HPDI32

High Performance 32-bit Digital I/O

PCI-HPDI32A
PCI64-HPDI32AL
PMC-HPDI32A
PMC64-HPDI32ALT

Software Development Kit
SDK 6.2.0 Setup Guide
For Linux

Manual Revision: October 7, 2014
Version 6.2.0

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

This setup guide applies to HPDI32 SDK release version 6.2.0. This release of the setup guide is used with HPDI32 SDK 6.2.0 Reference Manual.

1.1. Purpose

The purpose of this document is to describe installation, setup and use of the HPDI32 Software Development Kit under Linux kernel versions 3.x, 2.6 and 2.4.

1.2. Acronyms

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

Acronyms	Description
API	Application Programming Interface (This is sometimes used synonymously with SDK or API Library.)
DMA	Direct Memory Access
GPIO	General Purpose Input/Output
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
SDK	Software Development Kit (This is sometimes used synonymously with API or API Library.)

1.3. Definitions

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

Term	Definition
API Buffer	A physically contiguous block of memory allocated via the API.
API Library	This refers to the library implementing the application level HPDI32 interface. (This is sometimes used synonymously with SDK or API.)
Application	This refers to user mode processes.
Application Buffers	These are memory buffers allocated and maintained entirely by the application which are used for reading data from and writing data to the HPDI32's memory.
Device Driver	This refers to the driver executable component of the HPDI32 SDK.
Driver	This refers to the device driver, which runs under control of the operating system.
Rx	For I/O operations this refers to reading data from the HPDI32. Otherwise it refers to the reception of data over the cable interface, either into the Rx FIFOs or the GPIO input ports.
Tx	For I/O operations this refers to writing data to the HPDI32. Otherwise it refers to the transmission of data over the cable interface, either from the Tx FIFOs or the GPIO output ports.

1.4. Hardware Overview

The HPDI32 is a high-performance 32-bit parallel digital I/O interface board. The host side connection is PCI based and is either 32-bit or 64-bit according to the model ordered. The external I/O interface varies per model ordered. The board is capable of transmitting or receiving data at up to 200 Mbytes per second over the external I/O interface, depending on the model ordered. Onboard transmit and receive FIFOs of up to 128k data values each, buffer transfer data between the PCI bus and the cable interface. This allows the HPDI32 to maintain maximum bursts on the cable interface (at least up to the depth of the FIFOs) independent of the PCI bus interface. The onboard FIFOs can also be used to buffer data between the cable interface and the PCI bus to maintain a sustained data throughput for real-time applications.

The HPDI32 offers a half-duplex external I/O interface. The board can either transmit or receive data, but it cannot do both simultaneously. In addition to the 32 synchronous data I/O lines, the external interface includes a set of configurable flow control signals. Some of these can also be configured as discrete I/O. The board accommodates a wide range of applications. This range extends from sending or receiving relatively small blocks of data on demand, to sending or receiving large continuous streams of data for an extended period. Once a data link is established, the data is transferred to/from host memory by simply writing to or reading from the onboard FIFOs. The board has an advanced PCI interface engine, which provides for increased data throughput via DMA.

1.5. Software Overview

The software interface to the HPDI32 consists of a Device Driver and an API Library; the primary components of the SDK. The Device Driver operates under control of the operating system and must be loaded and running in order to access any installed HPDI32 devices. The interface provided by the API Library is based on the board's functionality and is organized around the HPDI32's set of main hardware features. Refer to the SDK Reference Manual for additional information.

1.5.1. Software Architecture

An application communicates with a HPDI32 using the two components described briefly above. Any number of applications may make simultaneous use of the library and each use is totally independent, unless specifically designed to do otherwise. Each instance provides access to at most 32 different HPDI32 devices. The diagram below describes the components and how they fit together.

Application	This is any application written to communicate with one or more HPDI32 devices using the HPDI32 API Library.
API Library	This library presents a HPDI32 feature based interface to applications wishing to communicate with HPDI32 devices.
Device Driver	The driver provides access to HPDI32 devices.
HPDI32	This refers to any number of installed HPDI32 devices.

1.6. Compatibility

The API Library interface presented in this version of the SDK is identical for kernel versions 3.x, 2.6 and 2.4. In addition, it is also identical for those operating systems for which this version of the SDK has been ported. Compatibility is based on the individual numbers in the overall version number. The first number changes when there are dramatic changes in the interface that will necessitate application porting. The second number changes when there are minor changes to the interface, including additional services. In this case, applications may be able to use the SDK as is. If not, the application may have to be recompiled. Changes in the third number refer to changes in the release that have to do with the support files or other issues.

