HPDI32-COS

High Performance 32-bit Digital I/O

All Form Factors
…-HPDI32B-COS

API Library
Reference Manual

Manual Revision: May 2, 2024
Driver Release Version 1.0.110.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

General Standards Corporation makes no warranty of any kind with regard to this documentation and/or software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release, **General Standards Corporation** assumes no responsibility for any errors, inaccuracies or omissions herein. This documentation, information and software are made available solely on an "as-is" basis. Nor is there any commitment to update or keep current this documentation.

General Standards Corporation does not assume any liability arising out of the application or use of documentation, software, product or circuit described herein, nor is any license conveyed under any patent rights or any rights of others.

General Standards Corporation assumes no responsibility for any consequences resulting from omissions or errors in this manual or from the use of information contained herein.

General Standards Corporation reserves the right to make any changes, without notice, to this documentation, software or product, to improve accuracy, clarity, reliability, performance, function, or design.

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the HPDI32-COS 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 HPDI32-COS 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
BMDMA	Block Mode DMA
COS	Change of State
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
LA	Logic Analyzer
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
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 HPDI32-COS installation directory or any of its subdirectories.
API Library	This is a library that provides application-level access to HPDI32-COS hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the HPDI32-COS device driver, which runs in kernel space with kernel mode privileges.
HPDI32-COS	This is used as a general reference to any device supported by this driver.
Library	This is usually a general reference to the API Library.
Linux	This refers to the Linux operating system. Refer to the *HPDI32-COS Linux 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 HPDI32-COS applications. The overall architecture is illustrated in Figure 1 below.

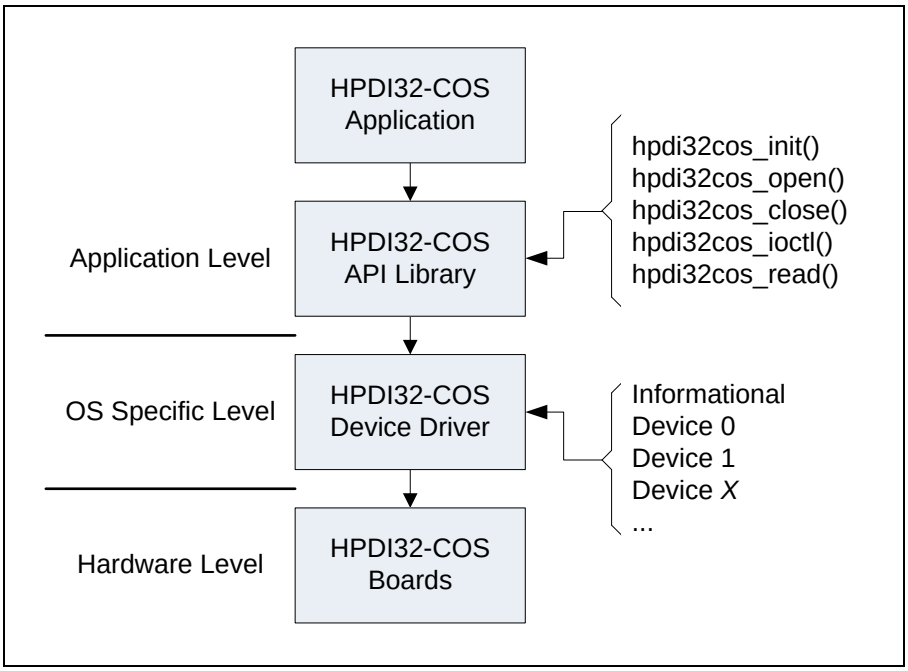

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing HPDI32-COS boards is via the HPDI32-COS 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 operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with HPDI32-COS 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 HPDI32-COS is a high-performance 32-bit parallel digital I/O interface board. The host side connection is PCI based. The board supports Change of State detection and simple Logic Analyzer input operations and can accumulate data at a rate of up to 32M 32-bit words per second at the cable interface. An onboard receive FIFO of 8k data values buffers transfer data between the PCI bus and the cable interface. This allows the HPDI32-COS to maintain maximum bursts on the cable interface (at least up to the depth of the Rx FIFO) independent of the PCI bus interface. The onboard FIFO can also be used to buffer data between the cable interface and the PCI bus to maintain sustained data throughput for real-time applications. The board accommodates a wide range of applications. The board has an advanced PCI interface engine, which provides for increased data throughput via DMA.

The 32 cable data signals can also be used as general-purpose inputs or outputs. The general-purpose outputs are configurable on a per byte basis. In addition to these 32 primary data lines, the external interface includes a set of seven general-purpose inputs. Two of these can be configured as discrete outputs reflecting on-board status.

1.6. Reference Material

The following reference material may be of particular benefit in using the HPDI32-COS, 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 *HPDI32-COS Device Driver User Manual* for your operating system from General Standards Corporation.
- The applicable *HPDI32-COS User Manual* from General Standards Corporation.
- The *PCI9080 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 additional information on driver installation refer to this same section number in the OS specific HPDI32-COS 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 HPDI32-COS 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: 1.0.110.50
32-bit support: yes
boards: 1
models: HPDI32B-COS
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/hpdi32cos`".

2.3. File List

For the list of primary files included with each release refer to this same section number in the OS specific HPDI32-COS 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 HPDI32-COS driver user manual.

Directory	Description
hpdi32cos/	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 41).
…/driver/	This directory contains the driver and any related files (section 5, page 40).
…/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 45).
…/utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 42).

2.5. Installation

For installation instructions refer to this same section number in the OS specific HPDI32-COS driver user manual.

2.6. Removal

For removal instructions refer to this same section number in the OS specific HPDI32-COS driver user manual.

2.7. Overall Make Script

Each HPDI32-COS 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 HPDI32-COS driver user manual.

2.8. Environment Variables

For environment variable information refer to this same section number in the OS specific HPDI32-COS driver user manual.

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing HPDI32-COS 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 HPDI32-COS 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 HPDI32-COS specific header files. Therefore, sources may include only this one HPDI32-COS header and make files may reference only this one HPDI32-COS include directory.

Description	File	Location	OS
Header File	`hpdi32cos_main.h`	`…/include/`	Linux

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the HPDI32-COS driver installation. For ease of use it is suggested that applications link one of the static library files shown below rather than individually linking those libraries identified separately later in this document. Linking this one library file pulls in all other pertinent HPDI32-COS specific static libraries. Therefore, make files may reference only a single HPDI32-COS static library and only this one HPDI32-COS library directory.

