DIO24

24-Bit Discrete Digital I/O

PCI-DIO24
PCI-DIO24-GD1

Linux Device Driver
User Manual

Manual Revision: January 25, 2005

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: http://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

1. Introduction

This user manual applies to driver version 1.06, release 0.

1.1. Purpose

The purpose of this document is to describe the interface to the DIO24 Linux device driver. This software provides the interface between "Application Software" and the DIO24 board. The interface to this board is at the device level.

1.2. Acronyms

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

Acronyms	Description
DMA	Direct Memory Access
PCI	Peripheral Component Interconnect

1.3. Definitions

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

Term	Definition
Driver	Driver means the kernel mode device driver, which runs in the kernel space with kernel mode privileges.
Application	Application means the user mode process, which runs in the user space with user mode privileges.

1.4. Software Overview

The DIO24 driver software executes under control of the Linux operating system and runs in Kernel Mode as a Kernel Mode device driver. The DIO24 device driver is implemented as a standard dynamically loadable Linux device driver written in the ‘C’ programming language. With the driver, user applications are able to open and close a device and, while open, perform I/O control operations.

1.5. Hardware Overview

The DIO24 is a simple 25-bit discrete I/O interface board. The host side connection is PCI based and the external I/O interface is via a 50 pin connector. The external interface includes 24 pin pairs that can be arbitrarily programmed as either input or output and one pin pair that is input only. The 24 programmable pins are divided into three groups of eight pins each; Port A, Port B and Port C. Ports A and B are each programmable as all inputs or all outputs. The Port C pins are individually programmable. The DIO24 has no DMA or interrupt functionality.

1.6. Reference Material

The following reference material may be of particular benefit in using the DIO24 and this driver. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *DIO24 User Manual* from General Standards Corporation.
- The *PCI9080 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

  PLX Technology Inc.
  870 Maude Avenue

Sunnyvale, California 94085 USA
Phone: 1-800-759-3735
WEB: http://www.plxtech.com

2. Installation

2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 2.2, 2.4 and 2.6 running on a PC system with one or more Intel x86 processors. Testing was performed under kernel versions 2.2.14-5 (Red Hat Linux 6.2), 2.4.20-8 (Red Hat Linux 9) and 2.6.9-1.667 (Red Hat Fedora Core 3). Testing was performed on a standard desktop PC system with a single Intel x86 processor.

> **NOTE:** The driver may have to be rebuilt before being used due to kernel version differences between the GSC build host and the customer's target host.
>
> **NOTE:** The driver has not been tested with a non-versioned kernel.
>
> **NOTE:** The driver has not been tested on an SMP host.

2.2. The /proc File System

While the driver is installed, the text file `/proc/dio24` can be read to obtain information about the driver. Each file entry includes an entry name followed immediately by a colon, a space character, then the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.06
built: Jan 25 2005, 16:29:55
boards: 1
```

Entry	Description
`version`	This gives the driver version number in the form `x.xx`.
`built`	This gives the driver build date and time in the C form of `printf("%s, %s", __DATE__, __TIME__)`.
`boards`	This identifies the total number of boards the driver detected.

2.3. File List

This release consists of the below listed files. The archives are described in detail in following subsections.

File	Description
`dio24.tar.gz`	This archive contains the driver and all related sources.
`dio24_linux_driver_user_manual.pdf`	This is a PDF version of this user manual.

2.4. Installation

Install the driver and its related files following the below listed steps. This includes the device driver, the documentation source code, and the sample application.

1. Change the current directory to `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Copy the archive file `dio24.tar.gz` into the current directory.

3. Issue the following command to decompress and extract the files from the provided archive. This creates the directory `dio24` in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf dio24.tar.gz
```

2.5. Removal

Follow the below steps to remove the driver and its related files. This includes the device driver, the documentation source code, and the sample application.

1. Shutdown the driver as described in following paragraphs.

2. Change to the directory where the driver archive was installed. This should be `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

3. Issue the below command to remove the driver archive and all of the installed driver files.

   ```
   rm –rf dio24.tar.gz dio24
   ```

4. Issue the below command to remove all of the installed device nodes.

   ```
   rm –rf /dev/dio24*
   ```

5. If the automated startup procedure was adopted (described in following paragraphs), then edit the system startup script `rc.local` and remove the line that invokes the `dio24_start` script. The file `rc.local` should be located in the `/etc/rc.d` directory.

