DIO32

32-bit Discrete Digital I/O

All Form Factors
…-DIO32A

API Library
Reference Manual

Manual Revision: May 28, 2024
Driver Release Version 1.6.111.X.X

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 DIO32 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 DIO32 hardware. The API Library and device driver interfaces are primarily IOCTL based.

1.2. Acronyms

The following is a list of commonly occurring acronyms which may appear throughout this document.

Acronyms	Description
API	Application Programming Interface
DIL	Driver Interface Library
DIO	Digital I/O
DLL	Dynamic Link Library
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PMC	PCI Mezzanine Card

1.3. Definitions

The following is a list of commonly occurring terms which may appear throughout this document.

Term	Definition
…	This is a shortcut representation of the DIO32 installation directory or any of its subdirectories.
API Library	This is a library that provides application-level access to DIO32 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
DIO32	This is used as a general reference to any DIO32 supported by the API Library and device driver.
Driver	This refers to the device driver. Depending on the OS, the driver may be a user space application, a kernel mode process, or something in between. The term Driver and Device Driver are often used interchangeably.
Library	This is usually a general reference to the API Library.
Linux	This refers to the Linux operating system. Refer to the *DIO32 Linux Driver User Manual*.
Windows	This refers to the Windows operating system. Refer to the *DIO32 Windows Driver User Manual*.

1.4. Software Overview

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise DIO32 applications. The overall architecture is illustrated in Figure 1 below.

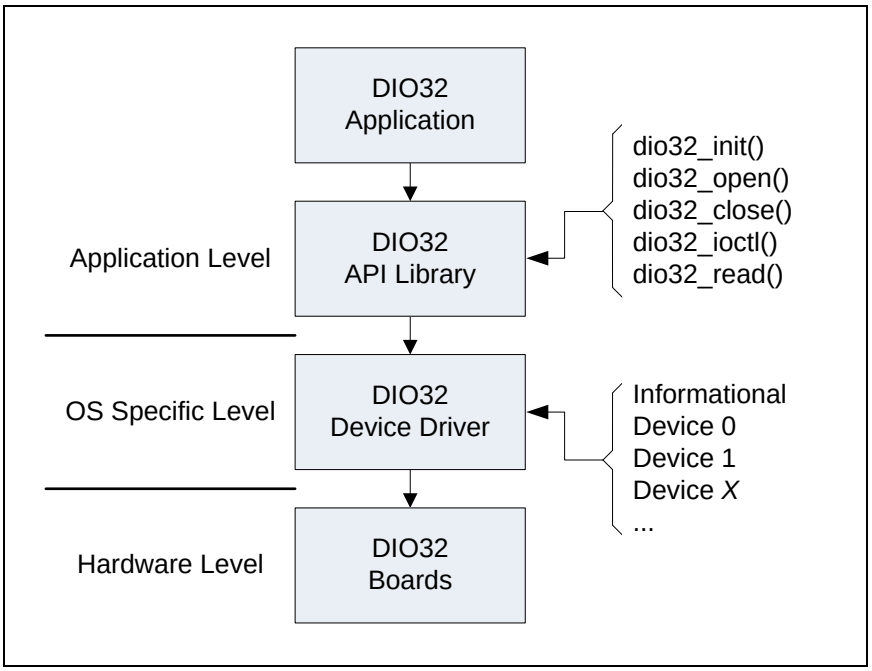

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing DIO32 hardware is via the DIO32 API Library. This library is application-level code that sits between an DIO32 application and the DIO32 device driver. With the library, applications are able to open and close a device and, while open, perform I/O control operations, and read data from the driver. For additional information refer to section 4 (page 13).

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with DIO32 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.

1.5. Hardware Overview

The DIO32 is a high-performance 32-bit discrete digital I/O interface board. The host side connection is PCI based and is either Express, 32-bit or 64-bit according to the model ordered. The external I/O interface varies per model ordered. Each of the 32 discrete signals is arbitrarily configurable as an input or an output. As inputs, the signals have configurable deglitch tolerance. Additionally, each change, high and/or low, can be conditionally configured to change the output value and tri-state condition of any or all of the board's outputs. Furthermore, each pin can also be configured to generate an interrupt on the input signal's rising and/or falling edge.

1.6. Reference Material

The following reference material may be of particular benefit in using the DIO32, the API Library and the device driver. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this device.

- The applicable *DIO32 Driver User Manual* for your operating system from General Standards Corporation.

- The applicable *DIO32 User Manual* from General Standards Corporation.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc. †
- The *PEX8311 PCI Express Bus Interface Chip* data handbook from PLX Technology, Inc. †

† PLX data books are available from PLX at the following location.

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

1.7. Licensing

For licensing information please refer to the text file `LICENSE.txt` in the root installation directory.

2. Installation

For additional information on driver installation refer to this same section number in the OS specific DIO32 driver user manual.

2.1. Host and Environment Support

For information on host and environment support refer to this same section number in the OS specific DIO32 driver user manual.