1.7. Requirements

1.7.1. Kernel Sources

Building the driver requires installation of the kernel sources. Numerous driver build errors will appear if the kernel sources are not installed.

1.8. Limitations and Restrictions

1.8.1. 64-bit DMA Not Supported

The HPDI32 can perform DMA to 32-bit physical addresses only. On installations with sufficient memory, Linux may grant an application memory in which the physical address of some pages requires more than a 32-bit address to access. When this occurs the I/O request will fail without any data having been transferred. To prevent this from occurring, applications can either use PIO mode data transfers or they can use API Buffers. Refer to the Reference Manual for additional information.

1.9. Reference Material

The following reference material may be of particular benefit in using the HPDI32 and this SDK. The specifications provide the information necessary for an in-depth understanding of the specialized features implemented on this board.

- The *HPDI32 Software Development Kit Reference Manual* from General Standards Corporation.
- The applicable *HPDI32 User Manual* from General Standards Corporation.
- The *PCI9080 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc. (for 32-bit PCI interface boards) *
- The *PCI9656 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc. (for 64-bit PCI interface boards) *

* 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/

2. Installation

2.1. CPU and Kernel Support

The SDK is designed to operate with Linux kernel versions 2.6 and 2.4 running on a PC system with one or more Intel x86 processors. This release of the SDK was tested under the below listed kernels.

Kernel	**Distribution**	**x86**	
		32-bit	**64-bit**
3.11.10	Red Hat Fedora Core 20	Yes	Yes
3.9.5	Red Hat Fedora Core 19	Yes	Yes
3.6.10	Red Hat Fedora Core 18	Yes	Yes
3.3.4	Red Hat Fedora Core 17	Yes	Yes
3.1.0	Red Hat Fedora Core 16	Yes	Yes
2.6.38	Red Hat Fedora Core 15	Yes	Yes
2.6.35	Red Hat Fedora Core 14	Yes	Yes
2.6.33	Red Hat Fedora Core 13	Yes	Yes
2.6.31	Red Hat Fedora Core 12	Yes	Yes
2.6.29	Red Hat Fedora Core 11	Yes	Yes
2.6.27	Red Hat Fedora Core 10	Yes	Yes
2.6.25	Red Hat Fedora Core 9	Yes	Yes
2.6.23	Red Hat Fedora Core 8	Yes	Yes
2.6.21	Red Hat Fedora Core 7	Yes	Yes
2.6.18	Red Hat Fedora Core 6	Yes	Yes
2.6.15	Red Hat Fedora Core 5	Yes	Yes
2.6.11	Red Hat Fedora Core 4	Yes	Yes
2.6.9	Red Hat Fedora Core 3	Yes	Yes
2.4.21	Red Hat Enterprise Linux Workstation Release 3	Yes	
2.4.18	Red Hat Linux 7.3	Yes	

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

NOTE: The driver and the API Library will have to be built before they can be used as they are 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.

NOTE: Sources for support of the 2.2 kernel are included in the release, but it is unsupported and untested.

