OPTO32

24 Optically Isolated Inputs, 8 Optically Isolated Outputs

All Form Factors
…-OPTO32/A/B/C/D

API Library
Reference Manual

Manual Revision: June 20, 2023
Driver Release Version 8.7.104.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 OPTO32 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 OPTO32 hardware. The API Library and driver interfaces are based on the board's functionality and 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
COS	Change of State
CPCI	Compact PCI
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 OPTO32 installation directory or any of its subdirectories.
API Library	This is a library that provides application-level access to OPTO32 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
D23	This is a reference that means "bit 23."
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 *OPTO32 Linux Driver User Manual*.
OPTO32	This is used as a general reference to any OPTO32 supported by the API Library and device driver.
Windows	This refers to the Windows operating system. Refer to the *OPTO32 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 OPTO32 applications. The overall architecture is illustrated in Figure 1 below.

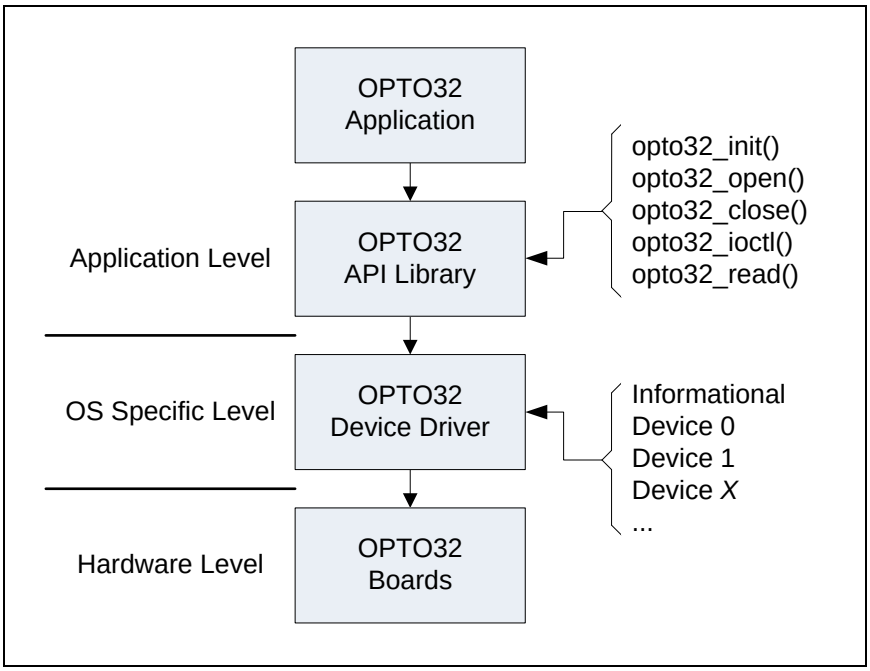

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing OPTO32 boards is via the OPTO32 API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 13). With the library, applications are able to open and close a device and, while open, perform I/O control and read and write operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with OPTO32 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 OPTO32 is a high performance optically isolated I/O board with Change of State (COS) detection on all inputs. The board includes 24 optically isolated inputs and eight optically isolated outs. All inputs are controlled by a global debounce timer. The debounce period consists of three sampling intervals, where the interval is configurable in 100ns increments from 200ns to just under 215 seconds. The denounced input can be read at any time. Also, each COS input can be independently configured to generate an interrupt on either a rising or falling state transition. The D23 input has the added capability of generating an interrupt after receipt of from 1 to 64K low-to-high state changes. The eight outputs include four with normal output capability and four with high output capability.

1.6. Reference Material

The following reference material may be of particular benefit in using the OPTO32, 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 *OPTO32 Driver User Manual* for your operating system from General Standards Corporation.

- The applicable *OPTO32 User Manual* from General Standards Corporation.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc. *
- The *PCI9060ES PCI Bus Master 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 information on driver installation refer to this same section number in the OS specific OPTO32 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 OPTO32 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 boards. The information is given in ASCII text format. Each entry includes an entry name followed immediately by a colon, a space character, and the 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 16).

```
version: 8.7.104.47
32-bit support: yes
boards: 1
models: OPTO32
```

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.

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

OS	Source
Linux	The file "`/proc/opto32`".
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 OPTO32 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 OPTO32 driver user manual.

