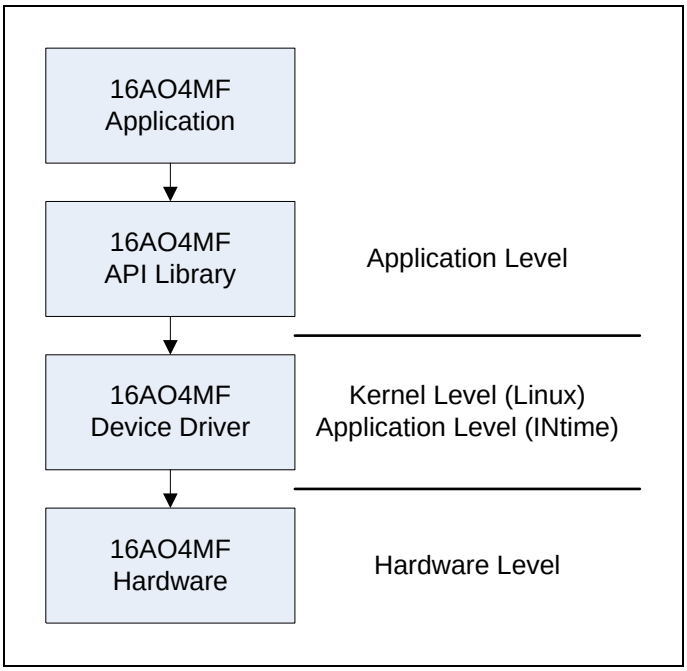

Figure 1 Architecture Representation

1.5. Hardware Overview

The 16AO4MF is a high-performance, 16-bit analog output board that incorporates four independent output channels. The host side connection is PCI based and the form factor is according to the model ordered. The board is capable of outputting data at up to 400K samples per second over each channel. Internal clocking permits sampling rates from 400K samples per second down to 244 samples per second. Onboard storage permits data buffering of up to 32K samples, for each channel individually, between the PCI bus and the cable interface. This allows the 16AO4MF to sustain continuous throughput to the cable interface independent of the PCI bus interface. The 16AO4MF also permits multiple boards to be synchronized so that all boards output data in unison.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AO4MF. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AO4MF Driver User Manual* from General Standards Corporation.
- The applicable *16AO4MF User Manual* from General Standards Corporation.
- The PCI9080 PCI Bus Master Interface Chip data handbook from PLX Technology, Inc.

  PLX Technology Inc.
  870 Maude Avenue
  Sunnyvale, California 94085 USA
  Phone: 1-800-759-3735
  WEB: http://www.plxtech.com

2. Important Support Files

2.1. Main Header File

The 16AO4MF driver package provides a main header file that does an include of all application level 16AO4MF headers. Throughout this document references are given for a variety of 16AO4MF specific header files. Plus, these collectively include numerous others not specifically named in this document, but which are also included in the 16AO4MF driver package. For ease of use it is suggested that applications include only the main header file shown below rather than individually including those headers identified separately throughout this document. Including the main header file pulls in all other pertinent 16AO4MF specific header files. Therefore, sources may include only this one 16AO4MF header and makefiles may reference only this one 16AO4MF include directory. All 16AO4MF API Library headers and all affiliated headers are included via this one header file and are all located in this one directory.

Description	File	Location	OS
Header File	`16ao4mf_main.h`	`…/include/`	All

2.2. Main Library File

The 16AO4MF driver package provides a main statically linkable library that is a substitute for separately linking all static libraries built individually as a part of the driver package. Throughout this document references are given for a variety of 16AO4MF specific static libraries. For ease of use it is suggested that applications link only the main static library file shown below rather than individually linking the entire set of 16AO4MF static libraries. Linking the main library file pulls in all other pertinent 16AO4MF specific libraries. Therefore, makefiles may link only this one 16AO4MF library and may reference only this one 16AO4MF library directory. The 16AO4MF API Library file and all affiliated libraries are incorporate via this one library file and all are located in this one directory.

Description	File	Location	OS
Library File	`16ao4mf_main.a`	`…/lib/`	Linux
	`16ao4mf_main.lib`	`…\lib\… *`	INtime

* Debug and release versions of the library are included under corresponding subdirectories.

2.3. Libraries

2.3.1. API Library

The Library is provided as a static library directly linkable with applications. The pertinent files are identified in the following table. Some source files are specific only to the 16AO4MF, some are specific only to the OS and some are 16AO4MF and OS independent.

Description	Files	Location	OS
Source Files	`*.c`	`…/api/`	All
Header File	`16ao4mf_api.h`	`…/include/`	All
Library File	`16ao4mf_api.a`	`…/lib/`	Linux
	`16ao4mf_api.lib` `16ao4mf_api.rtl` †	`…\lib\… *`	INtime

* Debug and release versions of the libraries are included under corresponding subdirectories.
† The run time executable is provided in the form of an INtime DLL.

2.3.2. Document Source Library

The source code examples given in this document with the API Library function calls are provided as C files included with the driver package. This is done to verify that the code compiles correctly. Additionally, the sources

are compiled and linked into a static library to simplify use of the examples. The pertinent files are identified in the following table. All source files are specific to the 16AO4MF, but independent of the OS.

Description	Files	Location	OS
Source Files	*.c	…/docsrc/	All
Header File	16ao4mf_dsl.h	…/include/	All
Library File	16ao4mf_dsl.a	…/lib/	Linux
	16ao4mf_dsl.lib	…\lib\… *	INtime

* Debug and release versions of the library are included under corresponding subdirectories.