2.2. Driver and Device Information

Each driver implements an OS specific means of obtaining generic, high-level information about the driver and the installed devices. The information is given in textual format. Each line of text begins with an entry name, which is followed immediately by a colon, a space character, and an entry value. Below is an example of what is provided, followed by descriptions of each entry. This information is accessed by passing a device index value of `-1` to the API open service (section 4.6.4, page 17).

```
version: 1.6.111.50
32-bit support: yes
boards: 1
models: DIO32
ids: 0x3
```

Entry	Description
`version`	This gives the driver version number in the form `x.x.x.x`.
`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 model numbers are listed in the same order that the boards are accessed via the API Library's open function.
`ids`	This is a list identifying the values read from each board's user jumpers. The id numbers are listed in the same order that the boards are accessed via the API Library's open function.

The API's source for the text provided is as follows.

OS	Source
Linux	The file "`/proc/dio32`".
Windows	The Driver Interface Library DLL.

2.3. File List

For the list of primary files included with each release refer to this same section number in the OS specific DIO32 driver user manual.

2.4. Directory Structure

The following table describes the directory structure utilized by the installed files. During installation the directory structure is created and populated with the respective files.

NOTE: Additional or alternate directories may be installed, depending on the OS. For additional information refer to this same section number in the OS specific DIO32 driver user manual.

Directory	Description
`dio32/`	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 11) and the below listed subdirectories.
`…/api/`	This directory contains the API Library source files (section 4, page 13).
`…/docsrc/`	This directory contains the code samples from the reference manual (section 6, page 35).
`…/driver/`	This directory contains the driver and any related files (section 5, page 34).
`…/include/`	This directory contains the header files for the various libraries.
`…/lib/`	This directory contains all of the libraries built from the installed sources.
`…/samples/`	This directory contains the sample application subdirectories and all of their corresponding source files (section 9, page 40).
`…/utils/`	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 36).

2.5. Installation

For installation instructions refer to this same section number in the OS specific DIO32 driver user manual.

2.6. Removal

For removal instructions refer to this same section number in the OS specific DIO32 driver user manual.

2.7. Overall Make Script

Each DIO32 installation includes an OS specific means of building all of the build targets included in the installation. For additional information refer to this same section number in the OS specific DIO32 driver user manual.

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing DIO32 based applications.

3.1. Main Header File

Throughout the remainder of this document references are made to various header files included as part of the DIO32 driver installation. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent DIO32 specific header files. Therefore, sources may include only this one DIO32 header and make files may reference only this one DIO32 include directory.

Description	File	Location	OS
Header File	`dio32_main.h`	`…/include/`	All

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the DIO32 driver installation. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other static libraries included with the driver. Therefore, make files may reference only this one DIO32 static library and only this one DIO32 library directory.

Description	File	Location	OS
Library File	`dio32_main.a` `dio32_multi.a`	`…/lib/`	Linux
	`dio32_main.lib` `dio32_multi.lib`	`…\lib\…`	Windows

> **NOTE**: For applications using the DIO32 and no other GSC devices, link the `dio32_main.a` library. For applications using multiple GSC device types, link the `xxxx_main.a` library for one of the devices and the `xxxx_multi.a` library for the others. Linking multiple `xxxx_main.a` libraries may likely produce link errors due to duplicate symbols being defined. While it may make little or no difference, it is recommended that one choose the `xxxx_main.a` library from the driver with the largest number in positions three (x.x.X.x.x) and/or four (x.x.x.X.x) in the driver release version number.
>
> **NOTE**: The DIO32 API Library is not implemented as a static library and is thus not linked with the DIO32 Main Library. The API Library must be linked with applications according to the application's build environment.

3.2.1. Build

For information on building the Main Library refer to this same section number in the OS specific DIO32 driver user manual.

3.2.2. Additional Libraries

For information on any additional required libraries refer to this same section number in the OS specific DIO32 driver user manual.

4. API Library

The DIO32 API Library is the software interface between user applications and the DIO32 device driver. The interface is accessed by including the header file `dio32_api.h`.

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

4.1. Files

The API Library is built into a library linkable with DIO32 applications. The pertinent files are identified in the following table. Some source files are specific only to the DIO32, some are specific only to the OS and some are DIO32 and OS independent.

Description	Files	Location	OS
Source Files	`*.c, *.h`	`…/api/`	All
Header File	`dio32_api.h`	`…/include/`	All
Library File	`libdio32_api.so` †	`…/lib/` `/usr/lib/`	Linux
	`dio32_api.lib` `dio32_api.dll` ‡	`…\lib\…`	Windows

† The Linux run time executable is implemented as a shared object file.
‡ The Windows run time executable is implemented as a Windows DLL.

4.2. Build

For build instructions refer to this same section number in the OS specific DIO32 driver user manual.

4.3. Library Use

For Library usage information refer to this same section number in the OS specific DIO32 driver user manual.

4.4. Macros

The Library interface includes the following macros, which are defined in `dio32.h`.

4.4.1. IOCTL Codes

The IOCTL macros are documented in section 4.7 (page 19).

4.4.2. Registers

The following gives the complete set of DIO32 registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific DIO32 registers. Please note that the set of registers supported by any given device may vary according to model and firmware version. For the set of supported registers and their detailed definitions refer to the appropriate *DIO32 User Manual*.

> **NOTE**: Refer to the output of the "id" sample application (`…/id/`) for a complete list of the registers supported by the device being accessed.