Directory	Description
opto32/	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 source files for the code samples given in this document (section 6, page 29).
…/driver/	This directory contains the device driver source files (section 1.7, page 9).
…/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 32).
…/utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 30).

2.5. Installation

For installation instructions refer to this same section number in the OS specific OPTO32 driver user manual.

2.6. Removal

For removal instructions refer to this same section number in the OS specific OPTO32 driver user manual.

2.7. Overall Make Script

Each OPTO32 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 OPTO32 driver user manual.

2.8. Environment Variables

For environment variable information refer to this same section number in the OS specific OPTO32 driver user manual.

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing OPTO32 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 OPTO32 driver archive. 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 OPTO32 specific header files. Therefore, sources may include only this one OPTO32 header and make files may reference only this one OPTO32 include directory.

Description	File	Location	OS
Header File	`opto32_main.h`	`…/include/`	All

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included with the driver. 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 OPTO32 static library and only this one OPTO32 library directory.

Description	File	Location	OS
Library File	`opto32_main.a` `opto32_multi.a`	`…/lib/`	Linux
	`opto32_main.lib` `opto32_multi.lib`	`…\lib\…`	Windows

> **NOTE**: For applications using the OPTO32 and no other GSC devices, link the `opto32_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 OPTO32 API Library is implemented as a shared or dynamically liked library and is thus not linked with the OPTO32 Main Library. Refer to the OS specific driver user manual for information on linking with the respective OS specific OPTO32 Main Library.

3.2.1. Build

For information on building the Main Library refer to this same section number in the OS specific OPTO32 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 OPTO32 driver user manual.

4. API Library

The OPTO32 API Library is the software interface between user applications and the OPTO32 device driver. The interface is accessed by including the header file `opto32_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 OPTO32 applications. The pertinent files are identified in the following table. Some source files are specific only to the OPTO32, some are specific only to the OS and some are OPTO32 and OS independent.

Description	Files	Location	OS
Source Files	`*.c`, `*.h`	`…/api/`	All
Header File	`opto32_api.h`	`…/include/`	All
Library File	`libopto32_api.so` †	`…/lib/` `/usr/lib/`	Linux
	`opto32_api.lib` `opto32_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 DLL.

4.2. Build

For build instructions refer to this same section number in the OS specific OPTO32 driver user manual.

4.3. Library Use

For Library usage information refer to this same section number in the OS specific OPTO32 driver user manual.

4.4. Macros

The Library interface includes the following macros, which are defined in `opto32.h`.

4.4.1. IOCTL Codes

The IOCTL macros are documented in section 4.7 (page 18).

4.4.2. Registers

The following gives the complete set of OPTO32 registers.

4.4.2.1. GSC Registers

The following table gives the OPTO32 firmware register macros used by the OPTO32 API Library interface.

Macros	Description
`OPTO32_GSC_BCSR`	Board Control/Status Register (BCSR)
`OPTO32_GSC_CDR`	Clock Divider Register (CDR)
`OPTO32_GSC_CIER`	COS Interrupt Enable Register (CIER)
`OPTO32_GSC_COSR`	Change of State Register (COSR)
`OPTO32_GSC_CPR`	COS Polarity Register (CPR)

OPTO32_GSC_ODR	Output Data Register (ODR)
OPTO32_GSC_RECR	Receive Event Counter Register (RECR)
OPTO32_GSC_RDR	Receive Data Register (RDR)

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 header file gsc_pci9056.h, which is automatically included via opto32_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 header file gsc_pci9056.h, which is automatically included via opto32_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 18).

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 and write, 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. opto32_close()

This function is the entry point to close a connection made via the API's open call (section 4.6.4, page 16). The device is put in an initialized state before this call returns.

Prototype

```
int opto32_close(int fd);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 16).

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

Example

```
#include <stdio.h>
```

```
#include "opto32_dsl.h"

int opto32_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = opto32_close(fd);

    if (ret)
        printf("ERROR: opto32_close() returned %d\n", ret);

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

4.6.2. opto32_init()

This function is the entry point to initializing the OPTO32 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 opto32_init(void);
```

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

Example

```
#include <stdio.h>

#include "opto32_dsl.h"

int opto32_init_dsl(void)
{
    int errs;
    int ret;

    ret = opto32_init();

    if (ret)
        printf("ERROR: opto32_init() returned %d\n", ret);

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

4.6.3. opto32_ioctl()

This function is the entry point to performing setup and control operations on a OPTO32 board. 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 18).

