PEX8112

PCI Express/PCI Bus Bridge Interface

PEX8112

Linux Device Driver And API Library User Manual

Manual Revision: October 11, 2022
Driver Release Version 2.3.101.43.0

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

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the PEX8112 API Library and to the underlying Linux 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 PEX8112 hardware. The API Library and driver interfaces are based on the devices's functionality.

1.2. Acronyms

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

Acronyms	Description
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PCIe	PCI Express

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 PEX8112 installation directory or any of its subdirectories.
API Library	This refers to the library implementing the PEX8112 API, which is implemented as a shared library.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is a kernel mode device driver, which runs in kernel space with kernel mode privileges.
Library	This is usually a general reference to the API Library.
PEX8112	This is used as a general reference to any device supported by this driver.

1.4. Software Overview

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise PEX8112 applications. The overall architecture is illustrated in Figure 1 below.

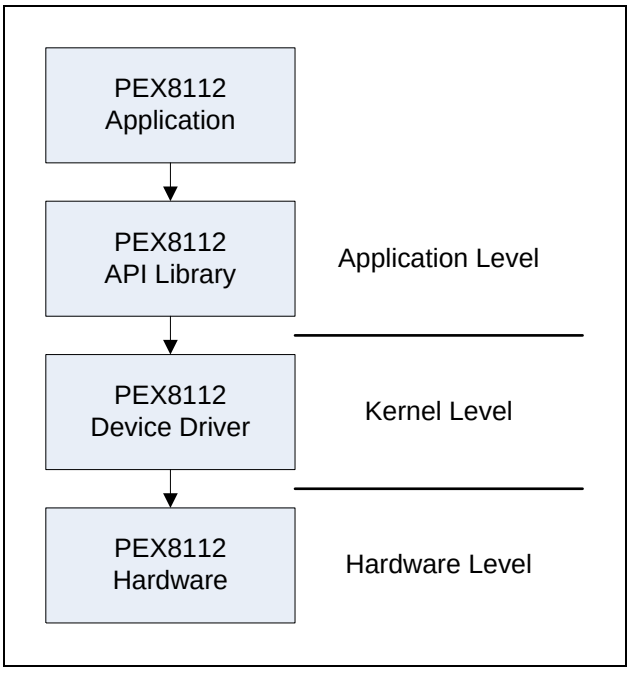

Figure 1 Architectural representation.

1.4.2. API Library

The primary means of accessing PEX8112 devices is via the PEX8112 API Library. This library forms a very thin layer between the application and the driver. Additional information is given in section 4 beginning on page 16. With the library, applications are able to open and close a device and, while open, perform I/O control operations. (See Figure 1 above.)

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with PEX8112 hardware. The driver executes under control of the operating system and runs in Kernel Mode as a Kernel Mode device driver. The driver is implemented as a standard dynamically loadable Linux device driver written in the C programming language. While applications can access the driver directly without use of the API Library, it is recommended that all access is made through the library. (See Figure 1 above.)

1.5. Hardware Overview

The PEX8112 is a high-performance bridge interface chip providing an interface between a PCI Express slot and a PCI Bridge chip, such as the PLX PCI 9056 or the PLX PCI 9080.

1.6. Reference Material

The following reference material may be of particular benefit in using the PEX8112. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this device.

- The *PEX8112 PCI Express Bridge 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

1.7. Licensing

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

2. Installation

2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 5.x, 4.x, 3.x, 2.6, 2.4 and 2.2 running on a PC system with one or more x86 processors. This release of the driver supports the below listed kernels.