2.3.3. Utilities Library

The 16AO4MF API Library includes a body of utility source code designed to aid in the understanding and use of all API calls and all IOCTL services. The essence of these utilities is to implement visual wrappers around the corresponding services. Utility sources are also included for device independent and common, general purpose services. The utility services are used extensively by the sample applications. For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `ao4mf_open()` there is the utility file `open.c` containing the utility function `ao4mf_open_util()`. The naming pattern is as follows: API function `ao4mf_xxxx()`, utility file name `xxxx.c`, utility function `ao4mf_xxxx_util()`. Additionally, for each IOCTL code there is a corresponding utility source file with a corresponding utility service. As an example, for IOCTL code `AO4MF_IOCTL_TX_IO_MODE` there is the utility file `util_tx_io_mode.c` containing the utility function `ao4mf_tx_io_mode()`. The naming pattern is as follows: IOCTL code `AO4MF_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `ao4mf_xxxx()`. The utility sources are compiled and linked into a static library to simplify their use. The pertinent files are identified in the following table. Some source files are specific only to the 16AO4MF, some are specific only to the OS and some are 16AO4MF and OS independent.

Description	Files	Location	OS
Source Files	*.c	…/utils/	All
Header File	16ao4mf_utils.h	…/include/	All
Library File	16ao4mf_utils.a	…/lib/	Linux
	16ao4mf_utils.lib	…\lib\… *	INtime

* Debug and release versions of the library are included under corresponding subdirectories.

2.4. Sample Applications

The driver package includes several example applications. These may be useful both for testing and for programming demonstration purposes. The examples make extensive use of the utility libraries also included in the driver package. The files are located as given in the table below. Most source files are specific to the 16AO4MF, but independent of the OS.

Description	Files	Location	OS
Source Files	*.c	…/samples/	All *

* Some sample applications are OS specific and not included with all driver packages.

3. API Library Interface

The 16AO4MF API Library is the software interface between user applications and the 16AO4MF device driver. All driver functionality is accessible through this interface.

NOTE: Contact General Standards Corporation if additional library functionality is required.

3.1. Library Use

The library is used both at application compile time and at application link time. At compile time include the below listed header file in each source file using a component of the library interface. At link time include the below listed library files with the objects being linked with the application.

Description	Files	Location	OS
Header File	`16ao4mf_api.h`	`…/include/`	All
Library Files	`16ao4mf_api.a`	`…/lib/`	Linux
	`16ao4mf_api.lib` `16ao4mf_api.rtl` †	`…\lib\… *`	INtime

* Debug and release versions of the libraries are included under corresponding subdirectories.
† The run time executable is provided in the form of an INtime DLL.

3.2. Macros

The Library interface includes the following macros, which are defined in `16ao4mf.h`.

3.2.1. IOCTL

The IOCTL macros are documented in section 3.5 beginning on page 16.

3.2.2. Registers

The following gives the complete set of 16AO4MF registers.

3.2.2.1. GSC Registers

The following tables give the complete set of GSC specific 16AO4MF registers. For detailed definitions of these registers refer to the relevant *16AO4MF User Manual*. Please note that the set of registers supported by any given board may vary according to model and firmware version. For the set of supported registers and detailed definitions of these registers please refer to the appropriate *16AO4MF User Manual*.

Macro	Description
`AO4MF_GSC_ACCR`	Adjustable Clock Control Register
`AO4MF_GSC_ADR`	Autocal Diagnostics Register
`AO4MF_GSC_BCR`	Board Control Register
`AO4MF_GSC_BOR`	Buffer Operations Register
`AO4MF_GSC_DIOPCR`	Digital I/O Port Control Register
`AO4MF_GSC_FRR`	Firmware Revision Register
`AO4MF_GSC_ICSCR`	Internal Clock Source Control Register
`AO4MF_GSC_ODBR`	Output Data Buffer Register
`AO4MF_GSC_SRCR`	Sample Rate Control Register

3.2.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the header file `gsc_pci9080.h`, which is automatically included via `16ao4mf.h`.

3.2.2.3. PLX PCI9080 Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the header file `gsc_pci9080.h`, which is automatically included via `16ao4mf.h`.

3.3. Data Types

The data types used by the Library are identified and described with the IOCTL macros (section 3.5, page 16).

3.4. Functions

The Library interface includes the following functions. The return values reflect the completion status of the requested operation. A value of zero indicates success. A negative value indicates that the request could not be completed successfully. The specific values returned are described in the table below. I/O services return positive values to indicate the number of bytes successfully transferred.

Return Value	Description	OS
`-1 to -999`	This is the value `(-errno)` (see `errno.h`).	All
`<= 1000`	This is the value `(-(int)(GetLastRtError()+1000))`.	INtime

3.4.1. ao4mf_close()

This function is the entry point to close a connection to an open 16AO4MF board.

Prototype

```
int ao4mf_close(int fd);
```

Argument	Description
`fd`	This is the file descriptor of the device to be closed.

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value descriptions above.

Example

```
#include <stdio.h>

#include "16ao4mf_dsl.h"

int ao4mf_close_dsl(int fd)
{
    int err;
    int ret;

    ret = ao4mf_close(fd);

    if (ret)
```

```
        printf("ERROR: ao4mf_close() returned %d\n", ret);

    err = ret ? 1 : 0;
    return(err);
}
```

3.4.2. ao4mf_init()

This function is the entry point to initializing the 16AO4MF API Library and must be the first call into the Library. This function may be called more than once, but only the first successful call actually initializes the library. Subsequent calls perform no operation at all. All other API calls return a failure status when the API Library is not initialized.