2.2. The /proc File System

While the driver is loaded, the text file `/proc/hpdi32` can be read to obtain information from 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: 7.2.3.4.2
built: Oct  7 2014, 15:07:22
boards: 1
```

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

2.3. Release Files

This release consists of the below listed files. The archive is described in detail in following subsections. (The file names may at times include the SDK version number.)

File	Description
hpdi32_sdk_linux.tar.gz	This archive contains the entire set of SDK files, including the reference manual and the setup guide.

2.4. Directory Structure

The following table describes the basic directory structure utilized by the HPDI32 SDK. During installation the directory structure is created and populated with the DSK files.

> **NOTE**: As of SDK release 6.1.0 the archive directory structure has been reorganized. All files are contained under one primary tree (hpdi32) rather than two (hpdi32/common and hpdi32/hpdi32).

Directory Structure	Content
hpdi32	This is the SDK's root directory.
hpdi32/api	This directory contains the sources for the HPDI32 API Library.
hpdi32/docsrc	This directory contains the sources included in this user manual. Refer to section 7 beginning on page 20.
hpdi32/driver	This directory contains the sources for the HPDI32 device driver. Refer to section 5 on page 16.
hpdi32/gpio/	This subdirectory contains the GPIO Library and its utility sources.
hpdi32/gpio/din	This directory contains the sources for the GPIO Digital Input sample application (section 8.7, page 28).
hpdi32/gpio/lib	This directory contains the sources for the GPIO Library. For detailed information refer to the GPIO Library Reference Manual.
hpdi32/gpio/dout	This directory contains the sources for the GPIO Digital Output sample application (section 8.8, page 29).
hpdi32/gpio/utils	This directory contains the sources for the GPIO Library utility library. For detailed information refer to the GPIO Library Reference Manual.
hpdi32/jumpers	This directory contains the sources for the User Jumpers sample application. Refer to section 8.1 on page 21.
hpdi32/rthrottle	This directory contains the sources for the Remote Throttle sample application. Refer to section 8.2 on page 22.
hpdi32/rx	This directory contains the sources for the Receive sample application. Refer to section 8.3 on page 24.
hpdi32/sbtest	This directory contains the sources for the Single Board test application. Refer to section 8.4 on page 25.
hpdi32/hpdi32/tx	This directory contains the sources for the Transmit sample application. Refer to section 8.5 on page 26.
hpdi32/utils	This directory contains the sources for utility routines used by the sample and test applications included in the driver archive.
hpdi32/xfer	This directory contains the sources for the Data Transfer sample application. Refer to section 8.6 on page 27.

2.5. Installation Procedure

Install the SDK and all of its files following the below listed steps. This places all SDK files under a single HPDI32 directory, which includes the driver, the API Library and all related support files. The directory structure is described briefly above. For specific information on building and loading the driver refer to section 5 on page 16. For specific information on building and installing the API Library refer to section 6 on page 19.

> **NOTE:** This procedure requires root privileges in order to start the driver and to install the API's shared library.

1. In a command shell, change to the directory where the SDK is to be installed. This may `/usr/src/linux/drivers`, though the path name may vary among distributions and kernel versions.

2. Copy the archive file `hpdi32_sdk_linux.tar.gz` into the current directory. (The file name may include the SDK version number.)

3. Issue the following command to decompress and extract the files from the provided archive. This creates the directory `hpdi32` in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzf hpdi32_sdk_linux.tar.gz
```

2.6. 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 SDK. It will also start the driver and install the SDK's shared library. The script is named `make_all`. It is recommended that this script be executed right after installing the SDK. Execute the script following the below steps. For individual instructions on building and loading the driver refer to section 4 on page 15. For individual instructions on building and installing the API Library refer to section 6 on page 19.

> **NOTE:** This procedure requires root privileges in order to start the driver and to install the API's shared library.

1. In a command shell, change to the SDK's root directory, which may be `/usr/src/linux/drivers/hpdi32`.

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

```
./make_all clean
```

3. Perform a build of all SDK targets by issuing the below command. This will also start the driver and install the API shared library.

```
./make_all all
```

> **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.

2.7. Version Numbers

The table below lists the version numbers for the SDK and its major components.

Component	File Name	Version
SDK	hpdi32_sdk_linux.tar.gz	6.2.0
Device Driver	hpdi32.ko	7.2.3.4.2 (2.4 kernels use .o)
API Library	libhpdi32_api.so.9.1.8.2.5	9.1.8.2.5

2.7.1. SDK Version Number

The SDK version number is maintained independent of and is generally unrelated to the version numbers for the device driver and the interface library. The SDK version number is found only in the documentation, though it may at times be included in the SDK archive or user manual file names.

2.7.2. Device Driver Version Number

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/hpdi32`. It can also be read by an application via the `hpdi32_version_get()` call with the `GSC_VERSION_DRIVER` identifier, or the utility macro service `HPDI32_VERSION_GET_DRIVER()`.

2.7.3. API Library Version Number

The library version number can be obtained in a variety of ways. It is included in the file's name as the five dot separated numbers. The file can be found in the `hpdi32/hpdi32/api/linux/gcc` subdirectory under the directory where the SDK was installed. After installation it can be found in the `/usr/local/lib` directory. It can also be read by an application via the `hpdi32_version_get()` call with the `GSC_VERSION_LIBRARY` identifier, or the utility macro service `HPDI32_VERSION_GET_LIBRARY()`.

3. Removal

Follow the below steps to remove the SDK. This includes the device driver, the interface library and all related support files.

> **NOTE:** This procedure requires root privileges in order to stop the driver, remove the device nodes and uninstall the API's shared library.