Kernel	Distribution
5.14.10	Red Hat Fedora Core 35
5.11.12	Red Hat Fedora Core 34
5.8.15	Red Hat Fedora Core 33
5.6.6	Red Hat Fedora Core 32
5.3.7	Red Hat Fedora Core 31
5.0.9	Red Hat Fedora Core 30
4.18.16	Red Hat Fedora Core 29
4.16.3	Red Hat Fedora Core 28
4.13.9	Red Hat Fedora Core 27
4.11.8	Red Hat Fedora Core 26
4.8.6	Red Hat Fedora Core 25
4.5.5	Red Hat Fedora Core 24
4.2.3	Red Hat Fedora Core 23
4.0.4	Red Hat Fedora Core 22
3.17.4	Red Hat Fedora Core 21
3.11.10	Red Hat Fedora Core 20
3.9.5	Red Hat Fedora Core 19
3.6.10	Red Hat Fedora Core 18
3.3.4	Red Hat Fedora Core 17
3.1.0	Red Hat Fedora Core 16
2.6.38	Red Hat Fedora Core 15
2.6.35	Red Hat Fedora Core 14
2.6.33	Red Hat Fedora Core 13
2.6.31	Red Hat Fedora Core 12
2.6.29	Red Hat Fedora Core 11
2.6.27	Red Hat Fedora Core 10
2.6.25	Red Hat Fedora Core 9
2.6.23	Red Hat Fedora Core 8
2.6.21	Red Hat Fedora Core 7
2.6.18	Red Hat Fedora Core 6
2.6.15	Red Hat Fedora Core 5
2.6.11	Red Hat Fedora Core 4
2.6.9	Red Hat Fedora Core 3

NOTE: Some older kernel versions are supported (the sources are maintained), but are not tested.

NOTE: While only Red Hat Fedora distributions are listed, numerous other distributions are supported and have been tested on an as needed basis.

NOTE: The driver will have to be built before being used as it is provided in source form only.

NOTE: The driver has not been tested with a non-versioned kernel.

NOTE: The driver has not been tested for SMP operation.

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "`32-bit support`" field in the `/proc/pex8112` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/pex8112` can be read to obtain information about the driver. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 2.3.101.43
32-bit support: yes
boards: 1
models: PEX8112
```

Entry	Description
`version`	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 devices the driver detected.
`models`	This gives a comma separated list of the device models identified by the driver. One model will be listed for each device identified in the system. For this driver the only model numbers listed will be "`PEX8112`."

2.3. File List

This release consists of the below listed primary files. The archive content is described in following subsections.

File	Description
`pex8112.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`pex8112_linux_um.pdf`	This is a PDF version of this user manual, which is included in the archive.

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.

Directory	Content
`pex8112/`	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 PEX8112 API Library sources (section 4, page 16).
`…/docsrc/`	This directory contains the code samples from this document (section 6, page 29).
`…/driver/`	This directory contains the driver and its sources (section 5, page 25).
`…/include/`	This directory contains the include files for the various libraries.
`…/lib/`	This directory contains all of the libraries built from the driver archive.
`…/samples/`	This directory contains the sample applications (section 9, page 32).

…/utils/	This directory contains utility sources used by the sample applications (section 7, page 30).

2.5. Installation

Perform installation following the below listed steps. This installs the device driver, the API Library and all related sources and documentation.

1. Create and change to the directory where the files are to be installed, such as /usr/src/linux/drivers/. (The path name may vary among distributions and kernel versions.)

2. Copy the archive file pex8112.linux.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 pex8112 in the current directory, and then copies all of the archive's files into this new directory.

   ```
   tar –xzvf pex8112.linux.tar.gz
   ```

2.6. Removal

Perform removal following the below listed steps. This removes the device driver, the API Library and all related sources and documentation.

1. Shutdown the driver as described in section 5.6 on page 28.

2. Change to the directory where the driver archive was installed, which may have been /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 pex8112.linux.tar.gz pex8112
   ```

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

   ```
   rm –f /dev/pex8112.*
   ```

5. If the automated startup procedure was adopted (section 5.3.2, page 26), then edit the system startup script rc.local and remove the line that invokes the PEX8112's start script. The file rc.local should be located in the /etc/rc.d/ directory.

2.7. Overall Make Script

An Overall Make Script is included in the root installation directory. Executing this script will perform a make for all build targets included in the release. The script also loads the driver and copies the API Library to /usr/lib/. The script is named make_all. Follow the below steps to perform an overall make and to load the driver.

> **NOTE**: The following steps may require elevated privileges.

