DMI32

Deep Memory 32-bit Digital I/O

PCI-DMI32

Software Development Kit
SDK 6.0.0 Setup Guide
For Linux

Manual Revision: October 29, 2007

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com

Preface

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation
8302A Whitesburg Dr.
Huntsville, Alabama 35802
Phone: (256) 880-8787
FAX: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com

Table of Contents

1. Introduction

This setup guide applies to DMI32 SDK release version 6.0.0.

1.1. Purpose

The purpose of this document is to describe installation, setup and use of the DMI32 Software Development Kit under Linux kernel versions 2.6, 2.4 and 2.2.

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
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 DMI32 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 DMI32's memory.
Device Driver	This refers to the driver executable component of the DMI32 SDK.
DRAM	This is a general reference to the DMI32's onboard memory.
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 DMI32's onboard memory. Otherwise it refers to the reception of data over the cable interface, either into memory or the User I/O ports.
Transfer Buffer	This is what defines a DMI32 memory region to be involved in a cable based data transfer. The buffer is defined by a zero based offset and a size. Both are defined in bytes and are 32-bit aligned.
Tx	For I/O operations this refers to writing data to the DMI32's onboard memory. Otherwise it refers to the transmission of data over the cable interface, either from memory or the User I/O ports.

1.4. Hardware Overview

The DMI32 is a high-capacity memory board with a 32-bit parallel digital I/O interface. The host side connection is 32-bit PCI based. 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 memory of up to 4GB (or 2GB, depending on the model) permits large volumes of data to be transmitted

from or recorded by the board. The DMI32 can transmit or receive single blocks up to the size of its memory. Alternatively, it can transmit or receive these blocks continuously without host intervention.

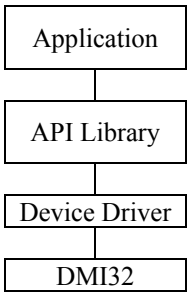
The DMI32 offers a half-duplex external I/O interface. The board can either transmit or receive data, but it cannot do both simultaneously. In addition to the 32 synchronous data I/O lines, the external interface includes a set of User I/O signals that control initiation of data transfer. They may also be used as limited general purpose 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 indefinitely.

1.5. Software Overview

The software interface to the DMI32 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 DMI32 devices. The interface provided by the API Library is based on the board's functionality and is organized around the DMI32's set of main hardware features.

1.5.1. Software Architecture

An application communicates with a DMI32 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 DMI32 devices. The diagram below describes the components and how they fit together.



This is any application written to communicate with one or more DMI32 devices using the DMI32 API Library.

This library presents a DMI32 feature based interface to applications wishing to communicate with DMI32 devices.

The driver provides access to DMI32 devices.

This refers to any number of installed DMI32 devices.

1.6. Compatibility

The API Library interface presented in this version of the SDK is identical for kernel versions 2.6, 2.4 and 2.2. 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. Applications should be able to use the updated driver 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. File System Soft-Links Required Under Linux 2.6

Under kernel version 2.6, the driver build procedure uses file system soft-links. This is done to make the driver sources appear in the driver build directory, which is a requirement of the 2.6 kernel module build environment. (The driver sources are distributed in a number of different directories.) Under the 2.4 and 2.2 kernels, the build procedure will automatically locate the files in their default locations.

1.8. Limitations and Restrictions

1.8.1. 64-bit DMA Not Supported

The DMI32 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 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.

1.8.2. Application Buffer DMA Not Supported Under Linux 2.2

Under kernel version 2.2, DMA is unsupported when using Application Buffers. I/O requests made with DMA and Application Buffers will return the error `GSC_UNSUPPORTED_FUNCTION`. Under the 2.2 kernel, applications should either use PIO mode data transfers or they should 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 DMI32 and this SDK. The specifications provide the information necessary for an in-depth understanding of the specialized features implemented on this board.

- The *DMI32 Software Development Kit Reference Manual* from General Standards Corporation.
- The applicable *DMI32 User Manual* from General Standards Corporation.
- The *PCI9080 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

  PLX data books are available from PLX at the following location.

  PLX Technology Inc.
  870 Maude Avenue
  Sunnyvale, California 94085 USA
  Phone: 1-800-759-3735
  WEB: http://www.plxtech.com

2. Installation

2.1. CPU and Kernel Support

The SDK is designed to operate with Linux kernel versions 2.6, 2.4 and 2.2 running on a PC system with one or more Intel x86 processors. This release of the SDK was tested under the below listed kernels.

> **NOTE:** Under the 2.2 kernel, applications may have no difficulty at all using the API, however support under the 2.2 kernel is incomplete in this release.

Kernel	Distribution	x86	
		32-bit	64-bit
2.6.21	Red Hat Fedora Core 7	Yes	Yes
2.6.18	SUSE 10.2	Yes	Yes
2.6.18	Red Hat Fedora Core 6	Yes	Yes
2.6.16	SUSE 10.1	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 Enterprise Linux Workstation Release 4	Yes	Yes
2.6.9	Red Hat Fedora Core 3	Yes	Yes
2.4.21	Red Hat Enterprise Linux Workstation Release 3	Yes	
2.4.20	Red Hat Linux 9	Yes	
2.4.18	Red Hat Linux 8.0	Yes	
2.4.18	Red Hat Linux 7.3	Yes	
2.4.7	Red Hat Linux 7.2	Yes	
2.2.14	Red Hat Linux 6.2	Yes	

> **NOTE:** The driver and the API Library will have to be built before it 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.