Macros	Description
DIO32_GSC_BCR	Board Control Register (BCR)
DIO32_GSC_BSR	Board Status Register (BSR)
DIO32_GSC_D00HLOMR †	D0 Hi-Low Output Mask Register (D00LHOMR) †
DIO32_GSC_D00HLOTSR †	D0 Hi-Low Output Tri-State Register (D00LHOTSR) †
DIO32_GSC_D00HLOVR †	D0 Hi-Low Output Value Register (D00LHOVR) †
DIO32_GSC_D00LHOMR †	D0 Low-Hi Output Mask Register (D00LHOMR) †
DIO32_GSC_D00LHOTSR †	D0 Low-Hi Output Tri-State Register (D00LHOTSR) †
DIO32_GSC_D00LHOVR †	D0 Low-Hi Output Value Register (D00LHOVR) †
DIO32_GSC_DGCR	Deglitch Control Register (DGCR)
DIO32_GSC_DGIR	Deglitch Input Register (DGIR)
DIO32_GSC_FR	Features Register (FR)
DIO32_GSC_FRR	Firmware Revision Register (FRR)
DIO32_GSC_HLIER	Hi-Low Interrupt Enable Register (HLIER)
DIO32_GSC_HLISCR	Hi-Low Interrupt Status/Clear Register (HLISCR)
DIO32_GSC_IODR	I/O Direction Register (IODR)
DIO32_GSC_ITR	Interrupt Type Register (ITR)
DIO32_GSC_LHIER	Low-Hi Interrupt Enable Register (LHIER)
DIO32_GSC_LHISCR	Low-Hi Interrupt Status/Clear Register (LHISCR)
DIO32_GSC_OCR	Output Clock Register (OCR)
DIO32_GSC_OTSR	Output Tri-State Register (OTSR)
DIO32_GSC_OVR	Output Value Register (OVR)
DIO32_GSC_PC1FR ‡	Programmable Clock 1 Frequency Register ‡
DIO32_GSC_PC1PWMDR ‡	Programmable Clock 1 Pulse Width Modulator Divider Register ‡
DIO32_GSC_PC1R ‡	Programmable Clock 1 Register ‡
DIO32_GSC_PIR	Pin Input Register (PIR)

† There is a corresponding register for each of the 32 GPIO pins numbered from 00 through 31.

‡ There are three programmable clocks, numbered 1, 2 and 3, each with three control registers.

4.4.2.2. 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 driver header file gsc_pci9056.h, which is automatically included via dio32_api.h.

4.4.2.3. PLX 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 driver header file gsc_pci9056.h, which is automatically included via dio32_api.h.

4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used. For additional information refer to section 4.7 (page 19).

4.5.1. dio32_port_t

This structure is used for configuring numerous device settings which are applicable on a per port basis. For usage information refer to section 8.4, page 38.

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	Description
port	This specifies the single port to access and is the primary means of selecting a single port. When this method is used and optional, the mask field is ignored. When optional and equal to -1, port selections are made via the mask field.
action	This must be one of the following. Option: DIO32_ACT_MX — Perform a read-modify-write of the referenced feature settings. This is valid only if the settings for a single port require just one bit. DIO32_ACT_RX — Read the value and return its settings. DIO32_ACT_TX — Apply the value provided to the referenced port(s).
value	This contains the port value or values to apply or retrieve. For feature settings represented by a single bit per port pin, this is a bitmap. For multi-bit value sets, this is one of the supported values and for read requests the value returned is for the lowest referenced port pin.
mask	This is a mask of the port pins to be accessed. If zero, then no action takes place. For multi-bit value sets, RX requests access only the lowest referenced port pin.

4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A return value less than zero always reflects an error condition. The table below summarizes the error status values. For the I/O function, read, non-negative return values reflect the number of bytes transferred between the application and the interface. A value equal to the requested transfer size indicates complete success. Return values less than the requested transfer size indicate that the I/O timeout expired. For the other API function calls a return value of zero indicates success.

Return Value	Description	OS
-1 to -499	This is the value "(-errno)" (see errno.h).	All
-500 to -999	This is the value returned from the Driver Interface Library. †	Windows
>= -1000	This is "(int)(GetLastError()+1000)" forced to a negative value.	

† Applicable error codes, if any, are defined in the header os_common.h.

4.6.1. dio32_close()

This function is the entry point to close a connection made via the API's open call (section 4.6.4, page 17). The device is put in an initialized state before this call returns.