2.6. The Driver

This driver and its related files are contained in the archive file `dio24.tar.gz`. The archive's device driver files are listed below. The paragraphs that follow give installation, build and startup instructions.

File	Description
`*.c`	The driver source files.
`*.h`	The driver header files.
`dio24.h`	A driver header file. This header should be included by DIO24 applications.
`dio24.ko` *	The driver executable.
`dio24_start`	Shell script to install the driver executable and device nodes.
`makefile`	The driver make file.
`makefile.dep`	An automatically generated make dependency file.

* The file name extension for the pre-built driver executable is `.ko`, which is the convention for the 2.6 kernel. The 2.6 build of the driver executable is included in the release as the final release is built using the 2.6 kernel. The convention for the 2.2 and 2.4 kernels is an extension of `.o`. The driver build procedure produces the appropriately named file.

2.6.1. Build

NOTE: Building the driver requires installation of the kernel sources.

Follow the below steps to build the driver.

1. Change to the directory where the driver and its sources were installed. This should be `/usr/src/linux/drivers/dio24/driver`.

2. Remove all existing build targets by issuing the below command.

```
make –f makefile clean
```

3. Build the driver by issuing the below command.

```
make –f makefile all
```

NOTE: Due to the differences between the many Linux distributions some build errors may occur. These errors may include system header location differences and should be easily correctable.

2.6.2. Startup

The startup script used in this procedure is designed to insure that the driver module in the install directory is the module that is loaded. This is accomplished by making sure that an already loaded module is first unloaded before attempting to load the module from the disk drive. In addition, the script also deletes and recreates the device nodes. This is done to insure that the device nodes in use have the same major number as assigned dynamically to the driver by the kernel, and so that the number of device nodes correspond to the number of boards identified by the driver.

NOTE: The driver may have to be rebuilt before being used due to kernel version differences between the GSC build host and the customer target host.

2.6.2.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

1. Login as root user, as some of the steps require root privileges.

2. Change to the directory where the driver was installed. This should be `/usr/src/linux/drivers/dio24/driver`.

3. Install the driver module and create the device nodes by executing the below command. If any errors are encountered then an appropriate error message will be displayed.

```
./dio24_start
```

NOTE: The script's default specifies that the driver is installed in the same directory as the script. The script will fail if this is not so.

NOTE: The above step must be repeated each time the host is rebooted.

NOTE: The DIO24 device node major number is assigned dynamically by the kernel. The minor numbers and the device node suffix numbers are index numbers beginning with zero, and increase by one for each additional board installed.

4. Verify that the device module has been loaded by issuing the below command and examining the output. The module name `dio24` should be included in the output.

```
lsmod
```

5. Verify that the device nodes have been created by issuing the below command and examining the output. The output should include one node for each installed board.

```
ls –l /dev/dio24*
```

2.6.2.2. Automatic Driver Startup Procedures

Start the driver automatically with each system reboot by following the below listed steps.

1. Locate and edit the system startup script `rc.local`, which should be in the `/etc/rc.d` directory. Modify the file by adding the below line so that it is executed with every reboot.

   ```
   /usr/src/linux/drivers/dio24/driver/dio24_start
   ```

   NOTE: The script's default specifies that the driver is installed in the same directory as the script. The startup script will fail if this is not so.

2. Load the driver and create the required device nodes by rebooting the system.

3. Verify that the driver is loaded and that the device nodes have been created by following the verification steps given in the manual startup procedures.

2.6.3. Verification

WARNING: When using the test application the DIO24 and any externally attached equipment may be damaged if the DIO24's external interface has a cable other than an appropriate loop back cable attached. Damage may result because the test application methodically configures each I/O pin as an input then an output, and alternately drives the output to both its high and low states. No damage will result if no cable at all is attached.

NOTE: Refer to the board user manual for information on the loop back test cable.

Follow the below steps to verify that the driver has been properly installed and started.

1. Install the sample applications as described in subsequent paragraphs.

2. Change to the directory where the test application `dio24Test.o` was installed.

3. Start the test application by issuing the below command.

   ```
   ./dio24Test.o
   ```

   NOTE: As described elsewhere, the test application fill not successfully test a DIO24 if an appropriate loop back cable is not attached to the external interface. The cable requirements are described elsewhere as well. If a cable is unattached the application will still test the driver and the DIO24's PCI interface.

2.6.4. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in `/var/log/messages`). It is recorded in the text file `/proc/dio24`. It can also be read by an application via the `DIO24_IOCTL_DRIVER_INFO_GET` IOCTL services.