Prototype

```
int ao4mf_init(void);
```

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value descriptions above.

Example

```
#include <stdio.h>

#include "16ao4mf_dsl.h"

int ao4mf_init_dsl(void)
{
    int err;
    int ret;

    ret = ao4mf_init();

    if (ret)
        printf("ERROR: ao4mf_init() returned %d\n", ret);

    err = ret ? 1 : 0;
    return(err);
}
```

3.4.3. ao4mf_ioctl()

This function is the entry point to performing setup and control operations on a 16AO4MF board. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver (section 3.5, page 16).

Prototype

```
int ao4mf_ioctl(int fd, int request, void* arg);
```

Argument	Description
fd	This is the file descriptor of the device to access.
request	This specifies the desired operation to be performed.
arg	This is a request specific argument. Refer to the IOCTL services for additional information (section 3.5, page 16).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value descriptions above.

Example

```
#include <stdio.h>

#include "16ao4mf_dsl.h"

int ao4mf_ioctl_dsl(int fd, int request, void* arg)
{
    int err;
    int ret;

    ret = ao4mf_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: ao4mf_ioctl() returned %d\n", ret);

    err = ret ? 1 : 0;
    return(err);
}
```

3.4.4. ao4mf_open()

This function is the entry point to open a connection to a 16AO4MF board.

Prototype

```
int ao4mf_open(int index, int share, int* fd);
```

Argument	Description
index	This is the zero based index of the 16AO4MF to access. *
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows.

Value	Description
>= 0	This is the handle to use to access the device in subsequent calls.
-1	There was an error. The device is not accessible.

* The index value -1 can also be given to acquire driver information (section 3.4.4.2, page 14).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value descriptions above.

Example

```
#include <stdio.h>

#include "16ao4mf_dsl.h"

int ao4mf_open_dsl(int index, int share, int* fd)
{
    int err;
    int ret;

    ret = ao4mf_open(index, share, fd);

    if (ret)
        printf("ERROR: ao4mf_open() returned %d\n", ret);

    err = ret ? 1 : 0;
    return(err);
}
```

3.4.4.1. Access Modes

Shared Access Mode:

Shared Access Mode allows multiple applications to access the same device simultaneously. In this mode, the first successful open request returns with the device in an initialized state. Subsequent successful Shared Access Mode open requests do not affect the state of the device. Once opened in Shared Access Mode, the device access remains in this mode until all Shared Access Mode accesses release the device with a close request.

Exclusive Access Mode:

Exclusive Access Mode allows a single application to acquire exclusive access to a device. In this mode, a successful open request returns with the device in an initialized state. While open in this mode all subsequent open requests will fail regardless of the access mode requested. Once opened in Exclusive Access Mode, the device access remains in this mode until released by the application with a close request.

3.4.4.2. Device Index -1

If an open request is made with a device index of `-1` the application gains access to the driver rather than to hardware. In this case, the application gains read access to the driver rather than to any underlying hardware. Read requests return the information shown below. Each line includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what is returned, followed by descriptions of each entry. Neither write nor IOCTL services are supported.

```
version: 3.3.79.25
32-bit support: yes (native)
boards: 1
models: 16AO4MF
```

Entry	Description
`version`	This gives the driver version number in the form `x.x.x.x`. The last number in the version number is specific to the OS.

32-bit support	This reports the driver's support for 32-bit applications. This will be either "yes" or "no" for 64-bit driver builds and "yes (native)" for 32-bit builds.
boards	This identifies the total number of boards the driver detected.
models	This gives a comma separated list of the basic model number for each board the driver detected.

The source for the text provided is as follows.

OS	Source
Linux	The file "/proc/16ao4mf".
INtime	The Driver Mailbox, which is named "16ao4mf".

3.4.5. ao4mf_read()

The 16AO4MF has no receive data streaming capabilities. This service is strictly for reading from access gained by an open on device index -1 (section 3.4.4.2, page 14). The function reads up to bytes bytes from the driver. The return value is the number of bytes actually read.

Prototype

```
int ao4mf_read(int fd, void* buf, size_t bytes);
```

Argument	Description
fd	This is the file descriptor of the device to access.
buf	The data read will be put here.
bytes	This is the desired number of bytes to read.

Return Value	Description
0 to bytes	The operation succeeded. A value less than bytes indicates that the request timed out.
< 0	An error occurred. See error value descriptions above.

Example

```
#include <stdio.h>

#include "16ao4mf_dsl.h"

int ao4mf_read_dsl(int fd, void* dst, size_t bytes)
{
    int ret;

    ret = ao4mf_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: ao4mf_read() returned %d\n", ret);

    return(ret);
}
```

3.4.6. ao4mf_write()

This function is the entry point to writing data to an open 16AO4MF. This function should only be called after a successful open of the respective device. The function writes up to bytes bytes to the board. The return value is the number of bytes actually written.

NOTE: The driver's write service may dynamically manipulate the output buffer threshold level. When this is done the original value will be restored before the write service returns. The output buffer threshold level will not be manipulated if the output buffer threshold status has been selected as an interrupt source. In these cases write performance may be reduced.

Prototype

```
int write(int fd, const void* buf, size_t bytes);
```

Argument	Description
fd	This is the file descriptor of the device to access.
buf	The data to write is taken from this pointer.
bytes	This is the desired number of bytes to write. This must be a multiple of four (4).

Return Value	Description
0 to bytes	The operation succeeded. A value less than bytes indicates that the request timed out.
< 0	An error occurred. See error value descriptions above.