Description	File	Location	OS
Library File	`hpdi32cos_main.a` `hpdi32cos_multi.a`	`…/lib/`	Linux

> **NOTE**: For applications using the HPDI32-COS and no other GSC devices, link the `hpdi32cos_main.a` library. For applications using multiple GSC device types, link the `xxxx_main.xxx` library for one of the devices and the `xxxx_multi.xxx` library for the others. Linking multiple `xxxx_main.xxx` 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.xxx` 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 HPDI32-COS API Library is implemented as a shared or dynamically loaded library and is thus not linked with the HPDI32-COS Main Library. The API Library must be linked with applications by an OS specific means. For additional information refer to this same section number in the OS specific HPDI32-COS driver user manual.

3.2.1. Build

For information on building the Main Library refer to this same section number in the OS specific HPDI32-COS 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 HPDI32-COS driver user manual.

4. API Library

The HPDI32-COS API Library is the software interface between user applications and the HPDI32-COS device driver. The interface is accessed by including the header file `hpdi32cos_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 HPDI32-COS applications. The pertinent files are identified in the following table. Some source files are specific only to the HPDI32-COS, some are specific only to the OS and some are HPDI32-COS and OS independent.

Description	Files	Location	OS
Source Files	`*.c, *.h`	`…/api/`	Linux
Header File	`hpdi32cos_api.h`	`…/include/`	Linux
Library File	`libhpdi32cos_api.so` †	`…/lib/` `/usr/lib/`	Linux

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

4.2. Build

For build instructions refer to this same section number in the OS specific HPDI32-COS driver user manual.

4.3. Library Use

For Library usage information refer to this same section number in the OS specific HPDI32-COS driver user manual.

4.4. Macros

The Library interface includes the following macros, which are defined in `hpdi32cos.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 HPDI32-COS registers.

4.4.2.1. GSC Registers

The following table give the complete set of GSC specific HPDI32-COS 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, please refer to the appropriate *HPDI32-COS User Manual*.

> **NOTE**: Refer to the output of the "id" sample application (see section 9 of the OS specific HPDI32-COS driver user manual) for a complete list of the registers supported by the device being accessed.

Macros	Description
HPDI32COS_GSC_BCR	Board Control Register (BCR)
HPDI32COS_GSC_BSR	Board Status Register (BSR)
HPDI32COS_GSC_DMR	Data Mask Register (DMR)
HPDI32COS_GSC_ECIVR	Event Counter Initial Value Register (ECIVR)
HPDI32COS_GSC_ECR	Event Counter Register (ECR)
HPDI32COS_GSC_FRR	Firmware Revision Register (FRR)
HPDI32COS_GSC_ICR	Interrupt Control Register (ICR)
HPDI32COS_GSC_IELR	Interrupt Edge/Level Register (IELR)
HPDI32COS_GSC_IHLR	Interrupt High/Low Register (IHLR)
HPDI32COS_GSC_ISR	Interrupt Status Register (ISR)
HPDI32COS_GSC_LATR	Logic Analyzer Trigger Register (LATR)
HPDI32COS_GSC_RAR	Rx Almost Register (RAR)
HPDI32COS_GSC_RDFR	Rx Data FIFO Register (RDFR)
HPDI32COS_GSC_RDIR	Rx Data Input Register (RDIR)
HPDI32COS_GSC_RFSR	Rx FIFO Size Register (RFSR)
HPDI32COS_GSC_RFWCR	Rx FIFO Word Count Register (RFWCR)
HPDI32COS_GSC_RWCR	Rx Word Count Register (RWCR)
HPDI32COS_GSC_SCDR	Sample Clock Divider Register (SCDR)
HPDI32COS_GSC_TDOR	Tx Data Output Register (TDOR)

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_pci9080.h`, which is automatically included via `hpdi32cos_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_pci9080.h`, which are automatically included via `hpdi32cos_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).

4.6. Functions

The Library interface includes the functions given in the following subsections. The function 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 from the device. 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

4.6.1. hpdi32cos_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 hpdi32cos_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 descriptions above.

Example

```
#include <stdio.h>

#include "hpdi32cos_dsl.h"

int hpdi32cos_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = hpdi32cos_close(fd);

    if (ret)
        printf("ERROR: hpdi32cos_close() returned %d\n", ret);

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

4.6.2. hpdi32cos_init()

This function is the entry point to initializing the HPDI32-COS 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 hpdi32cos_init(void);
```

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

Example

```
#include <stdio.h>

#include "hpdi32cos_dsl.h"

int hpdi32cos_init_dsl(void)
{
```

```
    int errs;
    int ret;

    ret = hpdi32cos_init();

    if (ret)
        printf("ERROR: hpdi32cos_init() returned %d\n", ret);

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

4.6.3. hpdi32cos_ioctl()

This function is the entry point to performing setup and control operations on an HPDI32-COS board. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 (page 19).

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

Prototype

```
int hpdi32cos_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 "hpdi32cos_dsl.h"

int hpdi32cos_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = hpdi32cos_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: hpdi32cos_ioctl() returned %d\n", ret);

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

4.6.4. hpdi32cos_open()

This function is the entry point to open a connection to an HPDI32-COS board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int hpdi32cos_open(int device, int share, int* fd);
```

Argument	Description
device	This is the zero-based index of the HPDI32-COS 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 descriptions above.

Example

```
#include <stdio.h>

#include "hpdi32cos_dsl.h"

int hpdi32cos_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = hpdi32cos_open(device, share, fd);

    if (ret)
        printf("ERROR: hpdi32cos_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. hpdi32cos_read()

This function is the entry point to reading data from an open connection. The function reads up to `bytes` bytes from the device.

> **NOTE:** If an open was performed using an index of `-1`, then read requests will acquire information from the driver (section 2.2, page 10) rather than data from a device.
>
> **NOTE:** For additional information refer to the Data Transfer Modes section (section 8.1.3, page 43).
>
> **NOTE:** The check for an overflow or an underflow is performed upon entry to the read service. The read service does not check for these conditions that occur while the read is in progress. For in-progress overflows or underflows an application must perform the check manually or wait for the check performed by a subsequent read request.