2.6.5. Shutdown

Shutdown the driver following the below listed steps.

1. Login as root user, as some of the steps require root privileges.

2. If the driver is currently loaded then issue the below command to unload the driver.

   ```
   rmmod dio24
   ```

3. Verify that the driver module has been unloaded by issuing the below command. The module name `dio24` should not be in the list.

   ```
   lsmod
   ```

2.7. Document Source Code Examples

The archive file `dio24.tar.gz` contains all of the source code examples included in this document. In addition, they are included as a statically linkable library usable with DIO24 console applications. The library and sources are delivered undocumented and unsupported. 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. The archive content is not described here, though its use is described in the following paragraphs. These files are installed into the directory `/usr/src/linux/drivers/dio24/docsrc`.

File	Description
`*.c`	These are the C source files.
`DIO24DocSrcLib.a`	This is a pre-built, linkable version of the library.
`DIO24DocSrcLib.h`	This is the library header file.
`makefile`	This is the library make file.
`makeile.dep`	This is an automatically generated make dependency file.

2.7.1. Build

Follow the below steps to compile the example files.

1. Change to the directory where the source code example files were installed. This should be `/usr/src/linux/drivers/dio24/docsrc`.

2. Remove all existing build targets by issuing the below command.

   ```
   make –f makefile clean
   ```

3. Compile the sample files by issuing the below command.

   ```
   make –f makefile all
   ```

NOTE: The build procedure will fail if the driver sources are not installed in the directory documented in the driver installation procedures.

2.7.2. Use

The library is used both at application compile time and at application link time. Compile time use has two requirements. First, include the header file `DIO24DocSrcLib.h` in each module referencing a library component. Second, expand the include file search path to search the directory where the library header is located. This should be `/usr/src/linux/drivers/dio24/docsrc`. Link time use also has two requires. First, include the static library `DIO24DocSrcLib.a` in the list of files to be linked into the application. Second, expand the library file search path to search the directory where the library is located. This should also be `/usr/src/linux/drivers/dio24/docsrc`.

2.8. Sample Application

2.8.1. dio24Test.o

> **CAUTION:** When using the sample application the DIO24 and any externally attached equipment may be damaged if the DIO24's external interface has a cable attached. Damage may result because the sample application uses a configuration which drives various external output signals. No damage will result if no cable at all is attached.

This sample application provides a command line driven Linux application that tests the functionality of the driver and a user specified DIO24 board. It can be used as the starting point for application development on top of the DIO24 Linux device driver. It utilizes the library `DIO24DocSrcLib` and is delivered undocumented and unsupported. It can however be used as a starting point for developing applications on top of the Linux driver and to help ease the learning curve. The application performs an automated test of the driver features. The application includes the below listed files, installed into the directory `/usr/src/linux/drivers/dio24/test`.

> **NOTE:** For the test application to successfully verify the functionality of a DIO24, an appropriate loop back cable must be attached to the external interface. Refer to the board user manual for information on the loop back test cable.

File	Description
`dio24Test.c`	This is the source file.
`dio24Test.o`	This is the pre-built test application.
`makefile`	This is the application make file.
`makeile.dep`	This is an automatically generated make dependency file.
`makeapp.sh`	This is the build script for the sample application.

2.8.2. Build

Follow the below steps to build/rebuild the sample application.

1. Change to the directory where the sample application was installed. This should be `/usr/src/linux/drivers/dio24/test`.

2. Remove all existing build targets by issuing the below command.

   ```
   make –f makefile clean
   ```

3. Build the application by issuing the below command.

   ```
   make –f makefile all
   ```

2.8.3. Execute

> **WARNING:** When using the test application the DIO24 and any externally attached equipment may be damaged if the DIO24's external interface has a cable other than an appropriate loop back cable attached. Damage may result because the test application methodically configures each I/O pin as an input then an output, and alternately drives the output to both its high and low states. No damage will result if no cable at all is attached.

> **NOTE:** If an appropriate loop back cable is not attached, then the application will fail as soon as the procedure begins testing the DIO24's I/O capabilities. In this case, the application still verifies the operation of the driver and the board's PCI interface.

NOTE: Refer to the board user manual for information on the loop back test cable.

Follow the below steps to execute and exercise the test application.