Example

```
#include <stdio.h>

#include "16ao4mf_dsl.h"

int ao4mf_write_dsl(int fd, const void* src, size_t bytes)
{
    int ret;

    ret = ao4mf_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: ao4mf_write() returned %d\n", ret);

    return(ret);
}
```

3.5. IOCTL Services

The 16AO4MF API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `ao4mf_ioctl()` function arguments.

3.5.1. AO4MF_IOCTL_AO_BUF_CLEAR

This service immediately clears the current content from the channel's output buffer.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_BUF_CLEAR
arg	Not used.

3.5.2. AO4MF_IOCTL_AO_BUF_LOAD_READY

This service reports the buffer's readiness to receive additional data when in circular buffer mode.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_BUF_LOAD_READY
arg	s32*

Valid values returned by the service are as follows.

Value	Description
AO4MF_AO_BUF_LOAD_READY_NO	The buffer is not ready to receive additional data.
AO4MF_AO_BUF_LOAD_READY_YES	The buffer is ready to receive additional data.

3.5.3. AO4MF_IOCTL_AO_BUF_LOAD_REQ

This service requests that the output buffer become ready to receive additional data when in circular buffer mode. The service returns immediately without waiting for the buffer to become ready.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_BUF_LOAD_REQ
arg	Not used.

3.5.4. AO4MF_IOCTL_AO_BUF_MODE

This service configures the channel's handling of data once it leaves the output buffer.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_BUF_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_AO_BUF_MODE_CIRC	Buffer data is recycled when it exits the buffer.
AO4MF_AO_BUF_MODE_OPEN	The buffer data is not recycled when it exits the buffer.

3.5.5. AO4MF_IOCTL_AO_BUF_SIZE

This service configures the active size of the output buffer.

> **NOTE:** It is recommended that the buffer be cleared after changing the buffer size (see AO4MF_IOCTL_AO_BUF_CLEAR, section 3.5.1, page 16).

Usage

Argument	Description
request	AO4MF_IOCTL_AO_BUF_SIZE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_AO_BUF_SIZE_4	Set the buffer's active size to 4 samples.
AO4MF_BUFFER_SIZE_8	Set the buffer's active size to 8 samples.
AO4MF_BUFFER_SIZE_16	Set the buffer's active size to 16 samples.
AO4MF_BUFFER_SIZE_32	Set the buffer's active size to 32 samples.
AO4MF_BUFFER_SIZE_64	Set the buffer's active size to 64 samples.
AO4MF_BUFFER_SIZE_128	Set the buffer's active size to 128 samples.
AO4MF_BUFFER_SIZE_256	Set the buffer's active size to 256 samples.
AO4MF_BUFFER_SIZE_512	Set the buffer's active size to 512 samples.
AO4MF_BUFFER_SIZE_1K	Set the buffer's active size to 1K samples (1,024).
AO4MF_BUFFER_SIZE_2K	Set the buffer's active size to 2K samples (2,048).
AO4MF_BUFFER_SIZE_4K	Set the buffer's active size to 4K samples (4,096).
AO4MF_BUFFER_SIZE_8K	Set the buffer's active size to 8K samples (8,192).
AO4MF_BUFFER_SIZE_16K	Set the buffer's active size to 16K samples (16,384).
AO4MF_BUFFER_SIZE_32K	Set the buffer's active size to 32K samples (32,768).

3.5.6. AO4MF_IOCTL_AO_BUF_STATUS

This service reports the relative fill level of the active buffer. The buffer's active size is set with the AO4MF_IOCTL_BUFFER_SIZE service (section 3.5.5, page 17).

Usage

Argument	Description
request	AO4MF_IOCTL_AO_BUF_STATUS
arg	s32*

The service returns one of the following values.

Value	Description
AO4MF_AO_BUF_STATUS_EMPTY	The buffer is empty.
AO4MF_AO_BUF_STATUS_1Q_FULL	The buffer is less than ¼ full.
AO4MF_AO_BUF_STATUS_MEDIUM	The buffer is from ¼ to ¾ full.
AO4MF_AO_BUF_STATUS_3Q_FULL	The buffer is more than ¾ full.
AO4MF_AO_BUF_STATUS_FULL	The buffer is full.

3.5.7. AO4MF_IOCTL_AO_CLK_ENABLE

This service enables and disables clocking of the channel's output data.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_CLK_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_AO_CLK_ENABLE_NO	This disables the output sample clock.
AO4MF_AO_CLK_ENABLE_YES	This enables the output sample clock.

3.5.8. AO4MF_IOCTL_AO_CLK_SRC

This service selects the source for the output sample clock.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_AO_CLK_SRC_EXT_SW	This selects the external/software option as the clock source.
AO4MF_AO_CLK_SRC_INT	This selects the internal rate generator as the source.

3.5.9. AO4MF_IOCTL_AO_EXT_CLKING_READY

This service reports the channel's readiness to accept external clocking.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_EXT_CLKING_READY
arg	s32*

The service returns one of the following values.

Value	Description
AO4MF_AO_EXT_CLKING_READY_NO	The device is not ready for the clock.
AO4MF_AO_EXT_CLKING_READY_YES	The device is ready for the clock.

3.5.10. AO4MF_IOCTL_AO_SW_CLK

This service initiates an output clock cycle. The service returns immediately and does not wait for the operation to complete.

Usage

Argument	Description
request	AO4MF_IOCTL_AO_SW_CLK
arg	Not used.