1. Shutdown the driver as described in section 5.1.4 on page 17.

2. In a command shell, change to the SDK's parent directory, which may be `/usr/src/linux/drivers`.

3. Issue the below command to remove the SDK. (The archive file name may include the SDK version number.)

   ```
   rm -rf hpdi32_sdk_linux.tar.gz hpdi32
   ```

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

   ```
   rm -rf /dev/hpdi32*
   ```

5. If the automated startup procedure was adopted (refer to section 5.1.2.2 on page 17), then edit the system startup script `rc.local` and remove the line that invokes the SDK's `make_all` script. The file `rc.local` should be located in the `/etc/rc.d` directory.

6. Remove the HPDI32 shared library and its references from `/usr/local/lib`. Do this by executing the below commands.

   ```
   rm -rf /usr/local/lib/*hpdi32*
   ldconfig
   ```

4. Using the SDK

4.1. Multithreaded

The API Library is multithreaded. This may require that applications and libraries using the API also be built for multithreaded operation. Difficult to identify bugs may appear when the API Library is used with applications built for single threaded only operation.

4.2. Compile Time Use

Compile time use has three requirements. First, include the header file `hpdi32_api.h` in each module referencing an API component. Second, configure the compiler for multithreaded operation, as applicable. And last, expand the include file search path to search the directories where the library header and its included files are located. This includes the following SDK directories.

Directory
`hpdi32/api`

4.3. Link Time Use

Link time use has two requirements. The first requirement is to insure that the API Library has been installed. This permits the library to be automatically located by the linker. For additional information refer to section 6 on page 19. The second requirement is to link applications with both the API Library and the `pthread` library. This can typically be done by adding the arguments `-lhpdi32_api` and `-lpthread` to the linker command line. The API Library argument should usually appear before the `pthread` argument.

4.4. Run Time Use

There are two run time requirements. The first is to insure that the driver has been built and loaded. This permits HPDI32 devices to be accessed. For additional information refer to section 5 on page 16. The second requirement is to insure that the API Library has been installed. This permits the library to be automatically located at run time. For additional information refer to section 6 on page 19.

5. Driver

The driver sources are contained in the installed archive. The sources used to build the driver are distributed among a number of the subdirectories as described briefly in the directory structure table of section 2.4 on page 11. The instruction for building, loading and shutting down the driver are given in the paragraphs that follow.

5.1.1. Build

Follow the below steps to build the driver.

1. In a command shell, change to the SDK's driver directory (…`/hpdi32/driver`).
2. Remove all existing build targets by issuing the below command.

   ```
   make clean
   ```

3. Build the driver by issuing the below command.

   ```
   make all
   ```

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 correctable by modifying the makefile.

5.1.2. Startup

The startup script performs all the tasks needed to load the driver and prepare it for use. This includes unloading the current driver, if one is loaded, and removing any existing device nodes. It also includes loading the driver and creating device nodes for each installed device. The recreation of new device nodes is done to insure that the device node major number is synchronized with the newly loaded driver since the major number is assigned dynamically by the kernel when the driver is loaded.

5.1.2.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: Starting the driver requires root privileges.

1. In a command shell, change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`.
2. Change to the subdirectory containing the startup script, as shown in the below command line.

   ```
   cd hpdi32/driver
   ```

3. Load 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: For manual startup above must be repeated each time the host is rebooted.

NOTE: The HPDI32 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 driver has been loaded by issuing the below command and examining the output. The module name `hpdi32` 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/hpdi32*
   ```

5.1.2.2. Automatic Driver Startup Procedures

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

NOTE: These steps require root privileges.

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 `/usr/src/linux/drivers` portion of the path should be replaced with the directory where the HPDI32 SDK archive was installed.

   ```
   /usr/src/linux/drivers/hpdi32/driver/start
   ```

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

3. Verify that the device driver has been loaded by issuing the below command and examining the output. The module name `hpdi32` 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 installed board.

   ```
   ls -l /dev/hpdi32*
   ```

5.1.3. Verification

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

1. Issue the below command to see if the driver's `/proc` file entry is present.

   ```
   ls /proc/hpdi32
   ```

   If the file is found, then the driver is loaded. If the file is not found, then the driver is not loaded

5.1.4. Shutdown

Shutdown the driver following the below listed steps.

NOTE: These steps require root privileges.

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

   ```
   rmmod hpdi32
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `hpdi32` should not be in the output generated.

   ```
   lsmod
   ```

6. API Library

The API Library sources are contained in the installed archive. The sources used to build the library are distributed among a number of subdirectories as described briefly in the directory structure table of section 2.4 on page 11. The instruction for building, installing and removing the API Library are given in the paragraphs that follow.

6.1. Build

Follow the below steps to build the API Library.