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

Prototype

```
int opto32_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 16).
`request`	This specifies the desired operation to be performed (section 4.7, page 18).
`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 "opto32_dsl.h"

int opto32_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = opto32_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: opto32_ioctl() returned %d\n", ret);

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

4.6.4. opto32_open()

This function is the entry point to open a connection to an OPTO32. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int opto32_open(int device, int share, int* fd);
```

Argument	Description
`device`	This is the zero-based index of the OPTO32 device 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 "opto32_dsl.h"

int opto32_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = opto32_open(device, share, fd);

    if (ret)
        printf("ERROR: opto32_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. opto32_read()

This function is the entry point to reading data from an open performed on device index `-1`. This function should only be called after a successful open. The function reads up to `bytes` bytes. The return value is the number of bytes actually read.

> **NOTE:** The read service has no functionality for reading from OPTO32 devices. Attempts to read from OPTO32 devices will return an error.

Prototype

```
int opto32_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 16).
`dst`	This is the destination for the data read from the device.
`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 "opto32_dsl.h"

int opto32_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = opto32_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: opto32_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 OPTO32 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `opto32_ioctl()` function arguments.

4.7.1. OPTO32_IOCTL_CLOCK_DIVIDER

This service sets the debounce clock divider value. The counter that is divided is the 20MHz on-board oscillator. The minimum achievable period is 200ns.

NOTE: The driver disables device interrupts while a clock divider value is being applied.

NOTE: COS State changes should be ignored when the clock divider value is changed.

Usage

Argument	Description
request	OPTO32_IOCTL_CLOCK_DIVIDER
arg	s32*

Valid argument values are from zero to 0xFFFFFF, and -1. A value of -1 will return the current divider value.

4.7.2. OPTO32_IOCTL_COS_POLARITY

This service sets the polarity of the state changes used for interrupt generation. If a bit is set, then low-to-high changes are detected. If a bit is clear, then high-to-low transitions are detected. Bit D0 corresponds to COS input zero.

NOTE: The COS Polarity configuration of input D23 does not affect counted event detection, which are always low-to-high transitions.

Usage

Argument	Description
request	OPTO32_IOCTL_COS_POLARITY
arg	s32*

Valid argument values are any value from zero to 0xFFFFFF, or -1 to retrieve the current setting.

4.7.3. OPTO32_IOCTL_COS_STATE

This service retrieves and clears the Change of State status as indicated in the Change of State Register. Writing a one to any bit clears the corresponding COS status. Writing a zero has no affect.

NOTE: If a COS input is configured to generate an interrupt its state is cleared automatically by the driver when the COS status is asserted.

Usage

Argument	Description
request	OPTO32_IOCTL_COS_STATE
arg	s32*

Valid argument values are any value from zero to 0xFFFFFF, or -1 to retrieve the current setting.

4.7.4. OPTO32_IOCTL_DEBOUNCE_MS

This service sets the debounce clock divider with millisecond resolution.

NOTE: The driver disables device interrupts while a clock divider value is being applied.

Usage

Argument	Description
request	OPTO32_IOCTL_DEBOUNCE_MS
arg	s32*

Valid argument values are from zero to 2000, in milliseconds, or -1. Using the value -1 will return the current setting. The value corresponding to 0ms is actually 200ns. When the current setting is retrieved the clock divider value is rounded to the closest millisecond interval.

4.7.5. OPTO32_IOCTL_DEBOUNCE_US

This service sets the debounce clock divider with microsecond resolution.

NOTE: The driver disables device interrupts while a clock divider value is being applied.

Usage

Argument	Description
request	OPTO32_IOCTL_DEBOUNCE_US
arg	s32*

Valid argument values are from zero to 2000000, in microseconds, or -1. Using the value -1 will return the current setting. The value corresponding to 0us is actually 200ns. When the current setting is retrieved the clock divider value is rounded to the closest microsecond interval.