Prototype

```
int hpdi32cos_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. When reading from a device, this must be a multiple of four bytes.

Return Value	Description
`0` to `bytes`	The operation succeeded. When reading from a device, a value less than `bytes` indicates that the I/O timeout period (section 4.7.38, page 32) lapsed before the entire request could be satisfied.
`< 0`	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "hpdi32cos_dsl.h"

int hpdi32cos_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = hpdi32cos_read(fd, dst, bytes);
```

```
    if (ret < 0)
        printf("ERROR: hpdi32cos_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 HPDI32-COS API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `hpdi32cos_ioctl()` function arguments.

4.7.1. HPDI32COS_IOCTL_BYTE_ENABLE

This service configures the use of the 32 digital cable data signals as GPIO.

> **NOTE**: The Tx Enable service (section 4.7.45, page 35) configures all fours bytes to outputs, but it does not affect the Byte Enable bits.

Usage

Argument	Description
request	HPDI32COS_IOCTL_BYTE_ENABLE
arg	s32*

Valid argument values are as follows. These options can be OR'd together to select multiple bytes at a time. Any byte referenced is enabled as an output. Any byte not referenced is an input only.

Value	Description
-1	Retrieve the current setting.
HPDI32COS_BYTE_ENABLE_ALL	This enables all four bytes as outputs.
HPDI32COS_BYTE_ENABLE_D7_D0	This enables bits D0 through D7 as outputs.
HPDI32COS_BYTE_ENABLE_D15_D8	This enables bits D0 through D7 as outputs.
HPDI32COS_BYTE_ENABLE_D23_D16	This enables bits D0 through D7 as outputs.
HPDI32COS_BYTE_ENABLE_D31_D24	This enables bits D0 through D7 as outputs.
HPDI32COS_BYTE_ENABLE_NONE	This sets all 32 bits as inputs only.

4.7.2. HPDI32COS_IOCTL_CC_D0_D6_STATE

This service retrieves the current signal state for Cable Command signals D0 through D6, irrespective of their current configuration.

Usage

Argument	Description
request	HPDI32COS_IOCTL_CC_D0_D6_STATE
arg	s32*

Valid values returned are from 0x0 through 0x7F. Cable Command signal D0 is represented by value bit D0, and so on.

4.7.3. HPDI32COS_IOCTL_CC_D5_D6

This service configures the use of Cable Command signals D5 and D6 as GPIO.

Usage

Argument	Description
request	HPDI32COS_IOCTL_CC_D5_D6
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HPDI32COS_CC_D5_D6_INPUT	This sets both signals as inputs only.
HPDI32COS_CC_D5_D6_OUTPUT	This sets both signals as outputs.

4.7.4. HPDI32COS_IOCTL_CLOCK_DIVIDER

This service configures the firmware clock divider to reduce the effective COS and LA sampling frequency. The value specified is the number of inactive clock cycles inserted between active clocking cycles.

Usage

Argument	Description
request	HPDI32COS_IOCTL_CLOCK_DIVIDER
arg	s32*

Valid argument values are in the range from zero through 0xFFFF, or -1 to retrieve the current setting.

4.7.5. HPDI32COS_IOCTL_COS_DETECTED

This service operates on the COS Detected status and always returns the current status.

> **NOTE**: In early firmware versions (0x02 and prior) the COS Detected status is cleared with any write to the Interrupt Status Register. Thus, if any interrupt is enabled and triggered, then the status is cleared when the interrupt is serviced.

Usage

Argument	Description
request	HPDI32COS_IOCTL_COS_DETECTED
arg	s32*

The following are the valid options that can be passed to the service.

Value	Description
HPDI32COS_COS_DETECTED_CHECK	This option requests the current status.
HPDI32COS_COS_DETECTED_CLEAR	This option clears the status.

Argument values returned are one of the following.

Value	Description
HPDI32COS_COS_DETECTED_NO	A COS event has not been detected.
HPDI32COS_COS_DETECTED_YES	A COS event has been detected.

4.7.6. HPDI32COS_IOCTL_COUNTER_ZERO

This service retrieves an indication of the Event Counter equaling zero.

Usage

Argument	Description
request	HPDI32COS_IOCTL_COUNTER_ZERO
arg	s32*

Valid argument values are as follows.

Value	Description
HPDI32COS_COUNTER_ZERO_NO	The Event Counter is not zero.
HPDI32COS_COUNTER_ZERO_YES	The Event Counter is zero.

4.7.7. HPDI32COS_IOCTL_EVENT_COUNT

This service retrieves the current event counter value.

Usage

Argument	Description
request	HPDI32COS_IOCTL_EVENT_COUNT
arg	u32*

Valid argument values are in the range from zero through 0xFFFFFFFF.

4.7.8. HPDI32COS_IOCTL_EVNT_CNT_INI_GET

This service retrieves the Event Counter Initial value, which is copied to the Event Counter each time the receiver is started.

Usage

Argument	Description
request	HPDI32COS_IOCTL_EVNT_CNT_INI_GET
arg	u32*

Argument values returned are in the range from zero through 0xFFFFFFFF.

4.7.9. HPDI32COS_IOCTL_EVNT_CNT_INI_SET

This service updated the Event Counter Initial value, which is copied to the Event Counter each time the receiver is started.

Usage

Argument	Description
request	HPDI32COS_IOCTL_EVNT_CNT_INI_SET

arg	u32*

Valid argument values are in the range from zero through 0xFFFFFFFF.

4.7.10. HPDI32COS_IOCTL_FIFO_WORD_COUNT

This service retrieves the Event Counter Initial value, which is copied to the Event Counter each time the receiver is started.

Usage

Argument	Description
request	HPDI32COS_IOCTL_FIFO_WORD_COUNT
arg	u32*

Argument values returned are in the range from zero up through 0xFFFFFFFF.

4.7.11. HPDI32COS_IOCTL_INITIALIZE

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

Usage

Argument	Description
request	HPDI32COS_IOCTL_INITIALIZE
arg	Not used.

4.7.12. HPDI32COS_IOCTL_IRQ_CONFIG_EDGE

This service configures firmware interrupts to be either edge triggered or level triggered. If a bit is set, then the interrupt is edge triggered. If a bit is clear, then the interrupt is level triggered.

Usage

Argument	Description
request	HPDI32COS_IOCTL_IRQ_CONFIG_EDGE
arg	s32*

Valid argument values include any bitwise combination of the bits defined for the HPDI32COS_IOCTL_IRQ_ENABLE service (section 4.7.14, page 23), or -1 to retrieve the current configurations.

4.7.13. HPDI32COS_IOCTL_IRQ_CONFIG_HIGH

This service configures firmware interrupts to be either high or low triggered. High refers to either a high level or a rising edge, depending on the interrupt's edge/level configuration. Low refers to either a low level or a falling edge, depending on the interrupt's edge/level configuration. If a bit is set, then the interrupt is high triggered. If a bit is clear, then the interrupt is low triggered.