1. In a command shell, change to the SDK's API Library directory (…`/hpdi32/api`).
2. Remove all existing build targets by issuing the below command.

   ```
   make clean
   ```

3. Build the API Library by issuing the below command.

   ```
   make all
   ```

4. Build the API Library install script by issuing the below command.

   ```
   make install
   ```

6.2. Install

Install the shared library for the SDK's API by following the below listed steps.

NOTE: Installing the shared library requires root privileges.

1. In a command shell, change to the SDK's API Library directory (…`/hpdi32/api`).
2. Install the library by executing the below command.

   ```
   ./install
   ```

   When the script completes the library should be installed. The installation should not report any errors.

3. Verify that the library file and soft links are in place by issuing the below command and examining the output. The output should include the library file itself, and two soft links to the library file. The name of one soft link should be `libhpdi32_api.so`. The name of the second soft link should be `libhpdi32_api.so.6`.

   ```
   ls -l /usr/local/lib/
   ```

4. Verify that the library has been installed by issuing the below command and examining the output. The output should include two entries with names identical to the soft links identified in the previous step.

   ```
   ldconfig -p | grep hpdi32
   ```

7. Documentation Source Code Library

All source code examples from the SDK reference manual are provided as C source files and are installed as part of the installation process. In addition, they are included as a statically linkable library usable with HPDI32 applications. The purpose of the source code 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 paragraphs that follow give additional information relating to these files. The installed files are summarized in the below table and are installed under the HPDI32 `hpdi32/docsrc` directory. The sources include a makefile to build the library file `hpdi32_dsl.a`.

File	Description
`docsrc\*.c`	These are the C source files.
`docsrc\hpdi32_dsl.h`	This is the library header file.
`docsrc\makefile`	This is a `gcc` makefile.
`docsrc\makefile.dep`	This is an automatically generated make dependency file.

7.1. hpdi32_dsl.a

The build target for these sample files is the statically linkable library `hpdi32_dsl.a`. This library can be used with customer applications as is if so desired. These files also provide building blocks upon which HPDI32 applications can be built and function as aids to ease the learning curve and reduce application development time.

7.2. Build

Follow the below steps to compile the source files and build the library.

1. In a command shell, change to the SDK's API Library directory (…`/hpdi32/api`).

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

   ```
   make clean
   ```

3. Build the library by issuing the below command.

   ```
   make all
   ```

7.3. Using the Library

7.3.1. Compile Time Use

Compile time use has two requirements. First, include the header file `hpdi32_dsl.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 is the subdirectory `hpdi32/docsrc` under the directory tree where the SDK was installed.

7.3.2. Link Time Use

Link time use also has two requirements. First, include the static library `hpdi32_dsl.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 is the subdirectory `hpdi32/docsrc` under the directory tree where the SDK was installed.

8. Sample Applications

8.1. jumpers – User Jumpers

This sample console application reports user jumper information. This is to assist users in identifying boards whose user configurable jumpers are used to distinguish one HPDI32 from another. The application includes the below listed files.

File	Description
`jumpers/*.c`	These are the application's source files.
`jumpers/main.h`	This is the application's header file.
`jumpers/makefile`	This is a makefile.
`jumpers/makefile.dep`	This is an automatically generated make dependency file.
`utils/*`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.1.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/jumpers`).

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.1.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/jumpers`).

2. Start the sample application by issuing the command given below. The command line arguments are described in the table below. A single iteration should complete in less than a second. Statistics are reported at the conclusion of each iteration.

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

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.

8.2. rthrottle – Remote Throttle

This sample console application is designed to transfer data between two HPDI32 boards connected by a strait passthrough cable. The primary purpose is to demonstrate the boards' capability of throttling data transfer in those instances where the transmit device outperforms the receive device. The application can, however, be used drive a single HPDI32 connected by a cable to some other device. The application includes the below listed files.

File	Description
`rthrottle/*.c`	These are the application's source files.
`rthrottle/main.h`	This is the application's header file.
`rthrottle/makefile`	This is a makefile.
`rthrottle/makefile.dep`	This is an automatically generated make dependency file.
`utils/*`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.2.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/rthrottle`).

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.2.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/rthrottle`).

2. Start the sample application by issuing the command given below. The command line arguments are described in the table below. Statistics are reported at the conclusion of each iteration.