1. Change to the root installation directory (…/pex8112/).

2. Remove existing build targets using the below command line. This does not unload the driver.

   ```
   ./make_all clean
   ```

3. Issue the following command to make all archive targets and to load the driver.

```
./make_all
```

NOTE: After the device driver is built the script starts the driver. After building the API Library it is copied by the script to `/usr/lib/`. The script can also perform a clean operation by adding the term "`clean`" as a command line argument. A clean operation does not unload the driver. However, a clean does delete the API Library file copied to `/usr/lib/`.

2.8. Environment Variables

Some build environments may require compiler or linker options not present in the provided make files. To accommodate local environment specific requirements, the provided make files incorporate support for the following set of GSC specific environment variables.

2.8.1. GSC_API_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the API Library. The compiler used by the API Library make file is "`gcc`". The content of this environment variable is noted in the make file's output to the screen. The table below shows a portion of the screen output. The "`xxx`" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	`== Compiling: init.c`
	`== Compiling: ioctl.c`
	`== Compiling: open.c`

Defined and Not Empty	`== Compiling: init.c (added 'xxx')`
	`== Compiling: ioctl.c (added 'xxx')`
	`== Compiling: open.c (added 'xxx')`

2.8.2. GSC_API_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the API Library. The linker used by the API Library make file is "`ld`". The content of this environment variable is noted in the make file's output to the screen. The table below shows a portion of the screen output. The "`xxx`" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	`==== Linking: ../lib/libpex8112_api.so`
Defined and Not Empty	`==== Linking: ../lib/libpex8112_api.so (added 'xxx')`

2.8.3. GSC_LIB_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the utility libraries. The compiler used by the utility library make files is "`gcc`". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "`xxx`" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	`== Compiling: close.c`
	`== Compiling: init.c`
	`== Compiling: ioctl.c`

Defined and	`== Compiling: close.c (added 'xxx')`

Not Empty	`== Compiling: init.c (added 'xxx')` `== Compiling: ioctl.c (added 'xxx')`

2.8.4. GSC_LIB_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the utility libraries. The linker used by the utility library make files is "`ld`". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "`xxx`" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	`==== Linking: ../lib/pex8112_utils.a`
Defined and Not Empty	`==== Linking: ../lib/pex8112_utils.a (added 'xxx')`

2.8.5. GSC_APP_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the sample applications. The compiler used by the sample application make files is "`gcc`". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "`xxx`" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	`== Compiling: main.c` `== Compiling: perform.c`
Defined and Not Empty	`== Compiling: main.c (added 'xxx')` `== Compiling: perform.c (added 'xxx')`

2.8.6. GSC_APP_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the sample applications. The linker used by the sample application make files is "`gcc`". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "`xxx`" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	`==== Linking: id`
Defined and Not Empty	`==== Linking: id (added 'xxx')`

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing PEX8112 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 PEX8112 driver archive. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent PEX8112 specific header files. Therefore, sources may include only this one PEX8112 header and make files may reference only this one PEX8112 include directory.

Description	File	Location
Header File	`pex8112_main.h`	`…/include/`

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the PEX8112 driver archive. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other pertinent PEX8112 specific static libraries. Therefore, make files may reference only this one PEX8112 static library and only this one PEX8112 library directory.

Description	File	Location
Static Library	`pex8112_main.a`	`…/lib/`

> **NOTE**: The PEX8112 API Library is implemented as a shared library and is thus not linked with the PEX8112 Main Library. The API Library must be linked with applications either explicitly or by adding the argument `-lpex8112_api` to the linker command line.

3.2.1. Build

The main library is built via the Overall Make Script (section 2.7, page 11), but can be built separately following the below steps.

1. Change to the directory where the main library resides (`…/lib/`).

2. Remove existing build targets using the below command line.

   ```
   make clean
   ```

3. Rebuild the main library by issuing the below command.

   ```
   make
   ```

3.2.2. System Libraries

In addition to linking the static library named above, applications may need to also link in additional system libraries as noted below.