Usage

Argument	Description
request	HPDI32COS_IOCTL_IRQ_CONFIG_HIGH

arg	s32*

Valid argument values include any bitwise combination of the bits defined for the HPDI32COS_IOCTL_IRQ_ENABLE service (section 4.7.14, page 23), or -1 to retrieve the current configurations.

4.7.14. HPDI32COS_IOCTL_IRQ_ENABLE

This service enables a specified set of firmware interrupts. If a bit is set, then the interrupt is enabled. If a bit is clear, then the interrupt is disabled. When an interrupt is generated, it is serviced and disabled by the driver.

Usage

Argument	Description
request	HPDI32COS_IOCTL_IRQ_ENABLE
arg	s32*

Valid argument values include any bitwise combination of the following bits, or -1 to retrieve the current configuration.

Value	Description
HPDI32COS_IRQ_ALL	This refers to all interrupt options.
HPDI32COS_IRQ_CC1_FALL	This refers to the Cable Command 1 signal, an input only signal, being negated.
HPDI32COS_IRQ_CC1_RISE	This refers to the Cable Command 1 signal, an input only signal, being asserted.
HPDI32COS_IRQ_CC2	This refers to the Cable Command 2 signal, an input only signal, being asserted or negated, which depends on how the interrupt is configured.
HPDI32COS_IRQ_CC3	This refers to the Cable Command 3 signal, an input only signal, being asserted or negated, which depends on how the interrupt is configured.
HPDI32COS_IRQ_CC4	This refers to the Cable Command 4 signal, an input only signal, being asserted or negated, which depends on how the interrupt is configured.
HPDI32COS_IRQ_CC5	This refers to the Cable Command 5 signal being asserted or negated, which depends on how the interrupt is configured. This signal may be an input or an output, depending on the HPDI32COS_IOCTL_CC_D5_D6 setting (section 4.7.3, page 20).
HPDI32COS_IRQ_CC6	This refers to the Cable Command 6 signal being asserted or negated, which depends on how the interrupt is configured. This signal may be an input or an output, depending on the HPDI32COS_IOCTL_CC_D5_D6 setting (section 4.7.3, page 20).
HPDI32COS_IRQ_COS_DETECTED	This refers to the detection of a COS event.
HPDI32COS_IRQ_EVENT_COUNT_ZERO	This refers to the Event Counter Zero status. This status is asserted when the count becomes zero or equals zero, depending on the interrupt configuration.
HPDI32COS_IRQ_LA_TRIGGERED	This refers to the Logic Analyzer being triggered.
HPDI32COS_IRQ_RX_FIFO_AE	This refers to the Rx FIFO's Almost Empty status.
HPDI32COS_IRQ_RX_FIFO_AF	This refers to the Rx FIFO's Almost Full status.
HPDI32COS_IRQ_RX_FIFO_EMPTY	This refers to the Rx FIFO's empty status.
HPDI32COS_IRQ_RX_FIFO_FULL	This refers to the Rx FIFO's full status.

HPDI32COS_IRQ_RX_FIFO_OVER	This refers to the Tx FIFO Overflow status.
HPDI32COS_IRQ_RX_FIFO_UNDER	This refers to the Tx FIFO Underflow status.
HPDI32COS_IRQ_RX_RUNNING	This refers to the receiver running status. The status is asserted when the receiver is started and negated when the receiver is stopped.
HPDI32COS_IRQ_RX_STOPPED	This refers to the Rx Stopped status.

4.7.15. HPDI32COS_IOCTL_LA_TRIG_WORD_GET

This service retrieves the current Logic Analyzer Trigger Word.

Usage

Argument	Description
request	HPDI32COS_IOCTL_LA_TRIG_WORD_GET
arg	u32*

Argument values returned are in the range from zero through 0xFFFFFFFF.

4.7.16. HPDI32COS_IOCTL_LA_TRIG_WORD_SET

This service updates the current Logic Analyzer Trigger Word.

Usage

Argument	Description
request	HPDI32COS_IOCTL_LA_TRIG_WORD_SET
arg	u32*

Valid argument values are in the range from zero through 0xFFFFFFFF.

4.7.17. HPDI32COS_IOCTL_LA_TRIGGERED

This service reports whether or not the Logic Analyzer has been triggered.

> **NOTE**: The LA Triggered status is cleared when Rx Start (section 4.7.40, page 33) is set to its *NO* option.

Usage

Argument	Description
request	HPDI32COS_IOCTL_LA_TRIGGERED
arg	s32*

The following are the valid options that can be passed to the service.

Value	Description
HPDI32COS_LA_TRIGGERED_CHECK	This option requests the current status.

Argument values returned are as follows.

Value	Description
HPDI32COS_LA_TRIGGERED_NO	The Logic Analyzer has not been triggered.
HPDI32COS_LA_TRIGGERED_YES	The Logic Analyzer has been triggered.

4.7.18. HPDI32COS_IOCTL_LOOPBACK

This service enables or disables the Loopback feature. When enabled, it allows simultaneous Rx Start (section 4.7.40, page 33) and Tx Enable (section 4.7.45, page 35) operation.

> **NOTE**: If Loopback is disabled, then Tx Enable (section 4.7.45, page 35) inhibits Rx Start (section 4.7.40, page 33). It doesn't affect the Rx Start setting, but it does prevent Rx Start from taking place.

Usage

Argument	Description
request	HPDI32COS_IOCTL_LOOPBACK
arg	s32*

Valid argument values include the following options.

Value	Description
HPDI32COS_LOOPBACK_DISABLE	This refers to the loopback feature being disabled.
HPDI32COS_LOOPBACK_ENABLE	This refers to the loopback feature being enabled.

4.7.19. HPDI32COS_IOCTL_MODE

This service selects the firmware's basic operating mode as either Change of State detection of Logic Analyzer.

Usage

Argument	Description
request	HPDI32COS_IOCTL_MODE
arg	s32*

Valid argument values include the following options.

Value	Description
-1	Retrieve the current setting.
HPDI32COS_MODE_COS	This refers to Change of State mode.
HPDI32COS_MODE_LA	This refers to Logic Analyzer mode.

4.7.20. HPDI32COS_IOCTL_QUERY

This service queries the driver for various pieces of information about the device and the driver. The query option is passed to the service and the associated information is returned.

Usage

Argument	Description
request	HPDI32COS_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
HPDI32COS_QUERY_CLOCK_DIV_MAX	This refers to the maximum Clock Divider value for the HPDI32COS_IOCTL_CLOCK_DIVIDER service (section

	4.7.4, page 20).
HPDI32COS_QUERY_COUNT	This refers to the number of query options supported. The range of options supported is from zero to count – 1.
HPDI32COS_QUERY_DEVICE_TYPE	This refers to the basic device type, which should be GSC_DEV_TYPE_HPDI32COS.
HPDI32COS_QUERY_FIFO_SIZE_RX	This refers to the size of the board's Rx FIFO.
HPDI32COS_QUERY_FORM_FACTOR	This refers to the board's formfactor. See the hpdi32cos_form_factor_t enumeration below.