4.7.6. OPTO32_IOCTL_INITIALIZE

This service returns the board to its initialized state. The initialize operation sets all hardware and software settings to their defaults.

Usage

Argument	Description
request	OPTO32_IOCTL_INITIALIZE
arg	Not used.

4.7.7. OPTO32_IOCTL_IRQ_ENABLE

This service enables and disables firmware interrupts. All specified interrupts will remain enabled until disabled by the application.

Usage

Argument	Description
request	OPTO32_IOCTL_IRQ_ENABLE
arg	s32*

Valid argument values include any combination of the below specified options, or -1.

Value	Description
-1	Retrieve the current setting.
OPTO32_IRQ_COS_00	This refers to the Change of State input number 0 interrupt.
OPTO32_IRQ_COS_01	This refers to the Change of State input number 1 interrupt.

OPTO32_IRQ_COS_02	This refers to the Change of State input number 2 interrupt.
OPTO32_IRQ_COS_03	This refers to the Change of State input number 3 interrupt.
OPTO32_IRQ_COS_04	This refers to the Change of State input number 4 interrupt.
OPTO32_IRQ_COS_05	This refers to the Change of State input number 5 interrupt.
OPTO32_IRQ_COS_06	This refers to the Change of State input number 6 interrupt.
OPTO32_IRQ_COS_07	This refers to the Change of State input number 7 interrupt.
OPTO32_IRQ_COS_08	This refers to the Change of State input number 8 interrupt.
OPTO32_IRQ_COS_09	This refers to the Change of State input number 9 interrupt.
OPTO32_IRQ_COS_10	This refers to the Change of State input number 10 interrupt.
OPTO32_IRQ_COS_11	This refers to the Change of State input number 11 interrupt.
OPTO32_IRQ_COS_12	This refers to the Change of State input number 12 interrupt.
OPTO32_IRQ_COS_13	This refers to the Change of State input number 13 interrupt.
OPTO32_IRQ_COS_14	This refers to the Change of State input number 14 interrupt.
OPTO32_IRQ_COS_15	This refers to the Change of State input number 15 interrupt.
OPTO32_IRQ_COS_16	This refers to the Change of State input number 16 interrupt.
OPTO32_IRQ_COS_17	This refers to the Change of State input number 17 interrupt.
OPTO32_IRQ_COS_18	This refers to the Change of State input number 18 interrupt.
OPTO32_IRQ_COS_19	This refers to the Change of State input number 19 interrupt.
OPTO32_IRQ_COS_20	This refers to the Change of State input number 20 interrupt.
OPTO32_IRQ_COS_21	This refers to the Change of State input number 21 interrupt.
OPTO32_IRQ_COS_22	This refers to the Change of State input number 22 interrupt.
OPTO32_IRQ_COS_23	This refers to the Change of State input number 23 interrupt.
OPTO32_IRQ_EVENT_COUNT	This refers to the Rx Event Counter interrupt.

4.7.8. OPTO32_IOCTL_LED

This service turns the on-board LED on or off.

Usage

Argument	Description
request	OPTO32_IOCTL_LED
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
OPTO32_LED_OFF	This turns the LED off.
OPTO32_LED_ON	This turns the LED on.

4.7.9. OPTO32_IOCTL_QUERY

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

Usage

Argument	Description
request	OPTO32_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
OPTO32_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
OPTO32_QUERY_DEVICE_TYPE	This refers to the board type, which should equal GSC_DEV_TYPE_OPTO32.

4.7.10. OPTO32_IOCTL_REG_MOD

This service performs a read-modify-write of an OPTO32 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to opto32.h for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	OPTO32_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.

4.7.11. OPTO32_IOCTL_REG_READ

This service reads the value of an OPTO32 register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to opto32.h and to gsc_pci9056.h for the complete list of accessible registers.

Usage

Argument	Description
request	OPTO32_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.

4.7.12. OPTO32_IOCTL_REG_WRITE

This service writes a value to an OPTO32 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `opto32.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	OPTO32_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.

4.7.13. OPTO32_IOCTL_RX_DATA

This service reads the 24-bit optically isolated inputs.

Usage

Argument	Description
request	OPTO32_IOCTL_RX_DATA
arg	s32*

Valid argument values returned are from `0x000000` to `0xFFFFFF`.

4.7.14. OPTO32_IOCTL_RX_EVENT_COUNTER

This service sets the receive event counter starting value. (The events counted are low-to-high transitions on the D23 optically isolated input, irrespective of the D23 COS Polarity configuration.) If enabled, an interrupt will be generated when the count rolls from `0xFFFF` to `0x0000`.

Usage

Argument	Description
request	OPTO32_IOCTL_RX_EVENT_COUNTER
Arg	s32*