Library	gcc Link Flag
Math	`-lm`
POSIX Thread	`-lpthread`

Real Time	`-lrt`

4. API Library

The PEX8112 API Library is the software interface between user applications and the PEX8112 device driver. The interface is accessed by including the header file `pex8112_api.h`.

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

4.1. Files

The library source files are summarized in the table below.

File	Description
`api/*.c`	These are library source files.
`api/*.h`	These are library header files.
`api/makefile`	This is the library make file.
`api/makefile.dep`	This is an automatically generated make dependency file.
`include/pex8112_api.h`	This is the library interface header file.
`lib/libpex8112_api.so`	This is the API Library shared library file. *

* The shared library is automatically copied to `/usr/lib/` when it is built.

4.2. Build

The API Library is built via the Overall Make Script (section 2.7, page 11), but can be built separately following the below steps.

> **NOTE**: The API's shared library is copied to `/usr/lib/` when it is built. Therefore, these steps may require elevated privileges.

1. Change to the directory where the library sources reside (…`/api/`).

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

   ```
   make clean
   ```

3. Compile the source files and build the library by issuing the below command.

   ```
   make
   ```

4.3. Library Use

The library is used at application compile time, at application link time and at application run time. At compile time include the below listed header file in each source file using a component of the library interface. At link time include the below listed linker argument on the linker command line. At link time and at run time the library is found in the directory `/usr/lib/`. (The shared library file is automatically copied to `/usr/lib/` when the library is built.)

Description	File	Location	Linker Argument
Header File	`pex8112_api.h`	…`/include/`	
Shared Library	`libpex8112_api.so`	…`/lib/`	
		`/usr/lib/`	`-lpex8112_api`

4.4. Macros

The interface includes the following macros.

4.4.1. IOCTL

The IOCTL macros are documented in section 4.7 beginning on page 22.

4.4.2. Registers

The following gives the complete set of PEX8112 registers.

4.4.2.1. GSC Registers

The PEX8112 contains no GSC specific registers. Any GSC specific registers will be associated with the device using the PEX8112 to interface to a PCI Express bus.

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `gsc_pex8112.h`, which is automatically included via `pex8112.h`.

4.4.2.3. PLX Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pex8112.h`, which is automatically included via `pex8112.h`.

4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used.

4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A value of zero indicates success. A negative value indicates that the request could not be completed successfully. The specific value returned is the negative of the corresponding error status value taken from `errno.h`. I/O services return positive values to indicate the number of bytes successfully transferred.

4.6.1. pex8112_close()

This function is the entry point to close a connection to an open PEX8112 device. The board is put in an initialized state before this call returns.

Prototype

```
int pex8112_close(int fd);
```

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

Return Value	Description
`0`	The operation succeeded.

< 0	An error occurred. This is the negative of errno from errno.h.

Example

```
#include <stdio.h>

#include "pex8112_dsl.h"

int pex8112_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = pex8112_close(fd);

    if (ret)
        printf("ERROR: pex8112_close() returned %d\n", ret);

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

4.6.2. pex8112_init()

This function is the entry point to initializing the PEX8112 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 pex8112_init(void);
```

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. This is the negative of errno from errno.h.

Example

```
#include <stdio.h>

#include "pex8112_dsl.h"

int pex8112_init_dsl(void)
{
    int errs;
    int ret;

    ret = pex8112_init();

    if (ret)
        printf("ERROR: pex8112_init() returned %d\n", ret);

    errs    = ret ? 1 : 0;
```

```
    return(errs);
}
```

4.6.3. pex8112_ioctl()

This function is the entry point to performing setup and control operations on a PEX8112 device. 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 beginning on page 22.

Prototype

```
int pex8112_ioctl(int fd, int request, void* arg);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`request`	This specifies the desired operation to be performed.
`arg`	This is a `request` specific argument. Refer to the IOCTL services for additional information (section 4.7, page 22).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. This is the negative of `errno` from `errno.h`.

Example

```
#include <stdio.h>

#include "pex8112_dsl.h"

int pex8112_ioctl_dsl(int fd, int request, void *arg)
{
    int errs;
    int ret;

    ret = pex8112_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: pex8112_ioctl() returned %d\n", ret);

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

4.6.4. pex8112_open()

This function is the entry point to open a connection to a PEX8112 device. The device is initialized before the function returns.