1. Change to the directory where the sample application was installed.

2. Start the sample application by issuing the command given below. The application uses the command line arguments given to direct its course of action. Once started the application will automatically execute a series of tests to verify the operation of the driver and the board. A single test cycle should take less than one second to complete. The "`index`" argument specifies the index of the board to access and is required only if multiple boards are installed. The first board is index zero (0). The argument "`-c`" specifies to run the test continuously until an error is encountered. The argument "`-C`" specifies to run the test continuously even if errors are encountered. The argument `-n#`, where `#` is a integer number, is the maximum number of tests to run when continuous testing is being performed. The argument `-m#`, where `#` is a integer number, is the maximum number of minutes to run tests when continuous testing is being performed. At the end of each test cycle the application reports accumulative test results.

   ```
   ./dio24Test.o <-c> <-C> <-n#> <-m#> <index>
   ```

3. Driver Interface

The DIO24 driver conforms to the device driver standards required by the Linux Operating System and contains the standard driver entry points. The device driver provides a standard driver interface to the GSC DIO24 board for Linux applications. The interface includes various macros, data types and functions, all of which are described in the following paragraphs. The DIO24 specific portion of the driver interface is defined in the header file `dio24.h`, portions of which are described in this section. The header defines numerous items in addition to those described here.

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

3.1. Macros

The driver interface includes the following macros which are defined in `dio24.h`. The header also contains various other utility type macros which are provided without documentation.

3.1.1. IOCTL

The IOCTL macros are documented following the function call descriptions.

3.1.2. Registers

The following table gives the complete set of DIO24 registers. The tables are divided by register categories. Unless otherwise stated, all registers are accessed by their native size of eight, 16 or 32-bits. The only exception is the PCICCR register which is 24 bits wide but accessed as if it were 32-bits wide. In this instance the upper eight-bits are to be ignored. Register values are passed as 32-bit entities, but bits outside the a register's native size are ignored.

3.1.2.1. GSC Registers

The following table gives the complete set of GSC specific DIO24 registers. For detailed definitions of these registers refer to the *DIO24 User Manual*.

Macros	Description
`DIO24_GSC_BCR`	Board Control Register (BCR)
`DIO24_GSC_BSR`	Board Status Register (BSR)
`DIO24_GSC_DDIR`	Discrete Data Input Register (DDIR)
`DIO24_GSC_DDOR`	Discrete Data Output Register (DDOR)
`DIO24_GSC_FRR`	Firmware Revision Register (FRR)
`DIO24_GSC_IOCR`	I/O Control Register (IOCR)

> **NOTE:** For DIO24-GD1 variations of the DIO24, an FRR value of 0xXX0BXXXX identifies the board as a GD1 version rather than an HPDI32. This is relevant for identification only, and does not reflect any functional differences. Refer to the PCI registers for additional identification information.

3.1.2.2. PLX PCI 9080 Registers

The following table gives the complete set of PLX PCI9080 registers. For detailed definitions of these registers refer to the *PCI9080 Data Book*. The registers are presented as grouped in the PLX data book.

PCI Configuration Registers

Macros	Description
DIO24_PCI_BAR0	PCI Base Address Register for Memory Accesses to Local, Runtime, and DMA Registers (PCIBAR0)
DIO24_PCI_BAR1	PCI Base Address Register for I/O Accesses to Local, Runtime, and DMA Registers (PCIBAR1)
DIO24_PCI_BAR2	PCI Base Address Register for Memory Accesses to Local Address Space 0 (PCIBAR2)
DIO24_PCI_BAR3	PCI Base Address Register for Memory Accesses to Local Address Space 1 (PCIBAR3)
DIO24_PCI_BAR4	Unused Base Address (PCIBAR4)
DIO24_PCI_BAR5	Unused Base Address (PCIBAR5)
DIO24_PCI_BISTR	PCI Built-In Self Test Register (PCIBISTR)
DIO24_PCI_CCR	PCI Class Code Register (PCICCR)
DIO24_PCI_CIS	PCI Cardbus CIS Pointer Register (PCICIS)
DIO24_PCI_CLSR	PCI Cache Line Size Register (PCICLSR)
DIO24_PCI_CR	PCI Command Register (PCICR)
DIO24_PCI_ERBAR	PCI Expansion ROM Base Address (PCIERBAR)
DIO24_PCI_HTR	PCI Header Type Register (PCIHTR)
DIO24_PCI_IDR	PCI Configuration ID Register (PCIIDR)
DIO24_PCI_ILR	PCI Interrupt Line Register (PCIILR)
DIO24_PCI_IPR	PCI Interrupt Pin Register (PCIIPR)
DIO24_PCI_LTR	PCI Latency Timer Register (PCILTR)
DIO24_PCI_MGR	PCI Min_Gnt Register (PCIMGR)
DIO24_PCI_MLR	PCI Max_Lat Register (PCIMLR)
DIO24_PCI_REV	PCI Revision ID Register (PCIREV)
DIO24_PCI_SID	PCI Subsystem ID Register (PCISID)
DIO24_PCI_SR	PCI Status Register (PCISR)
DIO24_PCI_SVID	PCI Subsystem Vendor ID Register (PCISVID)