Valid argument values are from 0x0000 to 0xFFFF, and −1. A value of −1 will return the current setting.

4.7.15. OPTO32_IOCTL_TX_DATA

This service updates the values applied to the eight optically isolated outputs.

Usage

Argument	Description
request	OPTO32_IOCTL_TX_DATA
arg	s32*

Valid argument values are from 0x00 to 0xFF, and −1. A value of −1 will return the current setting.

4.7.16. OPTO32_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via OPTO32_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.17, page 24), 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	OPTO32_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 4.7.17.2 on page 26.
gsc	This specifies the set of OPTO32_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.17.3 on page 26.
alt	This is unused by the OPTO32 driver and should be zero.
io	This is unused by the OPTO32 driver and should be zero.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.17. OPTO32_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 a 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 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	OPTO32_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 4.7.17.1on page 25.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.17.2 on page 26.
gsc	This specifies any number of OPTO32_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.17.3 on page 26.
alt	This is unused by the OPTO32 driver and must be zero.
io	This is unused by the OPTO32 driver and must be zero.
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.

4.7.17.1. `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.

4.7.17.2. 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 OPTO32 and other General Standards products.

Fields	Description
GSC_WAIT_MAIN_GSC	This refers to any of the GSC firmware interrupts.
GSC_WAIT_MAIN_OTHER	This generally refers to an interrupt generated by another device sharing the same interrupt as the OPTO32.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the OPTO32.
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.

4.7.17.3. gsc_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 OPTO32_IOCTL_IRQ_ENABLE IOCTL service (section 4.7.7, page 20).

Value	Description
OPTO32_WAIT_COS_00	This refers to the Change of State input number 0 interrupt.
OPTO32_WAIT_COS_01	This refers to the Change of State input number 1 interrupt.
OPTO32_WAIT_COS_02	This refers to the Change of State input number 2 interrupt.
OPTO32_WAIT_COS_03	This refers to the Change of State input number 3 interrupt.
OPTO32_WAIT_COS_04	This refers to the Change of State input number 4 interrupt.
OPTO32_WAIT_COS_05	This refers to the Change of State input number 5 interrupt.
OPTO32_WAIT_COS_06	This refers to the Change of State input number 6 interrupt.
OPTO32_WAIT_COS_07	This refers to the Change of State input number 7 interrupt.
OPTO32_WAIT_COS_08	This refers to the Change of State input number 8 interrupt.
OPTO32_WAIT_COS_09	This refers to the Change of State input number 9 interrupt.
OPTO32_WAIT_COS_10	This refers to the Change of State input number 10 interrupt.
OPTO32_WAIT_COS_11	This refers to the Change of State input number 11 interrupt.
OPTO32_WAIT_COS_12	This refers to the Change of State input number 12 interrupt.
OPTO32_WAIT_COS_13	This refers to the Change of State input number 13 interrupt.
OPTO32_WAIT_COS_14	This refers to the Change of State input number 14 interrupt.
OPTO32_WAIT_COS_15	This refers to the Change of State input number 15 interrupt.
OPTO32_WAIT_COS_16	This refers to the Change of State input number 16 interrupt.
OPTO32_WAIT_COS_17	This refers to the Change of State input number 17 interrupt.
OPTO32_WAIT_COS_18	This refers to the Change of State input number 18 interrupt.
OPTO32_WAIT_COS_19	This refers to the Change of State input number 19 interrupt.
OPTO32_WAIT_COS_20	This refers to the Change of State input number 20 interrupt.
OPTO32_WAIT_COS_21	This refers to the Change of State input number 21 interrupt.
OPTO32_WAIT_COS_22	This refers to the Change of State input number 22 interrupt.
OPTO32_WAIT_COS_23	This refers to the Change of State input number 23 interrupt.
OPTO32_WAIT_EVENT_COUNT	This refers to the Rx Event Counter interrupt.

4.7.18. OPTO32_IOCTL_WAIT_STATUS

This service counts all threads blocked via the OPTO32_IOCTL_WAIT_EVENT IOCTL service (section 4.7.17, page 24), 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	OPTO32_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 4.7.17.2 on page 26.
gsc	This specifies the set of OPTO32_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.17.3 on page 26.
alt	This is unused by the OPTO32 driver and should be zero.
io	This is unused by the OPTO32 driver and should be zero.