Prototype

```
int dio32_close(int fd);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 17).

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

Example

```
#include <stdio.h>

#include "dio32_dsl.h"

int dio32_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = dio32_close(fd);

    if (ret)
        printf("ERROR: dio32_close() returned %d\n", ret);

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

4.6.2. dio32_init()

This function is the entry point to initializing the DIO32 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 dio32_init(void);
```

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

Example

```
#include <stdio.h>

#include "dio32_dsl.h"

int dio32_init_dsl(void)
{
    int errs;
    int ret;

    ret = dio32_init();

    if (ret)
        printf("ERROR: dio32_init() returned %d\n", ret);
```

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

4.6.3. dio32_ioctl()

This function is the entry point to performing setup and control operations on a DIO32. 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, which are defined in section 4.7 (page 19).

NOTE: IOCTL operations are not supported for an open on device index `-1`.

Prototype

```
int dio32_ioctl(int fd, int request, void* arg);
```

Argument	Description
`fd`	This is the file descriptor obtained from the open service (section 4.6.4, page 17).
`request`	This specifies the desired operation to be performed (section 4.7, page 19).
`arg`	This is specific to the IOCTL operation specified by the `request` argument.

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

Example

```
#include <stdio.h>

#include "dio32_dsl.h"

int dio32_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = dio32_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: dio32_ioctl() returned %d\n", ret);

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

4.6.4. dio32_open()

This function is the entry point to open a connection to a DIO32 board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int dio32_open(int device, int share, int* fd);
```

Argument	Description
device	This is the zero-based index of the DIO32 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 2.2, page 10).

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

Example

```
#include <stdio.h>

#include "dio32_dsl.h"

int dio32_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = dio32_open(device, share, fd);

    if (ret)
        printf("ERROR: dio32_open() returned %d\n", ret);

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

4.6.4.1. Access Modes

The value of the share argument determines the device access mode, as follows.

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.

4.6.5. dio32_read()

This function is the entry point to reading data from an open performed on device index `-1`. The function reads up to `bytes` bytes.

> **NOTE:** The read service has no functionality for reading from DIO32 devices. Attempts to read from DIO32 devices will return an error.

Prototype

```
int dio32_read(int fd, void *dst, size_t bytes);
```

Argument	Description
`fd`	This is the file descriptor obtained from the open service (section 4.6.4, page 17).
`dst`	The data read is put here.
`bytes`	This is the desired number of bytes to read.