NOTE: A PCIIDR value of 0x908010B5 identifies the PCI interface chip as a PLX PCI9080. A PCISVID value of 0x10B5 indicates that the PCISID value has been assigned by PLX. A PCISID value of 0x2706 identifies the board as a DIO24. A PCISID value of 0x2400 identifies that the DIO24 is a GD1 variation, which falls under the HPDI32 product line. Refer to the GSC Firmware Revision Register for additional identification information. This is relevant for identification only, and does not reflect any functional differences.

Local Configuration Registers

Macros	Description
DIO24_PLX_BIGEND	Big/Little Endian Descriptor Register (BIGEND)
DIO24_PLX_DMCFGA	PCI Configuration Address Register for Direct Master to PCI IO/CFG (DMCFGA)
DIO24_PLX_DMLBAI	Local Bus Base Address Register for Direct Master to PCI IO/CFG (DMLBAI)
DIO24_PLX_DMLBAM	Local Bus Base Address Register for Direct Master to PCI Memory (DMLBAM)
DIO24_PLX_DMPBAM	PCI Base Address Register for Direct Master to PCI Memory (DMPBAM)
DIO24_PLX_DMRR	Local Range Register for Direct Master to PCI (DMRR)
DIO24_PLX_EROMBA	Expansion ROM Local Base Address Register (EROMBA)
DIO24_PLX_EROMRR	Expansion ROM Range Register (EROMRR)
DIO24_PLX_LAS0BA	Local Address Space 0 Local Base Address Register (LASOBA)
DIO24_PLX_LAS0RR	Local Address Space 0 Range Register for PCI-to-Local Bus (LASORR)
DIO24_PLX_LAS1BA	Local Address Space 1 Local Base Address Register (LAS1BA)
DIO24_PLX_LAS1RR	Local Address Space 1 Range Register for PCI-to-Local Bus (LAS1RR)
DIO24_PLX_LBRD0	Local Address Space 0/Expansion ROM Bus Region Descriptor Register (LBRD0)
DIO24_PLX_LBRD1	Local Address Space 1 Bus Region Descriptor Register (LBRD1)
DIO24_PLX_MARBR	Mode Arbitration Register (MARBR)

Runtime Registers

Macros	Description
DIO24_PLX_CNTRL	Serial EEPROM Control, CPI Command Codes, User I/O, Init Control Register (CNTRL)
DIO24_PLX_INTCSR	Interrupt Control/Status Register (INTCSR)
DIO24_PLX_L2PDBELL	Local-to-PCI Doorbell Register (L2PDBELL)
DIO24_PLX_MBOX0	Mailbox Register 0 (MBOX0)
DIO24_PLX_MBOX1	Mailbox Register 1 (MBOX1)
DIO24_PLX_MBOX2	Mailbox Register 2 (MBOX2)
DIO24_PLX_MBOX3	Mailbox Register 3 (MBOX3)
DIO24_PLX_MBOX4	Mailbox Register 4 (MBOX4)
DIO24_PLX_MBOX5	Mailbox Register 5 (MBOX5)
DIO24_PLX_MBOX6	Mailbox Register 6 (MBOX6)
DIO24_PLX_MBOX7	Mailbox Register 7 (MBOX7)
DIO24_PLX_P2LDBELL	PCI-to-Local Doorbell Register (P2LDBELL)
DIO24_PLX_PCIHIDR	PCI Permanent Configuration ID Register (PCIHIDR)
DIO24_PLX_PCIHREV	PCI Permanent Revision ID Register (PCIHREV)