Prototype

```
int pex8112_open(int device, int share, int* fd);
```

Argument	Description
device	This is the zero based index of the PEX8112 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.

* If the index value is -1, then the API Library accesses /proc/pex8112.

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. This is the negative of errno from errno.h.

Example

```
#include <stdio.h>

#include "pex8112_dsl.h"

int pex8112_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = pex8112_open(device, share, fd);

    if (ret)
        printf("ERROR: pex8112_open() returned %d\n", ret);

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

4.6.4.1. Access Modes

Shared Access Mode:

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

Exclusive Access Mode:

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

4.6.5. pex8112_read()

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

> **NOTE:** When performing an open on device index `-1`, the API Library accesses the `/proc/pex8112` text file. This read service then reads from that file. Refer to section 4.6.4, page 19.

> **NOTE:** The read service has no functionality for reading from PEX8112 devices. Attempts to read from PEX8112 devices will return an error.

Prototype

```
int pex8112_read(int fd, void *dst, size_t bytes);
```

Argument	Description
`fd`	This is the file descriptor of use for access.
`dst`	The data read will be put here.
`bytes`	This is the desired number of bytes to read.

Return Value	Description
`0` to `bytes`	The operation succeeded.
`< 0`	An error occurred. This is the negative of `errno` from `errno.h`.

Example

```
#include <stdio.h>

#include "pex8112_dsl.h"

int pex8112_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = pex8112_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: pex8112_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 PEX8112 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `pex8112_ioctl()` function arguments.

4.7.1. PEX8112_IOCTL_QUERY

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

Usage

Argument	Description
request	PEX8112_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
PEX8112_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
PEX8112_QUERY_DEVICE_TYPE	This returns the identifier value for the device's type. This should be GSC_DEV_TYPE_PEX8112.

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

Value	Description
PEX8112_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

4.7.2. PEX8112_IOCTL_REG_MOD

This service performs a read-modify-write of a PEX8112 register. At present there are registers that an application may modify. The PCI and PLX Feature Set Registers are read-only. Refer to `gsc_pex8112.h` for the complete list of accessible registers.

Usage

Argument	Description
request	PEX8112_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

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

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

	modified. If a bit here is zero, then the respective register bit is unmodified.

4.7.3. PEX8112_IOCTL_REG_READ

This service reads the value of a PEX8112 register. This includes the PCI registers and the PLX Feature Set Registers. Refer to `gsc_pex8112.h` for the complete list of accessible registers.

Usage

Argument	Description
request	PEX8112_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.

4.7.4. PEX8112_IOCTL_REG_WRITE

This service writes a value to a PEX8112 register. At present there are registers that an application may modify. The PCI and PLX Feature Set Registers are read-only. Refer to `gsc_pex8112.h` for a complete list of the accessible registers.

Usage

Argument	Description
request	PEX8112_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.

5. The Driver

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

5.1. Files

The device driver source files are summarized in the table below.

File	Description
`driver/*.c`	The driver source files.
`driver/*.h`	The driver header files.
`driver/pex8112.h`	This is the driver interface header file.
`driver/Makefile`	This is the driver make file.
`driver/start`	Shell script to install the driver executable and device nodes.

5.2. Build

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

The device driver is built via the Overall Make Script (section 2.7, page 11), but can be built separately following the below steps.

1. Change to the directory where the driver sources reside (`…/driver/`).

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

   ```
   make clean
   ```

3. Build the driver by issuing the below command.

   ```
   make
   ```

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

5.3. Startup

NOTE: The driver will have to be built before being used as it is provided in source form only.

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 devices identified by the driver.

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

1. Change to the directory where the driver sources reside (`…/driver/`).

2. 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.

   ```
   ./start
   ```

   NOTE: This script must be executed each time the host is rebooted.