HPDI32COS_QUERY_FREF_DEFAULT	This refers to board's default reference clock frequency.
HPDI32COS_QUERY_FREF_MAX	This refers to board's maximum supported reference clock frequency. The reference oscillator is user replaceable, but should not be replaced with an oscillator exceeding this frequency.
HPDI32COS_QUERY_JUMPER_ON	This reports the bit value returned when a user jumper is installed.
HPDI32COS_QUERY_JUMPER_QTY	This refers to the number of supported user jumpers.
HPDI32COS_QUERY_JUMPER_VAL	This refers to the value read from the user jumpers.
HPDI32COS_QUERY_MODEL	This refers to the board's basic model number. See the hpdi32cos_model_t enumeration below.

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

Value	Description
HPDI32COS_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 per the hpdi32cos_form_factor_t enumeration.

Value	Description
HPDI32COS_FORM_FACTOR_UNKNOWN	The form factor is unknown.
HPDI32COS_FORM_FACTOR_PCI	The form factor is PCI.
HPDI32COS_FORM_FACTOR_PMC	The form factor is PMC.

Valid return values for the Model query are as follows per the hpdi32cos_model_t enumeration.

Value	Description
HPDI32COS_MODEL_HPDI32A	The device is an HPDI32A-COS model board.
HPDI32COS_MODEL_HPDI32B	The device is an HPDI32B-COS model board.

4.7.21. HPDI32COS_IOCTL_REG_MOD

This service performs a read-modify-write of an HPDI32-COS register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to hpdi32cos.h for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	HPDI32COS_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.

4.7.22. HPDI32COS_IOCTL_REG_READ

This service reads the value of an HPDI32-COS register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `hpdi32cos.h` and `gsc_pci9080.h` for the complete list of accessible registers.

Usage

Argument	Description
request	HPDI32COS_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.

4.7.23. HPDI32COS_IOCTL_REG_WRITE

This service writes a value to an HPDI32-COS register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `hpdi32cos.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	HPDI32COS_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.

4.7.24. HPDI32COS_IOCTL_RX_DATA

This service retrieves the current state driven on the 32 cable data signals.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_DATA
arg	u32*

The value returned are in the range from zero through 0xFFFFFFFF.

4.7.25. HPDI32COS_IOCTL_RX_DATA_MASK_GET

This service retrieves the current data mask applied to the 32 cable data signal states before COS or LA processing.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_DATA_MASK_GET
arg	u32*

The value returned are in the range from zero through 0xFFFFFFFF.

4.7.26. HPDI32COS_IOCTL_RX_DATA_MASK_SET

This service updates the current data mask applied to the 32 cable data signal states before COS or LA processing. If a bit is set, then that bit is used for COS or LA processing. If a bit clear, then that bit is ignored.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_DATA_MASK_SET
arg	u32*

The value returned are in the range from zero through 0xFFFFFFFF.

4.7.27. HPDI32COS_IOCTL_RX_FIFO_AE

This service updates the Rx FIFO Almost Empty threshold level. The Rx FIFO content is discarded during this service when a setting is being applied.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_FIFO_AE
arg	s32*

Valid argument values are in the range from zero through 0xFFFF, or -1 to retrieve the current setting.

4.7.28. HPDI32COS_IOCTL_RX_FIFO_AF

This service updates the Rx FIFO Almost Full threshold level. The Rx FIFO content is discarded during this service when a setting is being applied.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_FIFO_AF
arg	s32*

Valid argument values are in the range from zero through the 0xFFFF, or -1 to retrieve the current setting.

> **NOTE**: If the threshold is set to a value larger than the Rx FIFO, then the Rx FIFO will never become Almost Full. Consequently, the Rx FIFO Almost Full status cannot stop the receiver. The result is that Rx FIFO may likely overflow.

4.7.29. HPDI32COS_IOCTL_RX_FIFO_OVER

This service reports the current Rx FIFO overflow status.

> **NOTE:** An overflow occurs when data is clocked into the Rx FIFO while the FIFO is already full. For the HPDI32-COS, this can only occur when the Rx FIFO Almost Full threshold level is set to a value larger than the size of Rx FIFO.

> **NOTE:** Rx FIFO overflows are cleared by resetting the Rx FIFO (section 4.7.30, page 30).

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_FIFO_OVER
arg	s32*

Valid options that can be passed to the service are as follows.

Value	Description
HPDI32COS_FIFO_ERROR_CHECK	This option requests the current error status.
HPDI32COS_FIFO_ERROR_CLEAR	This option clears the error status.

The current state is reported as one of the following values.

Value	Description
HPDI32COS_FIFO_ERROR_NO	The FIFO has not experienced an overflow condition.
HPDI32COS_FIFO_ERROR_YES	The FIFO has experienced an overflow condition.

4.7.30. HPDI32COS_IOCTL_RX_FIFO_RESET

This service resets the Rx FIFO, which clears the content. It also clears the Rx FIFO overflow and underflow status bits.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_FIFO_RESET
arg	Not used.

4.7.31. HPDI32COS_IOCTL_RX_FIFO_STATUS

This service retrieves the current Rx FIFO fill level status.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_FIFO_STATUS
arg	s32*

The current status is reported as one of the following values.

Value	Description
HPDI32COS_FIFO_STATUS_AE	The FIFO contains *Almost Empty* values or fewer.
HPDI32COS_FIFO_STATUS_AF	The FIFO has room to accept *Almost Full* or fewer additional values before becoming full.
HPDI32COS_FIFO_STATUS_EMPTY	The FIFO is empty.
HPDI32COS_FIFO_STATUS_FULL	The FIFO is full.
HPDI32COS_FIFO_STATUS_MEDIUM	The FIFO's fill level is between the *Almost Empty* mark and the *Almost Full* mark.

4.7.32. HPDI32COS_IOCTL_RX_FIFO_UNDER

This service reports the current Rx FIFO underflow status.

> **NOTE:** An Rx FIFO underflow occurs when the FIFO is read while empty. This can typically only occur when an application reads from the FIFO directly.

> **NOTE:** Rx FIFO underflows are cleared by resetting the Rx FIFO (section 4.7.30, page 30).

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_FIFO_UNDER
arg	s32*