3.5.11. AO4MF_IOCTL_AUTO_CAL

This service initiates an auto-calibration cycle for the current channel. Most configuration settings should be made before running an auto-calibration cycle. The driver waits for the operation to complete before returning.

> **NOTE:** Do not access the board while an auto-calibration cycle is in progress. Doing so may produce indeterminate results, and may lockup the board.

> **NOTE:** If the auto-calibration service returns an error status, an error message will be posted to the system log briefly describing the error condition.

Usage

Argument	Description
request	AO4MF_IOCTL_AUTO_CAL
arg	Not used.

3.5.12. AO4MF_IOCTL_AUTO_CAL_STS

This service retrieves the auto-calibration completion status for the current channel.

Usage

Argument	Description
request	AO4MF_IOCTL_AUTO_CAL_STS
arg	s32*

The value returned will be one of the following.

Value	Description
AO4MF_AUTO_CAL_STS_ACTIVE	Auto-calibration is in progress.
AO4MF_AUTO_CAL_STS_FAIL	Auto-calibration failed.
AO4MF_AUTO_CAL_STS_PASS	Auto-calibration passed.

3.5.13. AO4MF_IOCTL_BURST_MODE

This service enables or disables output bursting.

Usage

Argument	Description
request	AO4MF_IOCTL_BURST_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO4MF_BURST_MODE_DISABLE	This refers to output bursting being disabled.
AO4MF_BURST_MODE_ENABLE	This refers to output bursting being enabled.

3.5.14. AO4MF_IOCTL_BURST_READY

This service reports the channel's readiness for burst initiation.

Usage

Argument	Description
request	AO4MF_IOCTL_BURST_READY
arg	s32*

The service returns one of the following values.

Value	Description
AO4MF_BURST_READY_NO	The channel is not ready for burst initiation.

AO4MF_BURST_READY_YES	The channel is ready for burst initiation.

3.5.15. AO4MF_IOCTL_BURST_TRIGGER

This service initiates an output burst cycle. The service returns immediately without waiting for the burst to complete.

Usage

Argument	Description
request	AO4MF_IOCTL_BURST_TRIGGER
arg	Not used.

3.5.16. AO4MF_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	AO4MF_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_DATA_FORMAT_2S_COMP	Select the Twos Compliment data format.
AO4MF_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.

3.5.17. AO4MF_IOCTL_DIO_DIR

This service sets the direction of the eight digital I/O lines.

NOTE: The digital I/O lines are available on the board's first channel only.

Usage

Argument	Description
request	AO4MF_IOCTL_DIO_DIR
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting. This value is returned when the digital I/O lines are unsupported.
AO4MF_DIO_DIR_IN	This refers to the input direction.
AO4MF_DIO_DIR_OUT	This refers to the output direction.

3.5.18. AO4MF_IOCTL_DIO_RX

This service reads the value driven on the eight digital I/O lines.

NOTE: The digital I/O lines are available on the board's first channel only.

Usage

Argument	Description
request	AO4MF_IOCTL_DIO_RX
arg	s32*

Valid values returned are in the range from zero to 0xFF, and -1. The value -1 is returned when the digital I/O lines are unsupported.

3.5.19. AO4MF_IOCTL_DIO_TX

This service writes the value to be driven by the channel onto the eight digital I/O lines when configured as outputs.

NOTE: The digital I/O lines are available on the board's first channel only.

Usage

Argument	Description
request	AO4MF_IOCTL_DIO_TX
arg	s32*

Valid argument values are from zero to 0xFF. The value -1 is returned when the digital I/O lines are unsupported.

3.5.20. AO4MF_IOCTL_FREF_CLK_SRC

This service selects between the master reference clock and the adjustable reference clock.

Usage

Argument	Description
request	AO4MF_IOCTL_FREF_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_FREF_CLK_SRC_ADJ	This selects the adjustable reference clock.
AO4MF_FREF_CLK_SRC_MASTER	This selects the master reference clock.

3.5.21. AO4MF_IOCTL_INITIALIZE

This service returns all driver interface settings for the channel to the state they were in when the channel was first opened. This includes both hardware based settings and software based settings.

NOTE: If the initialization service returns an error status, an error message will be posted to the system log briefly describing the error condition.

Usage

Argument	Description
request	AO4MF_IOCTL_INITIALIZE

arg	Not used.

3.5.22. AO4MF_IOCTL_INT_SAMP_RATE

This service configures the internal sample rate as being variable or as one of several fixed rates.

Usage

Argument	Description
request	AO4MF_IOCTL_INT_SAMP_RATE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_INT_SAMP_RATE_10000_SPS	This refers to the fixed sample rate of 10,000 S/S.
AO4MF_INT_SAMP_RATE_22500_SPS	This refers to the fixed sample rate of 22,500 S/S.
AO4MF_INT_SAMP_RATE_27304_SPS	This refers to the fixed sample rate of 27,304 S/S.
AO4MF_INT_SAMP_RATE_60000_SPS	This refers to the fixed sample rate of 60,000 S/S.
AO4MF_INT_SAMP_RATE_224000_SPS	This refers to the fixed sample rate of 224,000 S/S.
AO4MF_INT_SAMP_RATE_VAR	This refers to a variable sample rate.

3.5.23. AO4MF_IOCTL_IRQ_SEL

This service configures the interrupt source selection for the firmware interrupt.