   NOTE: The PEX8112 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 device detected.

3. Verify that the device driver module has been loaded by issuing the below command and examining the output. The module name `pex8112` should be included in the output.

   ```
   lsmod
   ```

4. 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 device detected.

   ```
   ls –l /dev/pex8112.*
   ```

5.3.2. 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. The example is based on the driver being installed in `/usr/src/linux/drivers/`, though it may have been installed elsewhere.

   ```
   /usr/src/linux/drivers/pex8112/driver/start
   ```

   NOTE: For `systemd` installations the file `rc.local` may be located under the `/etc/` directory rather than under `/etc/rc.d/`.

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. Do this by following the verification steps given in the manual startup procedures.

5.3.2.1. File `rc.local` Not Present

Some distributions may not install a default version of `rc.local`. Some may not even create the directory `/etc/rc.d/`. If the directory is not present, then it may be created. The directory must be created with the owner and group set to `root`. The directory permissions must be set to `rwxr-xr-x`. If the file `/etc/rc.d/rc.local` is not present, then it too may be created. The file must also be created with the owner and group set to `root`. Additionally, the file permissions must also be set to `rwxr-xr-x`. After the directory and file are created as described, reboot to verify boot time loading of the driver. Here is an example of a default version of `rc.local`.

```
#!/bin/bash

# Add you local content here.
```

5.3.2.2. Default rc.local File Permissions

The rc.local script may fail to run at boot time because some distributions install a default version of the file without execute permissions. Without execute permissions, boot time invocation of the script fails, which inhibits boot time loading of the driver. If this is the case, then change the file permissions to rwxr-xr-x. After the file permissions are adjusted as described, reboot to verify boot time loading of the driver.

5.3.2.3. systemd Installations

With the advent of the systemd startup implementation, rc.local may be accessed via a systemd startup service. The service name may be rc-local, rc-local.service or something similar. This service may or may not be enabled by default. If the service is disabled, then the script will not execute, which prevents boot time loading of the driver. The service can be enabled with the below command line. After the service is enabled, reboot to verify boot time loading of the driver.

```
systemctl enable rc-local
```

NOTE: For systemd installations the file rc.local may be located under the /etc/ directory rather than under /etc/rc.d/.

5.3.2.4. systemd and rc.local Timing

If the above steps have been performed but the driver still does not start then examine the dmesg output for driver messages. If the output shows that the driver starts and immediately stops, then the problem may be timing. That is, since systemd doesn't serialize startup initialization as done in the past, driver loading may fail if required services have not completed their own initialization. If this is the problem, then it may be corrected simply by inserting a delay in rc.local prior to it calling the driver's start script (i.e. sleep for one or more seconds).

5.3.2.5. SElinux Implications

If not disabled, then SElinux may prevent boot time loading of the driver. If this is the case, then it can be verified and corrected using SElinux related tools and utilities. First, install the necessary software using the below command. (As necessary, replace the yum command line with that which is available for your distribution.)

```
yum install setroubleshoot setools
```

Next, run the below command to determine if SElinux is preventing the driver from loading at boot time.

```
sealert –a /var/log/audit/audit.log
```

If SElinux is preventing the driver from loading, then the output from the above command should include a reference to the driver's start script, the insmod command that loads the driver or the name of the driver executable. If so, then the output should also indicate the commands necessary to resolve the issue. The following is an example of the instructions given when the culprit is insmod, which is the start script command that loads the driver. After running these commands reboot the system to verify boot time loading of the driver.

```
ausearch -c 'insmod' --raw | audit2allow -M my-insmod
semodule -X 300 -i my-insmod.pp
```

5.4. Verification

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

1. Verify that the file `/proc/pex8112` is present. If the file is present then the driver is loaded and running. Verify the file's presence by viewing its content with the below command.

   ```
   cat /proc/pex8112
   ```

5.5. 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 places such as `/var/log/messages`). It is reported in the text file `/proc/pex8112` while the driver is loaded and running.

5.6. Shutdown

Shutdown the driver following the below listed steps.