   ```
   ./rthrottle [tx|rx] <-c> <-C> <-dma> <-dmdma> <-m#> <-n#> <-pio>
               <-q#> <-r-> <-s> <-s#> <index>
   ```

Argument	Description
`tx | rx`	This required argument indicates whether data is to be transmitted or received.
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-dma`	This specifies that I/O operations use DMA mode data transfer.
`-dmdma`	This specifies that I/O operations use Demand Mode DMA mode data transfer.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`-pio`	This specifies that I/O operations use PIO mode data transfer.
`-q#`	Block Quantity: This specifies the number of data blocks to be transferred.
`-r-`	Disable the Remote Throttle hardware flow control feature.

`-s`	Save the data to a file. For transmit operations, the data for a single block is written to `tx_data.txt`. For receive operations, all data received is written to `rx_data.txt`.
`-s#`	Block Size: This specifies the number of bytes in each block to be transferred. This should be a multiple of four bytes.
`index`	This is the zero based index of the board to access.

8.3. rx - Receive

This sample console application receives a block of data from the cable interface. This application makes no assumptions about a cable being attached. The application includes the below listed files.

File	Description
`rx/*.c`	These are the application's source files.
`rx/main.h`	This is the application's header file.
`rx/makefile`	This is a makefile.
`rx/makefile.dep`	This is an automatically generated make dependency file.
`utils/*`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.3.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/rx`).

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.3.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/rx`).

2. Start the sample application by issuing the command given below. The command line arguments are described in the table below. A single iteration should complete in just a few seconds. Statistics are reported at the conclusion of each iteration.

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

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.

8.4. sbtest - Single Board Test

This sample console application performs a series of automated tests of an HPDI32, the API Library and the driver. The application includes the below listed files.

File	Description
`sbtest/*.c`	These are the application's source files.
`sbtest/main.h`	This is the application's header file.
`sbtest/makefile`	This is a makefile.
`sbtest/makefile.dep`	This is an automatically generated make dependency file.
`utils/*`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.4.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/sbtest`).

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.4.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/sbtest`).

2. Start the sample application by issuing the command given below. The command line arguments are described in the table below. A single iteration should complete in less than two minutes. Statistics are reported at the conclusion of each iteration.

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

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.

8.5. tx - Transmit

This sample console application transmits a block of data out the cable connection. This application makes no assumptions about a cable being attached. The data is sent blindly. The application includes the below listed files.

File	Description
`tx/*.c`	These are the application's source files.
`tx/main.h`	This is the application's header file.
`tx/makefile`	This is a makefile.
`tx/makefile.dep`	This is an automatically generated make dependency file.
`utils/*.c`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.5.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/tx`).
2. Remove all existing build targets by issuing the below command.

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.5.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/tx`).
2. Start the sample application by issuing the command given below. The command line arguments are described in the table below. A single iteration should complete in just a few seconds. Statistics are reported at the conclusion of each iteration.

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

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.

8.6. xfer - Transfer

This sample console utility either transmits or receives a block of data according to command line arguments and a configuration file. (See the file readme.txt for information on the configuration file format.) The application includes the below listed files.

File	Description
`xfer/*.c`	These are the application's source files.
`xfer/main.h`	This is the application's header file.
`xfer/makefile`	This is a makefile.
`xfer/makefile.dep`	This is an automatically generated make dependency file.
`utils/*.c`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.6.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/xfer`).

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.6.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/xfer`).

2. Start the sample application by issuing the command given below. The command line arguments are described in the table below. A single iteration should complete in just a few seconds. Statistics are reported at the conclusion of each iteration.

   ```
   ./xfer <-c> <-C> <-m#> <-n#> <index> <direction> <file>
   ```

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.
`direction`	This must be either "`tx`" or "`rx`" to specify transmission or reception, respectively.
`file`	This is the name of the configuration file to use in configuring the HPDI32 for the requested operation.

8.7. gpio/din – GPIO Digital Input

This sample console configures the board for GPIO input then reports the state of the GPIO inputs for a designated period of time. This includes the GPIO signals based on both the seven control signals and the 32 data signals. When the input state changes the results are reported to the console. The application includes the below listed files.

File	Description
`gpio/din/*.c`	These are the application's source files.
`gpio/din/main.h`	This is the application's header file.
`gpio/din/makefile`	This is a makefile.
`gpio/din/makefile.dep`	This is an automatically generated make dependency file.
`utils/*.c`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.7.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/gpio/din`).
2. Remove all existing build targets by issuing the below command.

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.7.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/gpio/din`).
2. Start the sample application by issuing the command given below. With the default arguments a single iteration should complete in about 10 seconds. The command line arguments are described in the table below.