Valid options that can be passed to the service are as follows.

Value	Description
HPDI32COS_FIFO_ERROR_CHECK	This option requests the current error status.
HPDI32COS_FIFO_ERROR_CLEAR	This option clears the error status.

The current state is reported as one of the following values.

Value	Description
HPDI32COS_FIFO_ERROR_NO	The FIFO has not experienced an underflow condition.
HPDI32COS_FIFO_ERROR_YES	The FIFO has experienced an underflow condition.

4.7.33. HPDI32COS_IOCTL_RX_IO_ABORT

This service aborts an ongoing `hpdi32cos_read()` request.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
HPDI32COS_IO_ABORT_NO	An `hpdi32cos_read()` request was not aborted as none were ongoing.
HPDI32COS_IO_ABORT_YES	An ongoing `hpdi32cos_read()` request was aborted.

4.7.34. HPDI32COS_IOCTL_RX_IO_BMDMA_TRSH

This service sets the minimum DMA transfer size used during Block Mode DMA based read requests. As such read requests may consist of multiple smaller DMA transfers, this setting limits the smallest size of those individual transfers. This setting does not apply to the last DMA transfer of the read request. The unit of measure for this setting is bytes.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_BMDMA_TRSH
arg	s32*

Valid argument values are from zero to the depth of the Rx FIFO in 32-bit words, or `-1` to retrieve the current setting. The default is `60` bytes.

4.7.35. HPDI32COS_IOCTL_RX_IO_MODE

This service selects the mechanism used to retrieve data from the Rx FIFO during read requests.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_MODE
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current setting.
GSC_IO_MODE_BMDMA	Data is retrieved using Block Mode DMA.
GSC_IO_MODE_DMDMA	Data is retrieved using Demand Mode DMA.
GSC_IO_MODE_PIO	Data is retrieved using repetitive register accesses.

4.7.36. HPDI32COS_IOCTL_RX_IO_OVER

This service configures the read service to check for an Rx FIFO overflow before performing read operations. Data is lost when there is an overflow. If the check is performed and an overflow is detected, then the read service immediately returns an error.

> **NOTE:** The check for an overflow is performed upon entry to the read service. The read service does not check for overflows that occur while the read is in progress. For in-progress overflows an application must perform the check manually or wait for the check performed by a subsequent read request.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_OVER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HPDI32COS_IO_ERROR_CHECK	Perform the check. This is the default.
HPDI32COS_IO_ERROR_IGNORE	Do not perform the check.

4.7.37. HPDI32COS_IOCTL_RX_IO_PIO_TRSH

This service sets the threshold at which DMA read requests instead resort to PIO mode. When the number of data values in a read request is less than or equal to this value, then the operation automatically uses PIO instead of DMA. This is intended to improve efficiency as small read requests can be performed more efficiently when done using PIO rather than DMA.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_PIO_TRSH
arg	s32*

Valid argument values are any non-negative number, or -1 to retrieve the current setting. The default is 15 values.

4.7.38. HPDI32COS_IOCTL_RX_IO_TIMEOUT

This service sets the timeout limit for read requests. The value is expressed in seconds. The timeout limit is the total amount of time allowed for a single hpdi32cos_read() request. When this time limit has expired the service terminates. When this occurs the hpdi32cos_read() return value will be less than the number of bytes requested, and possibly zero.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and HPDI32COS_IO_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more data, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option HPDI32COS_IO_TIMEOUT_INFINITE is used, then the driver waits indefinitely rather than timing out. The default is 10 seconds.

4.7.39. HPDI32COS_IOCTL_RX_IO_UNDER

This service operates on the Rx FIFO underflow status.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_IO_UNDER
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current setting.
HPDI32COS_IO_ERROR_CHECK	Check the underflow status. This is the default.
HPDI32COS_IO_ERROR_IGNORE	Ignore the current status.

4.7.40. HPDI32COS_IOCTL_RX_START

This service starts or stops the receiver.

NOTE: Setting Rx Start to *NO* (0), clears the LA Triggered (section 4.7.17, page 24) status.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_START
arg	s32*

Valid argument values include the following options.

Value	Description
-1	Retrieve the current setting.
HPDI32COS_RX_START_NO	This option halts receiver operation.
HPDI32COS_RX_START_YES	This option activates receiver operation.

4.7.41. HPDI32COS_IOCTL_RX_STATE

This service retrieves the current state of the receiver.

NOTE: If Loopback (section 4.7.18, page 25) is disabled, then Tx Enable (section 4.7.45, page 35) inhibits Rx Start. It doesn't affect the Rx Start setting, but it does prevent Rx Start from taking place.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_STATE
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
HPDI32COS_RX_STATE_DISABLED	The receiver is disabled.
HPDI32COS_RX_STATE_RUNNING	The receiver is enabled and running.
HPDI32COS_RX_STATE_STOPPED	The receiver is stopped due to the Rx FIFO becoming Almost Full.

4.7.42. HPDI32COS_IOCTL_RX_WORD_COUNT

This service retrieves the number of data values recorded into the Rx FIFO since the receiver was last started.

Usage

Argument	Description
request	HPDI32COS_IOCTL_RX_WORD_COUNT
arg	u32*

Argument values returned are in the range from zero through 0xFFFFFFFF.

4.7.43. HPDI32COS_IOCTL_TX_DATA_GET

This service retrieves the current Tx Data provided for output on the 32 cable interface data signals. Only those bytes current configured as output are driven onto the cable interface (see HPDI32COS_IOCTL_BYTE_ENABLE, section 4.7.1, page 19).

Usage

Argument	Description
request	HPDI32COS_IOCTL_TX_DATA_GET
arg	u32*

Argument values returned are in the range from zero through 0xFFFFFFFF.

4.7.44. HPDI32COS_IOCTL_TX_DATA_SET

This service updates the current Tx Data provided for output on the 32 cable interface data signals. Only those bytes current configured as output are driven onto the cable interface (see HPDI32COS_IOCTL_BYTE_ENABLE, section 4.7.1, page 19).