> **NOTE**: The following steps may require elevated privileges.

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

   ```
   rmmod pex8112
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `pex8112` should not be in the listed output.

   ```
   lsmod
   ```

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.

File	Description
docsrc/*.c	These are the C source files.
docsrc/makefile	This is the library make file.
docsrc/makefile.dep	This is an automatically generated make dependency file.
include/pex8112_dsl.h	This is the primary utility header file.
lib/pex8112_dsl.a	This is the statically linkable library file.

6.2. Build

The library is built via the Overall Make Script (section 2.7, page 11), but can be built separately following the below steps.

1. Change to the directory where the documentation sources reside (…/docsrc/).

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

   ```
   make clean
   ```

3. Compile the sample files and build the library by issuing the below command.

   ```
   make
   ```

6.3. Library Use

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

Description	File	Location
Header File	pex8112_dsl.h	…/include/
Static Link Library	pex8112_dsl.a	…/lib/

7. Utility Source Code

The driver archive includes a body of utility services built into a statically linkable library that is usable with console applications. The primary purpose of the services is both for code reuse in the sample applications and to provide wrappers, mostly visual, around the driver's IOCTL services. The aim of the visual wrappers is to facilitate structured console output for the sample applications. 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.

7.1. Files

The library files are summarized in the table below.

File	Description
utils/util_*.c	These are device specific utility source files.
utils/gsc_*.c	These are device and OS independent utility source files.
utils/os_*.c	These are OS specific utility source files.
utils/makefile	This is the library make file.
utils/makefile.dep	This is an automatically generated make dependency file.
include/pex8112_utils.h	This is the primary utility header file.
lib/pex8112_utils.a	This is the statically linkable library file.

7.2. Build

The library is built via the Overall Make Script (section 2.7, page 11), but can be built separately following the below steps.

1. Change to the directory where the utility sources reside (…/utils/).

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

   ```
   make clean
   ```

3. Compile the sample files and build the library by issuing the below command.

   ```
   make
   ```

7.3. Library Use

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

Description	File	Location
Header File	pex8112_utils.h	…/include/
Static Link Library	pex8112_utils.a	…/lib/

8. Operating Information

This section explains some basic operational procedures for using the PEX8112. 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
Application	`id`	`…/id/`

9. Sample Applications

The driver archive includes a variety of sample and test applications. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 11), but each may be built individually by changing to its respective directory and issuing the commands "`make clean`" and "`make all`". The initial output from each application includes information on its supported command line arguments. The following gives a brief overview of each application.

9.1. id - Identify Device - …/id/

This application reports detailed device identification information. This can be used with tech support to help identify as much technical information about the device as possible from software.

9.2. regs - Register Access - …/regs/

This application provides menu based interactive access to the device's registers, and reports other pertinent information to the console.

Document History

Revision	Description
October 11, 2022	Updated to version 2.3.101.43.0. Expanded automatic startup information. Updated the kernel support table. Added section on environment variables. Updated the information for the open and close calls.
January 13, 2021	Updated to version 2.2.93.35.0. Updated the inside cover page. Updated the CPU and kernel support section. Numerous editorial changes. Added a licensing subsection. Expanded automatic startup information. Document reorganization.
December 7, 2016	Updated to version 2.1.69.18.0. Removed the `built` field from the `/proc/` file. Updated the kernel support table. Updated material on the open call. Added open access mode descriptions. Added a section for general operating information. Made various miscellaneous updates. Some document reorganization.
September 16, 2015	Updated to version 2.0.60.8.0. Updated the device node name to include a period before the device index. Removed double underscore that prefaced various data types.
February 28, 2014	Updated to version 1.2.52.0. Updated the kernel support data.
January 9, 2014	Updated to version 1.1.51.0. Updated the kernel support data.
November 13, 2013	Updated to version 1.1.49.0.
July 18, 2013	Updated to version 1.1.45.0. Updated the kernel support data.
July 24, 2012	Updated to version 1.1.39.0. Updated the kernel support data.
December 28, 2011	Initial release, version 1.0.34.0.