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

Example

```
#include <stdio.h>

#include "dio32_dsl.h"

int dio32_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = dio32_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: dio32_read() returned %d\n", ret);

    if (qty)
        qty[0]  = (ret < 0) ? 0 : (size_t) ret;

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

4.7. IOCTL Services

The DIO32 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `dio32_ioctl()` function arguments.

4.7.1. DIO32_IOCTL_DEGLITCH_FILTER

This service configures the Deglitch Filter count, which is used as a form of oversampling. The count specifies the number of sequential samples separated by the Deglitch Sample Rate period over which the input must be stable before the transition is recorded by the input latch. Refer to the Deglitch Feature for additional information (section 8.2, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_DEGLITCH_FILTER
arg	s32*

Valid argument values are in the range of zero to 0xFF, or `-1` to retrieve the current setting. A value of zero results in the deglitched input being the same as the raw input.

4.7.2. DIO32_IOCTL_DEGLITCH_RATE

This service configures the Deglitch Rate period, which is used as a form of oversampling. The count specifies the number of 100ns intervals between which consecutive samplings of the input are made. Refer to the Deglitch Feature for additional information (section 8.2, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_DEGLITCH_RATE
arg	s32*

Valid argument values are in the range of zero to 0xFFFFFF, or `-1` to retrieve the current setting. A value of zero results in the deglitched input being the same as the raw input.

4.7.3. DIO32_IOCTL_HL_OUT_MASK

This service configures the High-Low Output Mask for the Input Response Feature. Refer to the Input Response Feature for additional information (section 8.3, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_HL_OUT_MASK
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies the input port pin who's Hi-Low Output Mask is being configured.

action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the Output Mask bits to be accessed. If zero, then no action takes place.

4.7.4. DIO32_IOCTL_HL_OUT_TRI_STATE

This service configures the High-Low Output Tri-State setting for the Input Response Feature. Refer to the Input Response Feature for additional information (section 8.3, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_HL_OUT_TRI_STATE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies the input port pin who's Hi-Low Output Tri-State setting is being configured.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the Output Tri-State bits to be accessed. If zero, then no action takes place.

4.7.5. DIO32_IOCTL_HL_OUT_VALUE

This service configures the High-Low Output Value for the Input Response Feature. Refer to the Input Response Feature for additional information (section 8.3, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_HL_OUT_VALUE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies the input port pin who's Hi-Low Output Value is being configured.

action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the Output Value bits to be accessed. If zero, then no action takes place.

4.7.6. DIO32_IOCTL_INITIALIZE

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

Usage

Argument	Description
request	DIO32_IOCTL_INITIALIZE
arg	Not used.

4.7.7. DIO32_IOCTL_IO_IN_DEGLITCHED_GET

This service retrieves the latest deglitched input. Refer to the Deglitch Feature for additional information (section 8.2, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_IO_IN_DEGLITCHED_GET
arg	u32*

Valid argument values are in the range of zero to 0xFFFFFFFF.

4.7.8. DIO32_IOCTL_IO_IN_GET

This service retrieves the instantaneous I/O value at the port pins.

Usage

Argument	Description
request	DIO32_IOCTL_IO_IN_GET
arg	u32*

Valid argument values are in the range of zero to 0xFFFFFFFF.

4.7.9. DIO32_IOCTL_IO_OUT_CLOCK

This service configures the output clock setting for one or all valid port pins. Clock outputs can be configured only for the lower port number of each group of four ports. The clock output supersedes the programmed discrete outputs. As with the discrete outputs, the selected clock appears at the port pin only if it is an output and is not tri-stated.

Usage

Argument	Description
request	DIO32_IOCTL_IO_OUT_CLOCK
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies a single port to access and is the primary means of selecting a single port. If set to -1 this field is ignored and port selections are made via the mask field.
action	This is DIO32_ACT_RX/TX to read or write the setting.
value	This is a value from the below table.
mask	This is a mask of the port pins to be accessed. If zero, then no action takes place. For RX requests only the lowest referenced port pin is accessed.

Valid oscillator options are those listed in the below table.

Value	Description
DIO32_IO_OUT_CLOCK_NONE	Configure the ports for discrete output.
DIO32_IO_OUT_CLOCK_1	Configure the ports to output clock 1.
DIO32_IO_OUT_CLOCK_2	Configure the ports to output clock 2.
DIO32_IO_OUT_CLOCK_3	Configure the ports to output clock 3.

4.7.10. DIO32_IOCTL_IO_OUT_DIR

This service configures the GPIO direction (input vs. output) for the port pins.

Usage

Argument	Description
request	DIO32_IOCTL_IO_OUT_DIR
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies a single port to access and is the primary means of selecting a single port. If set to -1 this field is ignored and port selections are made via the mask field.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the port pins to be accessed. If zero, then no action takes place.

4.7.11. DIO32_IOCTL_IO_OUT_TRI_STATE

This service configures the output port tri-state setting for the output port pins. Settings are retained even if a port pin is configured as an input.

Usage

Argument	Description
request	DIO32_IOCTL_IO_OUT_TRI_STATE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies a single port to access and is the primary means of selecting a single port. If set to -1 this field is ignored and port selections are made via the mask field.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the port pins to be accessed. If zero, then no action takes place.

4.7.12. DIO32_IOCTL_IO_OUT_VAL

This service configures the output value for the output ports. Settings are retained even if a port pin is configured as an input.

Usage

Argument	Description
request	DIO32_IOCTL_IO_OUT_VAL
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies a single port to access and is the primary means of selecting a single port. If set to -1 this field is ignored and port selections are made via the mask field.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.

mask	This is a mask of the port pins to be accessed. If zero, then no action takes place.

4.7.13. DIO32_IOCTL_IRQ_HL_ENABLE

This service enables or disables the high-to-low transition interrupts.

Usage

Argument	Description
request	DIO32_IOCTL_IRQ_HL_ENABLE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies a single port to access and is the primary means of selecting a single port. If set to -1 this field is ignored and port selections are made via the mask field.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the port pins to be accessed. If zero, then no action takes place.

4.7.14. DIO32_IOCTL_IRQ_LH_ENABLE

This service enables or disables the low-to-high transition interrupts.

Usage

Argument	Description
request	DIO32_IOCTL_IRQ_LH_ENABLE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies a single port to access and is the primary means of selecting a single port. If set to -1 this field is ignored and port selections are made via the mask field.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.

mask	This is a mask of the port pins to be accessed. If zero, then no action takes place.

4.7.15. DIO32_IOCTL_LED*n*

These services configure the state of the board's LEDs. The LEDs and corresponding service macros are given in the below table.

LED Number	Service Macro
0	DIO32_IOCTL_LED0
1	DIO32_IOCTL_LED1
2	DIO32_IOCTL_LED2
3	DIO32_IOCTL_LED3
4	DIO32_IOCTL_LED4
5	DIO32_IOCTL_LED5

Usage

Argument	Description
request	DIO32_IOCTL_LED*n*
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DIO32_LED_OFF	This turns the LED off.
DIO32_LED_GREEN	This turns on the LED's green color option.
DIO32_LED_RED	This turns on the LED's red color option.
DIO32_LED_BOTH	This turns on both of the LED's color options.

4.7.16. DIO32_IOCTL_LH_OUT_MASK

This service configures the Low-High Output Mask for the Input Response Feature. Refer to the Input Response Feature for additional information (section 8.3, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_LH_OUT_MASK
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies the input port pin who's Low-High Output Mask is being configured.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.

value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the Output Mask bits to be accessed. If zero, then no action takes place.

4.7.17. DIO32_IOCTL_LH_OUT_TRI_STATE

This service configures the Low-High Output Tri-State setting for the Input Response Feature. Refer to the Input Response Feature for additional information (section 8.3, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_LH_OUT_TRI_STATE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies the input port pin who's Low-High Output Tri-State setting is being configured.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.
value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the Output Tri-State bits to be accessed. If zero, then no action takes place.

4.7.18. DIO32_IOCTL_LH_OUT_VALUE

This service configures the Low-High Output Value for the Input Response Feature. Refer to the Input Response Feature for additional information (section 8.3, page 37).

Usage

Argument	Description
request	DIO32_IOCTL_LH_OUT_VALUE
arg	dio32_port_t*

Definition