DMA Registers

Macros	Description
DIO24_PLX_DMAARB	DMA Arbitration Register (DMAARB)
DIO24_PLX_DMACSR0	DMA Channel 0 Command/Status Register (DMACSR0)
DIO24_PLX_DMACSR1	DMA Channel 1 Command/Status Register (DMACSR1)
DIO24_PLX_DMADPR0	DMA Channel 0 Descriptor Pointer Register (DMADPR0)
DIO24_PLX_DMADPR1	DMA Channel 1 Descriptor Pointer Register (DMADPR1)
DIO24_PLX_DMALADR0	DMA Channel 0 Local Address Register (DMALADR0)
DIO24_PLX_DMALADR1	DMA Channel 1 Local Address Register (DMALADR1)
DIO24_PLX_DMAMODE0	DMA Channel 0 Mode Register (DMAMODE0)
DIO24_PLX_DMAMODE1	DMA Channel 1 Mode Register (DMAMODE1)
DIO24_PLX_DMAPADR0	DMA Channel 0 PCI Address Register (DMAPADR0)
DIO24_PLX_DMAPADR1	DMA Channel 1 PCI Address Register (DMAPADR1)
DIO24_PLX_DMASIZ0	DMA Channel 0 Transfer Size Register (DMASIZ0)
DIO24_PLX_DMASIZ1	DMA Channel 1 Transfer Size Register (DMASIZ1)
DIO24_PLX_DMATHR	DMA Threshold Register (DMATHR)

Message Queue Registers

Macros	Description
DIO24_PLX_IFHPR	Inbound Free Head Pointer Register (IFHPR)
DIO24_PLX_IFTPR	Inbound Free Tail Pointer Register (IFTPR)
DIO24_PLX_IPHPR	Inbound Post Head Pointer Register (IPHPR)
DIO24_PLX_IPTPR	Inbound Post Tail Pointer Register (IPTPR)
DIO24_PLX_IQP	Inbound Queue Port Register (IQP)
DIO24_PLX_MQCR	Messaging Queue Configuration Register (MQCR)
DIO24_PLX_OFHPR	Outbound Free Head Pointer Register (OFHPR)
DIO24_PLX_OFTPR	Outbound Free Tail Pointer Register (OFTPR)
DIO24_PLX_OPHPR	Outbound Post Head Pointer Register (OPHPR)
DIO24_PLX_OPLFIM	Outbound Post List FIFO Interrupt Mask Register (OPLFIM)
DIO24_PLX_OPLFIS	Outbound Post List FIFO Interrupt Status Register (OPLFIS)
DIO24_PLX_OPTPR	Outbound Post Tail Pointer Register (OPTPR)
DIO24_PLX_OQP	Outbound Queue Port Register (OQP)
DIO24_PLX_QBAR	Queue Base Address Register (QBAR)

DIO24_PLX_QSR	Queue Status/Control Register (QSR)

3.2. Data Types

This driver interface includes the following data types which are defined in dio24.h.

3.2.1. dio24_debug_data_t

This structure is used for driver debugging purposes and is used only during driver development.

Definition

```
typedef struct
{
    __u32   valid;
    __s8    msg[128];
    __u32   ul[16];
} dio24_debug_data_t;
```

Fields	Description
valid	If this field is non-zero then the recorded data is valid. The data is otherwise meaningless.
msg	This is a textual message relating to some operation being debugged.
ul	This is a set of values related to the above message.

3.2.2. dio24_driver_info_t

This structure defines the data fields for the information returned by the DIO24_DRIVER_INFO_GET IOCTL service.

Definition

```
typedef struct
{
    __s8    version[8];
    __s8    built[32];
} dio24_driver_info_t;
```

Fields	Description
version	This field gives the driver version number as a string in the form of X.XX.
built	This field gives the driver build date and time as a string. It is given in the C form of printf("%s, %s", __DATE__, __TIME__).

3.2.3. dio24_reg_t

This structure defines the data fields for the information involved in the register access IOCTL services. Read the details of the individual services for additional information.

Definition

```
typedef struct
{
    __u32   reg;
    __u32   value;
    __u32   mask;
```

```
} dio24_reg_t;
```

Fields	Description
reg	This field identifies the register to be accessed.
value	This field identifies the value retrieved by read operations and the value to apply by write operations.
mask	This field identifies the register bits from the value field that are to be applied during the read-modify-write IOCTL service. If a bit is set in the mask, then the corresponding value bit is applied to the register. If a mask bit is not set then the corresponding register bit is left unchanged.

3.3. Functions

This driver interface includes the following functions.