Usage

Argument	Description
request	AO4MF_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_IRQ_SEL_AO_BUF_1Q_FULL	This refers to the buffer becoming less than ¼ full.
AO4MF_IRQ_SEL_AO_BUF_3Q_FULL	This refers to the buffer becoming more than ¾ full.
AO4MF_IRQ_SEL_AO_BUF_EMPTY	This refers to the buffer becoming empty.
AO4MF_IRQ_SEL_AUTO_CAL_DONE	This refers to the completion of auto-calibration.
AO4MF_IRQ_SEL_BURST_TRIG_READY	This refers to the channel becoming ready for a burst trigger.
AO4MF_IRQ_SEL_LOAD_READY	This refers to a circular buffer becoming ready to receive data.
AO4MF_IRQ_SEL_LOAD_READY_END	This refers to a circular buffer becoming not ready to receive data.
AO4MF_IRQ_SEL_NONE_INIT_DONE	This refers to the completion of initialization.

3.5.24. AO4MF_IOCTL_NCLK

This service sets the adjustable clock's NCLK adjustment value.

Usage

Argument	Description
request	AO4MF_IOCTL_NCLK
arg	s32*

Valid argument values are in the range from zero to 511, and −1. The value −1 is used to retrieve the current setting.

3.5.25. AO4MF_IOCTL_NRATE

This service sets the rate divider's NRATE adjustment value.

Usage

Argument	Description
request	AO4MF_IOCTL_NRATE
arg	s32*

Valid argument values are in the range from 68 to 0xFFFF, and −1. The value −1 is used to retrieve the current setting.

3.5.26. AO4MF_IOCTL_QUERY

This service queries the driver for various pieces of information about the device and the driver.

Usage

Argument	Description
request	AO4MF_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
AO4MF_QUERY_AUT_CAL_MS	This returns the maximum duration of the Auto Calibration cycle in milliseconds.
AO4MF_QUERY_CHANNEL_QTY	This returns the number of output channels on the current board.
AO4MF_QUERY_COUNT	This returns the number of query options supported by the query service.
AO4MF_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. The value is a member of the gsc_dev_type_t enumeration, which is defined in gsc_common.h. The value returned should be GSC_DEV_TYPE_16AO4MF.
AO4MF_QUERY_FREF_ADJ	This gives the adjustable clock's reference frequency in hertz.
AO4MF_QUERY_FREF_MASTER	This gives the master clock's reference frequency in hertz.
AO4MF_QUERY_FSAMP_MAX	This gives the maximum sample rate in S/S.
AO4MF_QUERY_FSAMP_MIN	This gives the minimum sample rate in S/S.
AO4MF_QUERY_INDEX_CHANNEL	This gives the index of the current channel and will always be a value from zero to three.
AO4MF_QUERY_INDEX_DEVICE	This gives the zero based device node index, which corresponds to the suffix in the device node name /dev/16ao4mf.x.
AO4MF_QUERY_INIT_MS	This returns the duration of a channel initialization in milliseconds.

AO4MF_QUERY_NCLK_MAX	This returns the maximum supported NCLK value.
AO4MF_QUERY_NCLK_MIN	This returns the minimum supported NCLK value.
AO4MF_QUERY_NRATE_MAX	This returns the maximum supported NRATE value.
AO4MF_QUERY_NRATE_MIN	This returns the minimum supported NRATE value.
AO4MF_QUERY_REG_DIOPCR	This indicates if the DIOPCR register is supported on the current channel (0 = no, else yes).
AO4MF_QUERY_SYNC_CLK_OUT	This indicates if the cable's Clock Output and SYNC Output signals are supported on the current channel (0 = no, else yes).

Valid return values are as given in the below table.

Value	Description
AO4MF_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

3.5.27. AO4MF_IOCTL_RATE_GEN_SEL

This service configures the internal sample rate as being variable or as one of several fixed rates.

Usage

Argument	Description
request	AO4MF_IOCTL_RATE_GEN_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_RATE_GEN_SEL_CH_0	This refers to the channel zero rate generator.
AO4MF_RATE_GEN_SEL_CH_ALT	This refers to the alternate rate generator option. *

* The alternate selection for channel zero is channel one. For channel one it is channel one, for channel two it is channel two and for channel three it is channel three.

3.5.28. AO4MF_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AO4MF register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao4mf.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO4MF_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access.
value	This contains the value for the register bits to modify.
mask	This specifies the set of bits to modify. If a bit here is set, then the respective register bits is modified. If a bit here is zero, then the respective register bit is unmodified.

3.5.29. AO4MF_IOCTL_REG_READ

This service reads the value of a 16AO4MF register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16ao4mf.h` and `gsc_pci9080.h` for the complete list of accessible registers.

Usage

Argument	Description
request	AO4MF_IOCTL_REG_READ
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access.
value	This is the value read from the specified register.
mask	This is ignored for read request.

3.5.30. AO4MF_IOCTL_REG_WRITE

This service writes a value to a 16AO4MF register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao4mf.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO4MF_IOCTL_REG_WRITE
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access.

value	This is the value to write to the specified register.
mask	This is ignored for write request.

3.5.31. AO4MF_IOCTL_TX_IO_ABORT

This service aborts an ongoing write() request.

Usage

Argument	Description
request	AO4MF_IOCTL_TX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
AO4MF_IO_ABORT_NO	A write() request was not aborted as none were ongoing.
AO4MF_IO_ABORT_YES	An ongoing write() request was aborted.

3.5.32. AO4MF_IOCTL_TX_IO_MODE

This service sets the I/O mode used for data write requests.

Usage

Argument	Description
request	AO4MF_IOCTL_TX_IO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GSC_IO_MODE_DMA	Use DMA.
GSC_IO_MODE_PIO	Use PIO mode, which is repetitive register access. This is the default.