   ```
   ./din <-c> <-C> <-m#> <-n#> <-s#> <index>
   ```

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`-s#`	This specifies the number of seconds that the GPIO inputs are monitored, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.

8.8. gpio/dout – GPIO Digital Output

This sample console configures the board for GPIO output then posts a changing pattern to GPIO outputs. This includes the GPIO signals based on both the seven control signals and the 32 data signals. The pattern is posted to the console as it is being posted to the cable interface. The application includes the below listed files.

File	Description
`gpio/dout/*.c`	These are the application's source files.
`gpio/dout/main.h`	This is the application's header file.
`gpio/dout/makefile`	This is a makefile.
`gpio/dout/makefile.dep`	This is an automatically generated make dependency file.
`utils/*.c`	These are utility sources used by the application.
`docsrc/*`	These are utility sources used by the application.

8.8.1. Build

Follow the below steps to build the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/gpio/dout`).

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

8.8.2. Execute

Follow the below steps to execute the sample application.

1. In a command shell, change to the application's source directory (…`/hpdi32/gpio/dout`).

2. Start the sample application by issuing the command given below. With the default arguments a single iteration should complete in about 12 seconds. The command line arguments are described in the table below.

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

Argument	Description
`-c`	This will cause the operation to repeat until an error is encountered.
`-C`	This will cause the operation to repeat even after an error is encountered.
`-m#`	This will cause the operation to repeat for at most "#" minutes, where "#" is a decimal number.
`-n#`	This will cause the operation to repeat at most "#" times, where "#" is a decimal number.
`index`	This is the zero based index of the board to access.

8.9. Windows Only

The Windows distribution of the HPDI32 SDK includes additional sample applications. The purpose of these applications is to help reduce one's learning curve and development time with HPDI32 applications. Linux users may benefit from using these applications. Please consider obtaining and installing the Windows version of the SDK and exercising these applications in order to gain these potential benefits.

8.9.1. Visual HPDI32 (Windows Only)

This sample GUI application, *Visual HPDI32*, gives a visual representation of an HPDI32 and permits one to interactively configure and exercise any board in the system. The application includes both a Configuration Wizard and a C Source Code Generator. The application also permits writing data to the board, reading data from the board, monitoring the Cable Control/GPIO lines, as well writing to the cable's GPIO lines. Refer to Figure 1 below for a screen shot.