3.3.1. close()

This function is the entry point to close a connection to an open DIO24 board. This function should only be called after a successful open of the respective device.

Prototype

```
int close(int fd);
```

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

Return Value	Description
-1	An error occurred. Consult errno.
0	The operation succeeded.

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <unistd.h>

#include "DIO24DocSrcLib.h"

int dio24_close(int fd, int verbose)
{
    int status;

    status  = close(fd);

    if ((verbose) && (status == -1))
        printf("close() failure, errno = %d\n", errno);

    return(status);
}
```

3.3.2. ioctl()

This function is the entry point to performing setup and control operations on a DIO24 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 any additional arguments. The set of supported IOCTL services is defined in a following section.

Prototype

```
int ioctl(int fd, int request, ...);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`request`	This specifies the desired operation to be performed.
`...`	This is any additional arguments. If `request` does not call for any additional arguments, then any additional arguments provided are ignored. The DIO24 IOCTL services use at most one argument, which is represented by a 32-bit value.

Return Value	Description
`-1`	An error occurred. Consult `errno`.
`0`	The operation succeeded.

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_ioctl(int fd, int request, void *arg, int verbose)
{
    int status;

    status  = ioctl(fd, request, (unsigned long) arg);

    if ((verbose) && (status == -1))
        printf("ioctl() failure, errno = %d\n", errno);

    return(status);
}
```

3.3.3. open()

This function is the entry point to open a connection to a DIO24 board.

Prototype

```
int open(const char* pathname, int flags);
```

Argument	Description
`pathname`	This is the name of the device to open.
`flags`	This is the desired read/write access. Use `O_RDWR`.

NOTE: Another form of the `open()` function has a `mode` argument. This form is not displayed here as the `mode` argument is ignored when opening an existing file/device.

Return Value	Description
`-1`	An error occurred. Consult `errno`.
else	A valid file descriptor.

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <assert.h>
#include <errno.h>
#include <fcntl.h>
#include <stdio.h>

#include "DIO24DocSrcLib.h"

int dio24_open(int board, int verbose)
{
    int     fd;
    char    name[80];

    assert(board >= 0);

    sprintf(name, "/dev/dio24%d", board);
    fd  = open(name, O_RDWR);

    if ((verbose) && (fd == -1))
    {
        printf( "open() failure on %s, errno = %d\n",
                name,
                errno);
    }

    return(fd);
}
```

3.3.4. read()

The DIO24 does not support a read operation as the board has neither data storage nor synchronous data reception capability. Data read operations are performed by reading the appropriate register.

3.3.5. write()

The DIO24 does not support a write operation as the board has neither data storage nor synchronous data transmission capability. Data write operations are performed by writing to the appropriate register.

3.4. IOCTL Services

The DIO24 driver implements the following IOCTL services. Each service is described along with the applicable `ioctl()` function arguments. In the definitions given the optional argument is identified as `arg` and is an `unsigned long` data type. Unless otherwise stated the return value definitions are those defined for the `ioctl()` function call and any errors codes are accessed via `errno`.

3.4.1. DIO24_IOCTL_DEBUG_DATA_GET

This service retrieves debug data produced by temporary code included in the driver only during driver debugging and development.

Usage

ioctl() Argument	Description
request	DIO24_IOCTL_DEBUG_DATA_GET
arg	dio24_debug_data_t*

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_debug_read(int fd, int verbose)
{
    dio24_debug_data_t  debug;
    int                 i;
    int                 j;
    int                 status;

    status  = ioctl(fd, DIO24_IOCTL_DEBUG_DATA_GET, &debug);

    if (verbose == 0)
    {
    }
    else if (status == -1)
    {
        printf("ioctl() failure, errno = %d\n", errno);
    }
    else if (debug.valid)
    {
        printf("Debug Data:\n");
        printf("  msg: '%s'\n", debug.msg);
        i   = sizeof(debug.ul) / sizeof(debug.ul[0]);

        for (j = 0; j < i; j++)
        {
            printf( "  ul[%2d]: 0x%08lX\n",
                    j,
                    (unsigned long) debug.ul[j]);
        }
    }

    return(status);
}
```

3.4.2. DIO24_IOCTL_DRIVER_INFO_GET

This service retrieves information about the driver itself.

Usage

ioctl() Argument	Description
request	DIO24_IOCTL_DRIVER_INFO_GET
arg	dio24_driver_info_t*

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_driver_info(int fd, dio24_driver_info_t* info, int verbose)
{
    int status;

    status  = ioctl(fd,
                    DIO24_IOCTL_DRIVER_INFO_GET,
                    (unsigned long) info);

    if ((verbose) && (status == -1))
        printf("ioctl() failure, errno = %d\n", errno);

    return(status);
}
```

3.4.3. DIO24_IOCTL_NO_COMMAND

This is an empty driver entry point. This IOCTL may be given to verify that the driver is correctly installed and that a DIO24 has been successfully opened. If an error status is returned then something isn't working properly.