3.5.33. AO4MF_IOCTL_TX_IO_TIMEOUT

This service sets the timeout limit for data write requests. The value is expressed in seconds.

Usage

Argument	Description
request	AO4MF_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AO4MF_IO_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more space, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option AO4MF_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

3.5.34. AO4MF_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via AO4MF_IOCTL_WAIT_EVENT IOCTL calls (section 3.5.35, page 28), according to the provided criteria. When a blocked thread is waiting for any event specified in the structure, then the thread is resumed.

> **NOTE:** The driver itself makes use of the wait services for various internal operations. Driver initiated waits are unaffected by application cancel requests.

Usage

Argument	Description
request	AO4MF_IOCTL_WAIT_CANCEL
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  alt;
    u32  io;
    u32  timeout_ms;
    u32  count;
} gsc_wait_t;
```

Fields	Description
flags	This is unused by wait cancel operations.
main	This specifies the set of GSC_WAIT_MAIN_* events whose wait requests are to be cancelled. Refer to section 3.5.35.2 on page 29.
gsc	This specifies the set of AO4MF_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 3.5.35.3 on page 30.
alt	This is unused by the 16AO4MF driver and should be zero.
io	This specifies the set of GSC_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 3.5.35.4 on page 30.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

3.5.35. AO4MF_IOCTL_WAIT_EVENT

This service blocks a thread until any one of a specified set of events occurs, or until a timeout lapses, whichever occurs first. The set of possible events to wait for are specified in the structure's main, gsc, alt and io fields. All field values must be valid and at least one event must be specified. If the thread is resumed because one of the referenced events has occurred, then the bit for the respective event is the only event bit that will be set. All other event bits and fields will be zero. (Multiple event bits will be set only if the events occur simultaneously.)

> **NOTE:** A wait timeout is reported via the gsc_wait_t structure's flags field having the GSC_WAIT_FLAG_TIMEOUT flag set, rather than via an ETIMEDOUT error.

Usage

Argument	Description
request	AO4MF_IOCTL_WAIT_EVENT
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  alt;
    u32  io;
    u32  timeout_ms;
    u32  count;
} gsc_wait_t;
```

Fields	Description
flags	This must initially be zero. Upon return this indicates the reason that the thread was resumed. Refer to section 3.5.35.1 on page 29.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 3.5.35.2 on page 29.
gsc	This specifies any number of AO4MF_WAIT_GSC_* events that the thread is to wait for. Refer to section 3.5.35.3 on page 30.
alt	This is unused by the 16AO4MF driver and must be zero.
io	This specifies any number of GSC_WAIT_IO_* events that the thread is to wait for. Refer to section 3.5.35.4 on page 30.
timeout_ms	This specified the maximum amount of time, in milliseconds, that the thread is to wait for any of the referenced events. A value of zero means do not timeout at all. If non-zero, then upon return the value will be the approximate amount of time actually waited.
count	This is unused by wait event operations and must be zero.

3.5.35.1. gsc_wait_t.flags Options

Upon return from a wait request the wait structure's flags field will indicate the reason that the thread was resumed. Only one of the below options will be set.

Fields	Description
GSC_WAIT_FLAG_CANCEL	The wait request was cancelled.
GSC_WAIT_FLAG_DONE	One of the referenced events occurred.
GSC_WAIT_FLAG_TIMEOUT	The timeout period lapsed before a referenced event occurred.

3.5.35.2. gsc_wait_t.main Options

The wait structure's main field may specify any of the below primary interrupt options. These interrupt options are supported by the 16AO4MF and other General Standards products.

Fields	Description
GSC_WAIT_MAIN_DMA0	This refers to the DMA Done interrupt on DMA engine number zero.
GSC_WAIT_MAIN_DMA1	This refers to the DMA Done interrupt on DMA engine number one.
GSC_WAIT_MAIN_GSC	This refers to any of the board's firmware based interrupts.
GSC_WAIT_MAIN_OTHER	This generally refers to an interrupt generated by another device sharing the

	same interrupt as the 16AO4MF.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the 16AO4MF.
GSC_WAIT_MAIN_SPURIOUS	This refers to device interrupts which should never be generated.
GSC_WAIT_MAIN_UNKNOWN	This refers to device interrupts whose source could not be identified.

3.5.35.3. gsc_wait_t.gsc Options

The wait structure's gsc field may specify any combination of the below interrupt options. These are the board's firmware based interrupts. Applications are responsible for selecting the desired interrupt options (see AO4MF_IOCTL_IRQ_SEL, section 3.5.23, page 23).

Value	Description
AO4MF_WAIT_GSC_AO_BUF_1Q_FULL	This refers to the buffer becoming less than ¼ full.
AO4MF_WAIT_GSC_AO_BUF_3Q_FULL	This refers to the buffer becoming more than ¾ full.
AO4MF_WAIT_GSC_AO_BUF_EMPTY	This refers to the buffer becoming empty.
AO4MF_WAIT_GSC_AUTO_CAL_DONE	This refers to the completion of auto-calibration.
AO4MF_WAIT_GSC_BURST_TRIG_READY	This refers to the channel becoming ready for a burst trigger.
AO4MF_WAIT_GSC_INIT_DONE	This refers to the completion of initialization.
AO4MF_WAIT_GSC_LOAD_READY	This refers to a circular buffer becoming ready to receive data.
AO4MF_WAIT_GSC_LOAD_READY_END	This refers to a circular buffer becoming not ready to receive data.

3.5.35.4. gsc_wait_t.io Options

The wait structure's io field may specify any of the below event options. These events are generated in response to application channel data read requests.