Usage

Argument	Description
request	HPDI32COS_IOCTL_TX_DATA_SET
arg	u32*

Valid argument values are in the range from zero through 0xFFFFFFFF.

4.7.45. HPDI32COS_IOCTL_TX_ENABLE

This service enables or disables the 32 cable interface data signals to be driven as discrete outputs. When enabled, only those bytes current configured as output are driven onto the cable interface (see `HPDI32COS_IOCTL_BYTE_ENABLE`, section 4.7.1, page 19).

> **NOTE**: If Loopback (section 4.7.18, page 25) is disabled, then Tx Enable inhibits Rx Start (section 4.7.40, page 33). It doesn't affect the Rx Start setting, but it does prevent Rx Start from taking place.

> **NOTE**: The Tx Enable service configures all fours Byte Enable (section 4.7.1, page 19) bytes as outputs, but it does not affect the Byte Enable configuration bits.

Usage

Argument	Description
`request`	`HPDI32COS_IOCTL_TX_ENABLE`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	Description
`-1`	Request current setting.
`HPDI32COS_TX_ENABLE_NO`	Disable the transmission functionality.
`HPDI32COS_TX_ENABLE_YES`	Enable the transmission functionality.

4.7.46. HPDI32COS_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via `HPDI32COS_IOCTL_WAIT_EVENT` IOCTL calls (section 4.7.47, page 36), according to the provided criteria. When a blocked thread is waiting for any event specified in the structure, then the thread is resumed.

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

Usage

Argument	Description
`request`	`HPDI32COS_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 4.7.47.2 (page 37).
gsc	This specifies the set of HPDI32COS_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.47.3 (page 37).
alt	This is unused by the HPDI32-COS driver and should be zero.
io	This specifies the set of HPDI32COS_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.47.4 (page 38).
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.47. HPDI32COS_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 and io fields. All field values must be valid and at least one event must be specified. If the thread is resumed because one of the referenced events has occurred, then the bit for the respective event is the only event bit set. All other event bits and fields are zero. (Multiple event bits are 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	HPDI32COS_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 4.7.47.1 (page 37).
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.47.2 (page 37).
gsc	This specifies any number of HPDI32COS_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.47.3 (page 37).
alt	This is unused by the HPDI32-COS driver and must be zero.
io	This specifies any number of HPDI32COS_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.47.4 (page 38).

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 is the approximate amount of time actually waited.
count	This is unused by wait event operations and must be zero.

4.7.47.1. gsc_wait_t.flags Options

Upon return from a wait request the wait structure's flags field indicates 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.

4.7.47.2. 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 HPDI32-COS and other General Standards products.

Fields	Description
GSC_WAIT_MAIN_DMA0	This refers to the DMA Done interrupt on DMA engine number zero.
GSC_WAIT_MAIN_DMA1	This refers to the DMA Done interrupt on DMA engine number one.
GSC_WAIT_MAIN_GSC	This refers to any of the 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 accessed HPDI32-COS device.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the HPDI32-COS.
GSC_WAIT_MAIN_SPURIOUS	This refers to device interrupts which should never be generated.
GSC_WAIT_MAIN_UNKNOWN	This refers to device interrupts whose source could not be identified.

4.7.47.3. 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 Interrupt Control Register. Applications are responsible for enabling the desired interrupt options. Refer to HPDI32COS_IOCTL_IRQ_ENABLE (section 4.7.14, page 23). If a device supports the Interrupt Edge/Level Register then interrupts configured for level triggering are disabled by the driver when they occur. If a device does not support this register, then all firmware interrupts are disabled by the driver when they occur.

Value	Description
HPDI32COS_WAIT_GSC_ALL	This refers to all firmware interrupts.
HPDI32COS_WAIT_GSC_CC1_FALL	This refers to a falling edge on the Cable Command 1 signal.
HPDI32COS_WAIT_GSC_CC1_RISE	This refers to a rising edge on the Cable Command 1 signal.
HPDI32COS_WAIT_GSC_CC2	This refers to the Cable Command 2 signal.
HPDI32COS_WAIT_GSC_CC3	This refers to the Cable Command 3 signal.
HPDI32COS_WAIT_GSC_CC4	This refers to the Cable Command 4 signal.
HPDI32COS_WAIT_GSC_CC5	This refers to the Cable Command 5 signal.
HPDI32COS_WAIT_GSC_CC6	This refers to the Cable Command 6 signal.
HPDI32COS_WAIT_GSC_COS_DETECTED	This refers to the detection of a Change of State event.
HPDI32COS_WAIT_GSC_EVENT_COUNT_0	This refers to the Event Counter equaling zero.
HPDI32COS_WAIT_GSC_LA_TRIGGERED	This refers to Logic Analyzer being triggered.
HPDI32COS_WAIT_GSC_RX_FIFO_AE	This refers to the Rx FIFO's Almost Empty status.
HPDI32COS_WAIT_GSC_RX_FIFO_AF	This refers to the Rx FIFO's Almost Full status.
HPDI32COS_WAIT_GSC_RX_FIFO_EMPTY	This refers to the Rx FIFO's empty status.

HPDI32COS_WAIT_GSC_RX_FIFO_FULL	This refers to the Rx FIFO's full status.
HPDI32COS_WAIT_GSC_RX_FIFO_OVER	This refers to an Rx FIFO overflow event.
HPDI32COS_WAIT_GSC_RX_FIFO_UNDER	This refers to an Rx FIFO underflow event.
HPDI32COS_WAIT_GSC_RX_RUNNING	This refers to the receiver being enable.
HPDI32COS_WAIT_GSC_RX_STOPPED	This refers to the receiver being stopped.

4.7.47.4. gsc_wait_t.io Options

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

Fields	Description
HPDI32COS_WAIT_IO_RX_ABORT	This refers to read requests which have been aborted.
HPDI32COS_WAIT_IO_RX_DONE	This refers to read requests which have been satisfied.
HPDI32COS_WAIT_IO_RX_ERROR	This refers to read requests which end due to an error.
HPDI32COS_WAIT_IO_RX_TIMEOUT	This refers to read requests which end due to a timeout period lapse.

4.7.48. HPDI32COS_IOCTL_WAIT_STATUS