Usage

ioctl() Argument	Description
request	DIO24_IOCTL_NO_COMMAND
arg	Not used.

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_no_command(int fd, int verbose)
{
    int status;

    status  = ioctl(fd, DIO24_IOCTL_NO_COMMAND);

    if ((verbose) && (status == -1))
```

```
        printf("ioctl() failure, errno = %d\n", errno);

    return(status);
}
```

3.4.4. DIO24_IOCTL_REG_MOD

This service performs a read-modify-write operation on a DIO24 register. This includes only the GSC specific registers. All PCI and PLX PCI9080 feature set registers are read-only. Refer to `dio24.h` for a complete list of the accessible registers.

Usage

ioctl() Argument	**Description**
request	DIO24_IOCTL_REG_MOD
arg	dio24_reg_t*

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_reg_mod(
    int             fd,
    unsigned long   reg,
    unsigned long   value,
    unsigned long   mask,
    int             verbose)
{
    dio24_reg_t parm;
    int         status;

    parm.reg    = reg;
    parm.value  = value;
    parm.mask   = mask;
    status      = ioctl(fd,
                        DIO24_IOCTL_REG_MOD,
                        (unsigned long) &parm);

    if ((verbose) && (status == -1))
        printf("ioctl() failure, errno = %d\n", errno);

    return(status);
}
```

3.4.5. DIO24_IOCTL_REG_READ

This service reads the value of a DIO24 register. This includes all PCI registers, all PLX PCI9080 feature set registers, and all GSC specific registers. Refer to `dio24.h` for a complete list of the accessible registers.

Usage

ioctl() Argument	Description
request	DIO24_IOCTL_REG_READ
arg	dio24_reg_t*

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_reg_read(
    int             fd,
    unsigned long   reg,
    unsigned long*  value,
    int             verbose)
{
    dio24_reg_t parm;
    int         status;

    parm.reg    = reg;
    parm.value  = 0xDEADBEEFL;
    parm.mask   = 0;    /* ignored for reads */
    status      = ioctl(fd,
                        DIO24_IOCTL_REG_READ,
                        (unsigned long) &parm);

    if (status == 0)
        value[0]    = parm.value;
    else if (verbose)
        printf("ioctl() failure, errno = %d\n", errno);

    return(status);
}
```

3.4.6. DIO24_IOCTL_REG_WRITE

This service writes a value to a DIO24 register. This includes only the GSC specific registers. All PCI and PLX PCI9080 feature set registers are read-only. Refer to `dio24.h` for a complete list of the accessible registers.

Usage

ioctl() Argument	Description
request	DIO24_IOCTL_REG_WRITE
arg	dio24_reg_t*

Example

```
/* This software is covered by the GNU GENERAL PUBLIC LICENSE (GPL). */
#include <errno.h>
#include <stdio.h>
```

```
#include <sys/ioctl.h>

#include "DIO24DocSrcLib.h"

int dio24_reg_write(
    int             fd,
    unsigned long   reg,
    unsigned long   value,
    int             verbose)
{
    dio24_reg_t parm;
    int         status;

    parm.reg    = reg;
    parm.value  = value;
    parm.mask   = 0;    /* ignored for writes */
    status      = ioctl(fd,
                        DIO24_IOCTL_REG_WRITE,
                        (unsigned long) &parm);

    if ((verbose) && (status == -1))
        printf("ioctl() failure, errno = %d\n", errno);

    return(status);
}
```

Document History

Revision	Description
January 25, 2005	Updated to release 1.06.0. Minor editorial changes.
January 13, 2005	Reorganized the directory structure. Ported to the 2.6 kernel. Made the source code samples into a library.
August 31, 2004	Made the driver GPL. Made other minor document mods.
April 26, 2003	Added support for the PCI-DIO24 (no suffix).
July 24, 2002	Ported to the 2.4.7 and 2.4.18 kernels plus other minor updates.
January 29, 2002	Initial release.