Fields	Description
GSC_WAIT_IO_TX_ABORT	This refers to write requests which have been aborted.
GSC_WAIT_IO_TX_DONE	This refers to write requests which have been satisfied.
GSC_WAIT_IO_TX_ERROR	This refers to write requests which end due to an error.
GSC_WAIT_IO_TX_TIMEOUT	This refers to write requests which end due to the timeout period lapse.

3.5.36. AO4MF_IOCTL_WAIT_STATUS

This service counts all threads blocked via the AO4MF_IOCTL_WAIT_EVENT IOCTL service (section 3.5.35, page 28), according to the provided criteria. A match is made when a waiting thread's wait criteria matches any of the criteria specified in the structure passed to this service.

> **NOTE:** The driver itself makes use of the wait services for various internal operations. Driver initiated waits are ignored by application status requests.

Usage

Argument	Description
request	AO4MF_IOCTL_WAIT_STATUS
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
```

```
    u32  alt;
    u32  io;
    u32  timeout_ms;
    u32  count;
} gsc_wait_t;
```

Fields	Description
flags	This is unused by wait status operations.
main	This specifies the set of GSC_WAIT_MAIN_* events whose wait requests are to be counted. Refer to section 3.5.35.2 on page 29.
gsc	This specifies the set of AO4MF_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 3.5.35.3 on page 30.
alt	This is unused by the 16AO4MF driver and should be zero.
io	This specifies the set of GSC_WAIT_IO_* events whose wait requests are to be counted. Refer to section 3.5.35.4 on page 30.
timeout_ms	This is unused by wait status operations.
count	Upon return this indicates the number of waits that met any of the specified criteria.

3.5.37. AO4MF_IOCTL_XCVR_IN

This service selects between the TTL and LVDS clock and SYNC inputs.

Usage

Argument	Description
request	AO4MF_IOCTL_XCVR_IN
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO4MF_XCVR_IN_LVDS	Use LVDS signaling.
AO4MF_XCVR_IN_TTL	Use TTL signaling.

4. Operating Information

This section explains some basic operational procedures for using the 16AO4MF. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

4.1. Analog Output Configuration

The basic steps for Analog Output configuration are illustrated in the utility function noted below. The table also gives the location of the source file, the header file and the corresponding library containing the executable code.

Description	File/Name	Location	OS
Function	`ao4mf_config_ao()`	Source File	ALL
Source File	`util_config_ao.c`	`…/utils/`	ALL
Header File	`16ao4mf_utils.h`	`…/include/`	ALL
Library File	`16ao4mf_utils.a`	`…/lib/`	Linux
	`16ao4mf_utils.lib`	`…\lib\…` *	INtime

* Debug and release versions of the library are included under corresponding subdirectories.

4.2. I/O Modes

4.2.1. PIO - Programmed I/O

This involves repetitive register accesses. In this mode the driver will write data to the output buffer one value at a time. As needed, the driver will repeatedly sleep for one system time tick in order to wait for addition space in the output buffer. This process is repeated until the data is exhausted or the I/O timeout expires, whichever occurs first.

4.2.2. DMA - Block Mode DMA

For DMA transfers, hardware onboard the 16AO4MF is used to transfer the data without processor intervention. In this mode the driver first checks for available space in the output buffer. When space is available a DMA transfer is performed. Depending on the size of the write request, the driver may break the request into smaller transfers in order to insure data integrity. The breakup is based on the size of the request relative to the size of the active buffer. If the active buffer is empty, then the driver will perform a DMA transfer for up to the size of the active buffer. If the active buffer is up to ¼ full, then the driver will perform a DMA transfer for up to ¾ the size of the active buffer. If the active buffer is from ¼ to ¾ full, then the driver will perform a DMA transfer for up to the ¼ the size of the active buffer. If the active buffer is ¾ full or more, then the driver will sleep for one system timer tick and then check again. The process is repeated until the data is exhausted or the I/O timeout expires, whichever occurs first.

4.3. Debugging Aids

The driver package includes the following items useful development and or debugging aids.

4.3.1. Device Identification

When communicating with technical support complete device identification is virtually always necessary. The *id* example application is provided for this specific purpose. This is a text only console application. The output can be piped to a file, which can then be emailed to GSC technical support when requested. Locate the application as follows.

Description	File	Location	OS
Application	`id`	`…/id/`	Linux
	`id.rta`	`…\id\…` *	INtime

* Debug and release versions of the application are included under corresponding subdirectories.

4.3.2. Detailed Register Dump

Among the utility services provided is a function to generate a detailed listing of the board's registers to the console. When used, the function is typically used to verify the board's configuration. In these cases, the function should be called just prior to the first read or write operation. When intended for sending to GSC tech support, please set the *detail* argument to `1`. The function arguments are as follows. The utility location is given in the subsequent table.

Argument	Description
`fd`	This is the file descriptor used to access the device.
`detail`	If non-zero the GSC register dump will include details of each register field.

Description	File/Name	Location	OS
Function	`ao4mf_reg_list()`	Source File	ALL
Source File	`util_reg.c`	`…/utils/`	ALL
Header File	`16ao4mf_utils.h`	`…/include/`	ALL
Library File	`16ao4mf_utils.a`	`…/lib/`	Linux
	`16ao4mf_utils.lib`	`…\lib\…` *	INtime

* Debug and release versions of the library are included under corresponding subdirectories.

Document History

Revision	Description
July 9, 2018	Updated to version 3.3.79.x.0. Updated inside cover page.
June 10, 2018	Initial release, version 3.2.77.x.0.