```
typedef struct
{
    s32  port;
    s32  action;
    u32  value;
    u32  mask;
} dio32_port_t;
```

Fields	**Description** (For additional information refer to section 8.4, page 38.)
port	This specifies the input port pin who's Low-High Output Value is being configured.
action	This is DIO32_ACT_MX/RX/TX to modify, read or write the setting.

value	This contains the port values to apply or the port values retrieved.
mask	This is a mask of the Output Value bits to be accessed. If zero, then no action takes place.

4.7.19. DIO32_IOCTL_QUERY

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

Usage

Argument	Description
request	DIO32_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
DIO32_QUERY_BUS_WIDTH	This returns the board's PCI interface bus width in bits, which is either 32 or 64.
DIO32_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
DIO32_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_DIO32.
DIO32_QUERY_FORM_FACTOR	This indicates the board's native form factor. The options are listed below.
DIO32_QUERY_IO_BITS	This indicates the number of supported I/O port pins.
DIO32_QUERY_USER_JUMPER_QTY	This indicates the number of user jumpers.
DIO32_QUERY_USER_JUMPER_SENSE	This indicates which bit value indicates that the jumper is installed.
DIO32_QUERY_USER_JUMPER_VALUE	This indicates the value from reading the user jumpers.
DIO32_QUERY_XCVR_TYPE	This indicates the board's transceiver type. The options are listed below.

Valid return values are as indicated in the above table and as given in the below table.

Value	Description
DIO32_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

Valid return values for the Form Factor query are as follows.

Value	Description
DIO32_QUERY_FF_CPCI	The form factor is Compact PCI.
DIO32_QUERY_FF_PC104P	The form factor is PC/104+.
DIO32_QUERY_FF_PCI	The form factor is PCI.
DIO32_QUERY_FF_PMC	The form factor is PMC.
DIO32_QUERY_FF_UNKNOWN	The form factor is unknown.

Valid return values for the Transceiver query are as follows.

Value	Description
DIO32_QUERY_XCVR_RS485	The board has RS-485 transceivers.
DIO32_QUERY_XCVR_UNKNOWN	This indicates that the transceiver type is unknown.

4.7.20. DIO32_IOCTL_REG_MOD

This service performs a read-modify-write of a DIO32 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `dio32.h` for the complete list of GSC firmware registers.

Usage

Argument	Description
request	DIO32_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 bit is modified. If a bit here is zero, then the respective register bit is unmodified.

4.7.21. DIO32_IOCTL_REG_READ

This service reads the value of a DIO32 register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `dio32.h` and to `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
request	DIO32_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.

4.7.22. DIO32_IOCTL_REG_WRITE

This service writes a value to a DIO32 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `dio32.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	DIO32_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.

4.7.23. DIO32_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via `DIO32_IOCTL_WAIT_EVENT` IOCTL calls (section 4.7.24, page 31), according to the provided criteria. When a blocked thread is waiting for any event specified in the structure, then the thread is resumed.

Usage

Argument	Description
request	DIO32_IOCTL_WAIT_CANCEL
arg	dio32_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  hi_low;
    u32  low_hi;
    u32  timeout_ms;
    u32  count;
} dio32_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 4.7.24.2 on page 32.

gsc	This specifies the set of DIO32_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.24.3 on page 32.
hi_low	This specifies the set High-to-Low interrupt events whose wait requests are to be cancelled. Refer to section 4.7.24.4 on page 32.
low_hi	This specifies the set Low-to-High interrupt events whose wait requests are to be cancelled. Refer to section 4.7.24.5 on page 32.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.24. DIO32_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, hi_low and low_hi 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: The service waits only for the first of the specified events, not for all specified events.

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	DIO32_IOCTL_WAIT_EVENT
arg	dio32_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  hi_low;
    u32  low_hi;
    u32  timeout_ms;
    u32  count;
} dio32_wait_t;
```

Fields	Description
flags	This must initially be zero. Upon return this indicates the reason that the thread was resumed. Refer to section 4.7.24.1on page 32.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.24.2 on page 32.
gsc	This specifies any number of DIO32_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.24.3 on page 32.
hi_low	This specifies the set High-to-Low interrupt events that the thread is to wait for. Each bit, zero through 31, refers to the corresponding input port. Valid values are from zero to 0xFFFFFFFF.
low_hi	This specifies the set Low-to-High interrupt events that the thread is to wait for. Each bit, zero through 31, refers to the corresponding input port. Valid values are from zero to 0xFFFFFFFF.

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.

4.7.24.1. dio32_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.

4.7.24.2. dio32_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 DIO32 and other General Standards products.

Fields	Description
GSC_WAIT_MAIN_GSC	This refers to any of the Interrupt Control/Status Register interrupts.
GSC_WAIT_MAIN_OTHER	This generally refers to an interrupt generated by another device sharing the same interrupt as the DIO32.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the DIO32.
GSC_WAIT_MAIN_SPURIOUS	This refers to board interrupts which should never be generated.
GSC_WAIT_MAIN_UNKNOWN	This refers to board interrupts whose source could not be identified.

4.7.24.3. dio32_wait_t.gsc Options

The wait structure's gsc field may specify any combination of the below interrupt options. These are the interrupt options referenced in the interrupt enable registers. Applications are responsible for enabling the desired interrupt options. Refer to DIO32_IOCTL_IRQ_HL_ENABLE and DIO32_IOCTL_IRQ_LH_ENABLE (sections 4.7.13, page 25, and section 4.7.14, page 25, respectively).