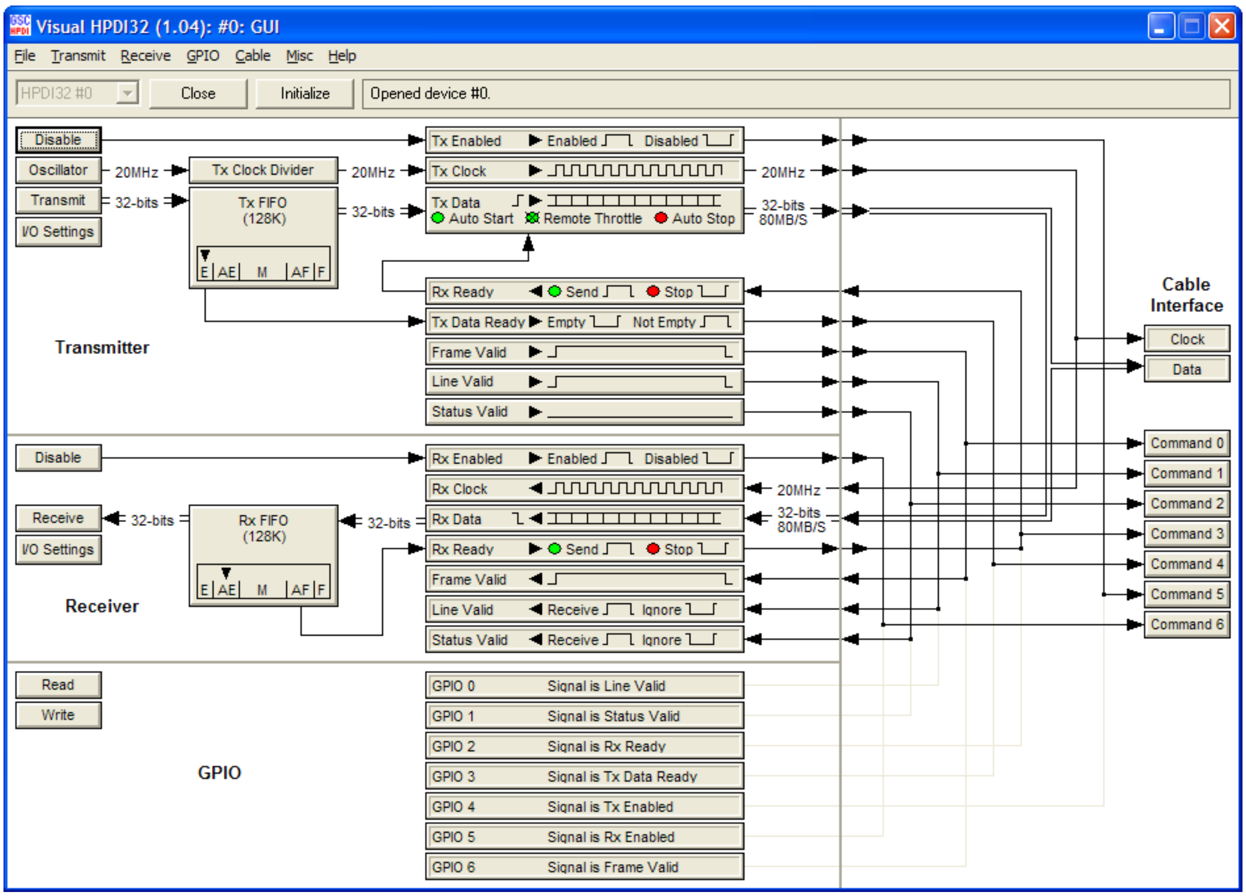

Figure 1 A screen shot of the Visual HPDI32 application (Windows only).

8.9.2. Ring Buffer (Windows Only)

This sample GUI application, *Ring Buffer*, is designed to help identify the I/O buffering configuration that may produce the highest throughput rates for an HPDI32 application. Most HPDI32 parameters are configurable via the menus. The I/O buffering parameters are presented directly in the GUI. The application is designed to collect data and then to dispense with the data collected. Between the collecting and the dispensing is a buffer manager. Data collection can be configured to be 1) filling the receive buffer with an incrementing 32-bit pattern, 2) reading data from an HPDI32, or 3) loading data from a disk file. Dispensing with the data can be configured to be 1) discard the data (do nothing with it), 2) write the data to an HPDI32, or 3) save the data to disk. Refer to Figure 2 below for a screen shot.

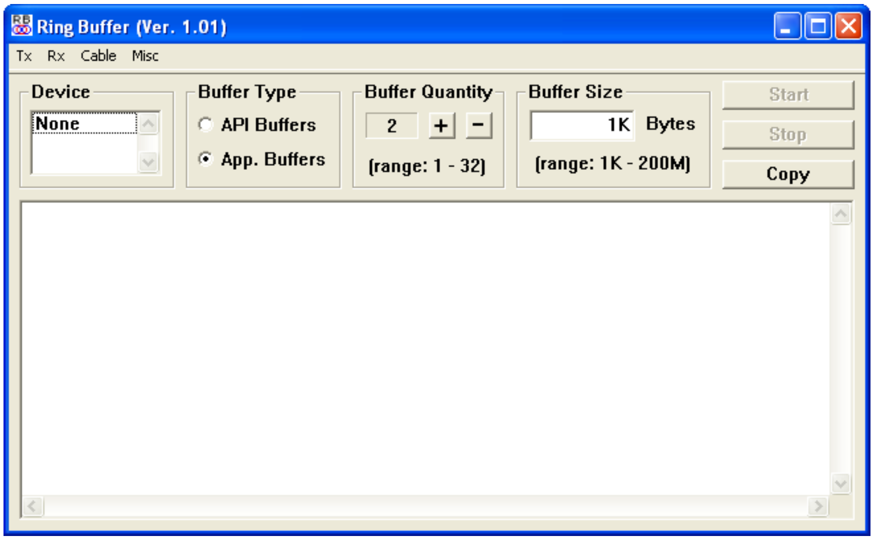

Figure 2 A screen shot of the Ring Buffer application (Windows only).

Document History

Revision	Description
October 7, 2014	Updated to release 6.2.0. Updated the kernel support table. Added the GPIO Library.
October 31, 2013	Updated to release 6.1.0. The soft links requirement section has been removed. The CPU support table has been updated. Added the Remote Throttle sample application to the release. The directory structure has been reorganized. The overall make script has been renamed to `make_all`. The driver start script was renamed to `start`.
May 27, 2009	Updated to release 6.0.1. Added the "jumpers" sample application.
March 21, 2008	Initial release.