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 driver is built into an OS specific executable. The device driver files are summarized in the following table. Some source files are specific to the OPTO32, some are specific only to the OS and some are OPTO32 and OS independent.

Description	Files	Location	OS
Source Files	`*.c`, `*.h`	…/driver/	Linux
Header File	`opto32.h`	…/driver/	Linux
Driver File	`opto32.ko` †	…/driver/	Linux (kernels version 2.6 and later)
	`opto32.o` †	…/driver/	Linux (kernels version 2.4 and earlier)
	`opto32_dil.lib` `opto32_dil.dll`	…\lib\…	Windows
	`opto32_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 OPTO32 driver user manual.

5.3. Startup

For instructions on starting the driver executable refer to this same section number in the OS specific OPTO32 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 OPTO32 driver user manual.

5.5. Version

For instructions on obtaining the driver version number refer to this same section number in the OS specific OPTO32 driver user manual.

5.6. Shutdown

For instructions on terminating the driver executable refer to this same section number in the OS specific OPTO32 driver user manual.

6. Document Source Code Examples

The source code examples included in this document are built into a static library usable with OPTO32 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 …`	`…/docsrc/`	All
Header File	`opto32_dsl.h`	`…/include/`	All
Library File	`opto32_dsl.a`	`…/lib/`	Linux
	`opto32_dsl.lib`	`…\lib\…`	Windows

6.2. Build

For library build instructions refer to this same section number in the OS specific OPTO32 driver user manual.

6.3. Library Use

For library usage information refer to this same section number in the OS specific OPTO32 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 `opto32_open()` there is the utility file `open.c` containing the utility function `opto32_open_util()`. The naming pattern is as follows: API function `opto32_xxxx()`, utility file name `xxxx.c`, utility function `opto32_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 `OPTO32_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `opto32_query()`. The naming pattern is as follows: IOCTL code `OPTO32_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `opto32_xxxx()`.

7.1. Files

The library files are summarized in the table below.

Description	Files	Location	OS
Source Files	`*.c, *,h …`	`…/utils/`	All
Header File	`opto32_utils.h`	`…/include/`	All

Library File	`opto32_utils.a`
	`gsc_utils.a`
	`os_utils.a`
	`plx_utils.a`
		`…/lib/`	Linux

	`opto32_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 OPTO32 driver user manual.

7.3. Library Use

For library usage information refer to this same section number in the OS specific OPTO32 driver user manual.

8. Operating Information

This section explains some basic operational procedures for using the OPTO32. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use. For additional operating information refer to this same section number in the OS specific *OPTO32 Driver User Manual*.

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 the device registers to the console. When used, the function is typically used to verify the device configuration. In these cases, the function should be called after complete board configuration. 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	`opto32_reg_list()`	Source File	All
Source File	`util_reg.c`	`…/utils/`	All
Header File	`opto32_utils.h`	`…/include/`	All
Library File	`opto32_utils.a`	`…/lib/`	Linux
	`opto32_utils.lib`	`…\lib\…`	Windows

9. Sample Applications

For information on the sample applications refer to this same section number in the OS specific OPTO32 driver user manual.

Document History

Revision	Description
June 20, 2023	Initial release. Version 8.7.104.x.x. Minor editorial changes.
October 7, 2022	Initial release. Version 8.6.101.x.x. Updated the information for the open and close calls.
July 5, 2022	Initial release. Version 8.5.100.x.x Minor editorial changes.
June 30, 2022	Initial release. Version 8.5.99.x.x. Removed all references to the SDK.
March 25, 2021	Initial release. Version 8.4.93.x.x. Added error code information. Added SDK discontinuation notice. Various minor editorial changes.
May 22, 2020	Initial release. Version 8.4.91.x.x. Minor editorial changes.
May 22, 2020	Initial release. Version 8.4.91.x.x. Numerous editorial changes. Added WAIT_EVENT notes.
August 9, 2019	Various editorial updates.
July 17, 2019	Initial release. Version 8.3.87.x.x. Minor editorial changes. Added a licensing subsection. Document reorganization. Divided document into an OS specific user manual and an API reference manual.
September 6, 2018	Initial release. Version 8.2.80.x.x.