Value	Description
DIO32_WAIT_GSC_HI_LOW	This refers to any High-to-Low interrupt from any input port.
DIO32_WAIT_GSC_LOW_HI	This refers to any Low-to-High interrupt from any input port.

4.7.24.4. dio32_wait_t.hi_low Options

The wait structure's hi_low field refers to the interrupts generated by high-to-low transitions on the input pins. The field may specify any combination of bits referencing any combination of port pins. Valid field values are from zero to 0xFFFFFFFF.

4.7.24.5. dio32_wait_t.low_hi Options

The wait structure's low_hi field refers to the interrupts generated by low-to-high transitions on the input pins. The field may specify any combination of bits referencing any combination of port pins. Valid field values are from zero to 0xFFFFFFFF.

4.7.25. DIO32_IOCTL_WAIT_STATUS

This service counts all threads blocked via the DIO32_IOCTL_WAIT_EVENT IOCTL service (section 4.7.24, page 31), 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.

Usage

Argument	Description
request	DIO32_IOCTL_WAIT_STATUS
arg	dio32_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  hi_low;
    u32  low_hi;
    u32  timeout_ms;
    u32  count;
} dio32_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 4.7.24.2 on page 32.
gsc	This specifies the set of DIO32_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.24.3 on page 32.
hi_low	This specifies the set High-to-Low interrupt events whose wait requests are to be counted. Valid values are from zero to 0xFFFFFFFF.
low_hi	This specifies the set Low-to-High interrupt events whose wait requests are to be counted. Valid values are from zero to 0xFFFFFFFF.
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.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location	OS
Source Files	`*.c`, `*.h`	…/driver/	Linux
Header File	`dio32.h`	…/driver/	Linux
Driver File	`dio32.ko` †	…/driver/	Linux (kernels version 2.6 and later)
	`dio32.o` †	…/driver/	Linux (kernels version 2.4 and earlier)
	`dio32_dil.lib` `dio32_dil.dll`	…\lib\…	Windows
	`dio32_9056.sys` ‡	…\driver\…	

† The Linux run time executable is implemented as a loadable kernel module.
‡ The Windows run time executable is implemented as a driver `.sys` file.

5.2. Build

For instructions on building the driver refer to this same section number in the OS specific DIO32 driver user manual.

5.3. Startup

For instructions on starting the driver executable refer to this same section number in the OS specific DIO32 driver user manual.

5.4. Verification

For instructions on verifying that the driver has been loaded and is running refer to this same section number in the OS specific DIO32 driver user manual.

5.5. Version

For instructions on obtaining the driver version number refer to this same section number in the OS specific DIO32 driver user manual.

5.6. Shutdown

For instructions on terminating the driver executable refer to this same section number in the OS specific DIO32 driver user manual.

6. Document Source Code Examples

The source code examples included in this document are built into a statically linkable library usable with console applications. The purpose of these files is to verify that the documentation samples compile and to provide a library of working sample code to assist in a user's learning curve and application development effort.

6.1. Files

The library files are summarized in the table below.

Description	Files	Location	OS
Source Files	`*.c`, `*.h`, `makefile` …	…`/docsrc/`	All
Header File	`dio32_dsl.h`	…`/include/`	All
Library File	`dio32_dsl.a`	…`/lib/`	Linux
	`dio32_dsl.lib`	…`\lib\`…	Windows

6.2. Build

For library build instructions refer to this same section number in the OS specific DIO32 driver user manual.

6.3. Library Use

For library usage information refer to this same section number in the OS specific DIO32 driver user manual.

7. Utilities Source Code

The API Library installation includes a body of utility source code designed to aid in the understanding and use of the interface calls and IOCTL services. Utility sources are also included for device independent and common, general-purpose services. Most of the utilities are implemented as visual wrappers around the corresponding services to facilitate structured console output for the sample applications. The utility sources are compiled and linked into static libraries to simplify their use. An additional purpose of these utility services is to provide a library of working sample code to assist in a user's learning curve and application development effort.

For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `dio32_open()` there is the utility file `open.c` containing the utility function `dio32_open_util()`. The naming pattern is as follows: API function `dio32_xxxx()`, utility file name `xxxx.c`, utility function `dio32_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 `DIO32_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `dio32_query()`. The naming pattern is as follows: IOCTL code `DIO32_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `dio32_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location	OS
Source Files	`*.c`, `*.h` …	…`/utils/`	All
Header File	`dio32_utils.h`	…`/include/`	All

Library File	`dio32_utils.a`
	`gsc_utils.a`
	`os_utils.a`
	`plx_utils.a`
		…`/lib/`	Linux

	`dio32_utils.lib`
	`gsc_utils.lib`
	`os_utils.lib`
	`plx_utils.lib`
		…`\lib\`…	Windows

7.2. Build

For library build instruction refer to this same section number in the OS specific DIO32 driver user manual.

7.3. Library Use

For library usage information refer to this same section number in the OS specific DIO32 driver user manual.

8. Operating Information

This section explains some basic operational procedures for using the DIO32. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

8.1. Debugging Aids

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

8.1.1. 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.exe`	`…\id\…`	Windows

8.1.2. Detailed Register Dump

Among the utility services provided is a function to generate a detailed listing of device registers to the console. When used, the function is typically used to verify device configuration. In these cases, the function should be called after complete device configuration and before the first I/O call. When intended for sending to GSC tech support, please set the *detail* arguments 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 register dump will include details of each register field.