This service counts all threads blocked via the HPDI32COS_IOCTL_WAIT_EVENT IOCTL service (section 4.7.47, page 36), according to the provided criteria. A match is made when a waiting thread's wait criteria matches any of the criteria specified in the structure passed to this service.

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

Usage

Argument	Description
request	HPDI32COS_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 4.7.47.2 (page 37).
gsc	This specifies the set of HPDI32COS_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.47.3 (page 37).
alt	This is unused by the HPDI32-COS driver and should be zero.

io	This specifies the set of HPDI32COS_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.47.4 (page 38).
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 pertinent files are identified in the following table. Some source files are specific only to the HPDI32-COS, some are specific only to the OS and some are HPDI32-COS and OS independent.

Description	Files	Location	OS
Source Files	`*.c`, `*.h`	…/driver/	Linux
Header File	`hpdi32cos.h`	…/driver/	Linux
Driver File	`hpdi32cos.ko` †	…/driver/	Linux (kernels version 2.6 and later)
	`hpdi32cos.o` †	…/driver/	Linux (kernels version 2.4 and earlier)

† The Linux run time executable is implemented as a loadable kernel module.

5.2. Build

For instructions on building the driver refer to this same section number in the OS specific HPDI32-COS driver user manual.

5.3. Startup

For instructions on starting the driver executable refer to this same section number in the OS specific HPDI32-COS 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 HPDI32-COS driver user manual.

5.5. Version

For instructions on obtaining the driver version number refer to this same section number in the OS specific HPDI32-COS driver user manual.

5.6. Shutdown

For instructions on terminating the driver executable refer to this same section number in the OS specific HPDI32-COS 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 …`	`…/docsrc/`	Linux
Header File	`hpdi32cos_dsl.h`	`…/include/`	Linux
Library File	`hpdi32cos_dsl.a`	`…/lib/`	Linux

6.2. Build

For library build instructions refer to this same section number in the OS specific HPDI32-COS driver user manual.

6.3. Library Use

For library usage information refer to this same section number in the OS specific HPDI32-COS 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 `hpdi32cos_open()` there is the utility file `open.c` containing the utility function `hpdi32cos_open_util()`. The naming pattern is as follows: API function `hpdi32cos_xxxx()`, utility file name `xxxx.c`, utility function `hpdi32cos_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 `HPDI32COS_IOCTL_QUERY` there is the utility file `query.c` containing the utility function `hpdi32cos_query()`. The naming pattern is as follows: IOCTL code `HPDI32COS_IOCTL_XXXX`, utility file name `xxxx.c`, utility function `hpdi32cos_xxxx()`.

7.1. Files

The library files are summarized in the table below.

Description	Files	Location	OS
Source Files	`*.c, *,h …`	`…/utils/`	Linux
Header File	`hpdi32cos_utils.h`	`…/include/`	Linux

Library File	`hpdi32cos_utils.a`
	`gsc_utils.a`
	`os_utils.a`
	`plx_utils.a`
		`…/lib/`	Linux

7.2. Build

For library build instruction refer to this same section number in the OS specific HPDI32-COS driver user manual.

7.3. Library Use

For library usage information refer to this same section number in the OS specific HPDI32-COS driver user manual.

8. Operating Information

This section explains some basic operational procedures for using the HPDI32-COS. 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

8.1.2. API Listing

Among the utility services provided is a function to generate a listing to the console of all API settings and all register contents. When used, the function is typically used to verify the device configuration. In these cases, the function should be called just prior to the first read or digital I/O operation. 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.

Description	File/Name	Location	OS
Function	`hpdi32cos_api_listing()`	Source File	All
Source File	`api_listing.c`	`…/utils/`	All
Header File	`hpdi32cos_utils.h`	`…/include/`	All
Library File	`hpdi32cos_utils.a`	`…/lib/`	Linux

8.1.3. 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	`hpdi32cos_reg_list()`	Source File	All
Source File	`reg.c`	`…/utils/`	Linux
Header File	`hpdi32cos_utils.h`	`…/include/`	Linux
Library File	`hpdi32cos_utils.a`	`…/lib/`	Linux

8.2. I/O Modes

8.2.1. PIO - Programmed I/O

This mode transfers data by repetitive register accesses. Of the modes supported this is the least efficient. However, it is the only mode that can be used with an I/O Timeout setting of zero.

8.2.2. BMDMA - Block Mode DMA

This option uses block mode DMA transfers to fulfill application I/O requests. In this mode, movement of data by the DMA engine is initiated for reads only after the data is already present in the Rx FIFO. In this mode the driver subdivides requests, as needed, based on the current state of the Rx FIFO. Consequently, each I/O request may consist of a single DMA transfer, a few DMA transfers or of many DMA transfers. Typically, the lower the transfer clock relative to bus activity between the card and the host, the smaller and more numerous the number of transfers.

> **NOTE**: The Rx Block Mode DMA Threshold settings is used to limit the smallest size of individual DMA transfers. This is done to help improve efficiency. The BMDMA default exceeds the equivalent PIO Threshold default to limit the use of PIO to those instances where it is required to complete I/O requests. This too is done to help improve efficiency. For additional information refer to section 4.7.34 (page 31, Rx BMDMA Threshold) and section 4.7.37 (page 32, Rx PIO Threshold).

8.2.3. DMDMA - Demand Mode DMA

In this DMA mode the transfers are subdivided and started differently. First, subdividing is done only as needed to limit individual transfers to the size of the Rx transfer buffers maintained by the driver for each device. Second, the transfers are started without waiting for data to arrive in the Rx FIFO. In this mode the DMA engine moves Rx data as it appears in the Rx FIFO.

9. Sample Applications

For information on the sample applications refer to this same section number in the OS specific HPDI32-COS driver user manual.

Document History

Revision	Description
May 2, 2024	Preliminary release. Version 1.0.110.X.X.