Description	File/Name	Location	OS
Function	`dio32_reg_list()`	Source File	All
Source File	`util_reg.c`	`…/utils/`	All
Header File	`dio32_utils.h`	`…/include/`	All
Library File	`dio32_utils.a`	`…/lib/`	Linux
	`dio32_utils.lib`	`…\lib\…`	Windows

8.2. Deglitch Feature

The board's Deglitch Feature implements a type of oversampling that acts to limit the recognition of glitches on the input pins as valid, stable transitions. The feature includes two settings that apply to each input pin individually. The first setting is the Glitch Filter. This specifies the number of times the sampling of the input must have the same value before it is considered a valid transition. If the sampled value changes during the count, then the transition is ignored as a glitch and the count is reset awaiting a new transition. The other setting is the Sample Rate. This specifies the period of time that is inserted between each Glitch Filter count. If either setting is zero, then the deglitched input is the same as the raw input.

8.3. Input Response Feature

The Input Response Feature is the means by which a transition on an input can cause a change on one or more outputs. Each of the 32 inputs has an identical set of three control registers for both high-to-low input transitions and for low-to-high input transitions. The result is 64 sets of control registers, each totally independent of the other. Each

set's Output Mask Register controls which set of outputs are to be affected. The mask may specify any combination of the 32 pins, though the feature has no immediate effect on port pins configured as inputs. If a mask bit is set, then it enables the corresponding output port pin to be affected. If a mask bit is clear, then the corresponding output port pin is untouched. If all mask bits are zero, then that respective transition has no side effects. The Output Value Register bits enabled via the Output Mask Register are applied to the corresponding outputs when a respective transition occurs. Likewise, the Output Tri-State Register bits enabled via the Output Mask Register are applied to the corresponding outputs when a respective transition occurs. Thus, a transition on an input can update the value and tri-state condition of any number of desired outputs.

8.4. dio32_port_t Usage

The `dio32_port_t` structure has two primary use cases. The first is when the `port` field is required. The second is when it is optional.

8.4.1. The `port` field is required.

When configuring any of the Input Response Feature registers the `port` field is required and must specify a valid port index. The remaining fields dictate the operation to perform and the register bits to be modified.

Example 1: `DIO32_IOCTL_LH_OUT_MASK`

Fields	Value
port	4
action	DIO32_ACT_TX
value	0x00000001
mask	0x0000000F

This example configures the Low-High Output Mask for input port four. The operation changes the lower four bits of the mask to equal the value 0x1. The other bits are unchanged. (This is equivalent to using `DIO32_ACT_MX`.) As a result, a low-to-high transition on input port D4 will apply settings to port pin D0, but not to D1, D2 or D3. The transition may affect other port pins as well, but it isn't evident from just thus one example.

Example 2: `DIO32_IOCTL_HL_OUT_VALUE`

Fields	Value
port	6
action	DIO32_ACT_RX
value	0xXXXXXXXX
mask	0xFFFF0000

This example reads the High-Low Output Value for input port six. The operation reads the entire register value, but returns only the upper 16 bits. Being masked off, the lower 16 bits are returned as zero.

8.4.2. The `port` field is optional.

For those IOCTL services using this structure, but which aren't configuring any of the Input Response Feature registers the `port` field is optional. The value assigned may be any valid port index or `-1`. If `-1`, then the `port` field is ignored and the `mask` field is used to indicate the port pins to be accessed. If a port index is given, then the `mask` field is ignored.

Example 1: `DIO32_IOCTL_IO_OUT_CLOCK`

Fields	Value
port	-1
action	DIO32_ACT_TX
value	DIO32_IO_OUT_CLOCK_1
mask	0x00001111

This example configures the I/O Output Clock setting. Since for this service the `port` field is optional and set to `-1`, the port selections come from the `mask` field. In this case the setting is applied to ports D0, D4, D8 and D12. The settings for the remaining port pins are unchanged.

Example 2: `DIO32_IOCTL_IO_OUT_CLOCK`

Fields	Value
port	6
action	DIO32_ACT_RX
value	XXX
mask	Ignored

This example attempts to retrieve the I/O Output Clock setting for port six. The request fails as port pin six does not support an I/O Output Clock setting.

Example 3: `DIO32_IOCTL_IO_OUT_DIR`

Fields	Value
port	-1
action	DIO32_ACT_MX
value	0x0000FFFF
mask	0xFFFFFFFF

This example configures the I/O Output Direction. Since for this service the `port` field is optional and set to `-1`, the port selections come from the `mask` field. The `mask` field has all bits set so the entire `value` field is applied, configuring the upper 16 port pins as inputs and the lower 16 port pins as outputs.

9. Sample Applications

For information on the sample applications refer to this same section number in the OS specific DIO32 driver user manual.

Document History

Revision	Description
May 28, 2024	Updated to version 1.6.111.X.X. Updated the information for the open and close calls.
October 14, 2022	Updated to version 1.5.101.X.X. Updated the information for the open and close calls.
March 8, 2021	Initial release, version 1.4.93.X.X.