2.2. The /proc File System

While the driver is loaded, the text file `/proc/dmi32` 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: 6.1.2.1.1
built: Oct 29 2007, 13:47:24
boards: 1
```

Entry	Description
`version`	This gives the driver version number in the form `x.xx`.
`built`	This gives the driver build date and time 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
dmi32_sdk_linux.tar.gz	This archive contains the entire set of SDK files.
dmi32_sdk_rm.pdf	This is a PDF version of the SDK reference manual. (A copy is in the archive.)
dmi32_sdk_sg_linux.pdf	This is a PDF version of this user manual. (A copy is in the archive.)

2.4. Directory Structure

The following table describes the basic directory structure utilized by the DMI32 SDK. During installation the directory structure is created and populated with the DSK files.

Directory Structure	Content
dmi32	This is the SDK's root directory.
dmi32/common	This directory contains files used by the DMI32 SDK which are also used by other product SDKs.
dmi32/common/api	This directory contains library interface files used by the DMI32 SDK which are also used by other product SDKs.
dmi32/common/driver	This directory contains driver files used by the DMI32 SDK which are also used by other product SDKs.
dmi32/dmi32	This directory contains files specific to the DMI32.
dmi32/dmi32/api	This directory contains library interface files specific to the DMI32.
dmi32/dmi32/bbtest	This directory contains the files for the DMI32 board-to-board test application. Refer to section 8.3 on page 24.
dmi32/dmi32/docsrc	This directory contains the dmi32_dsl library sources. Refer to section 7 beginning on page 20.
dmi32/dmi32/driver	This directory contains driver files specific to the DMI32. Refer to section 5 on page 15.
dmi32/dmi32/sbtest	This directory contains the files for the DMI32 single board test application. Refer to section 8.2 on page 23.
dmi32/dmi32/tx	This directory contains the files for the DMI32 transmit sample application. Refer to section 8.1 on page 22.
dmi32/dmi32/utils	This directory contains utility files used by the sample and test applications.

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 DMI32 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 15. For specific information on building and installing the API Library refer to section 6 on page 18.

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

1. Change the current directory to /usr/src/linux/drivers (the path name may vary among distributions and kernel versions) or to another directory of your choosing.

2. Copy the archive file dmi32_sdk_linux.tar.gz into the current directory. (The file name may include the SDK version number.)

NOTE: Under the 2.6 kernel the file system where the archive is extracted must support soft links. For additional information refer to section 1.7 on page 7.

3. Issue the following command to decompress and extract the files from the provided archive. This creates the directory `dmi32` in the current directory, and then copies all of the archive's files into this new directory.

   ```
   tar –xzvf dmi32_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 `dmi32_make`. 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 14. For individual instructions on building and installing the API Library refer to section 6 on page 18.

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

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. From this directory change to the root DMI32 directory by issuing the below command.

   ```
   cd dmi32
   ```

3. Issue the below command to initiate a complete make, including starting the driver and installing the SDK shared library.

   ```
   ./dmi32_make
   ```

   The first time through the make should take less than two minutes on most machines. Observe the output of the overall make process and watch for any and all errors and warnings. The output from each target reported by the script should produce no errors or warnings.

2.7. Version Numbers

The SDK, the Device Driver and the API Library all include version numbers. These are listed in the table below and are described in the following paragraphs. The API Library file name includes the library version number after being built.

Component	File Name	Version
SDK	`dmi32_sdk_linux.tar.gz`	6.0.0
Device Driver	`dmi32.ko`	6.1.2.1.1 (2.6 uses `.ko`, 2.4 and 2.2 use `.o`)
API Library	`libdmi32_api.so.*`	7.1.7.1.3 (* This refers to the library version number.)

2.7.1. SDK Version Number

The SDK version number is maintained independently and is generally unrelated to the version numbers for the Device Driver and the API Library. The version number is found only in this Setup Guide, though it may at times be included in the SDK file name.

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 recorded in the text file `/proc/dmi32`. It can also be read by an application via the `dmi32_version_get()` call with the `GSC_VERSION_DRIVER` identifier, or the utility macro service `DMI32_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 `dmi32/dmi32/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 `dmi32_version_get()` call with the `GSC_VERSION_LIBRARY` identifier, or the utility macro service `DMI32_VERSION_GET_LIBRARY()`.

3. Removal

Follow the below steps to remove the SDK. This includes the Device Driver, the API Library and all related support files.

NOTE: This procedure requires superuser 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 16.

2. Change to the directory where the SDK archive was installed. This 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 SDK. (The archive file name may include the SDK version number.)

   ```
   rm –rf dmi32_sdk_linux.tar.gz dmi32
   ```

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

   ```
   rm –rf /dev/dmi32*
   ```

5. If the automated startup procedure was adopted (refer to section 5.1.2.2 on page 16), then edit the system startup script `rc.local` and remove the line that invokes the `dmi32_start` script. The file `rc.local` should be located in the `/etc/rc.d` directory.

6. Remove the DMI32 shared library and its references from `/usr/local/lib`. Do this by executing the below commands.

   ```
   rm –rf /usr/local/lib/*dmi32*
   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 `dmi32_api.h` in each module referencing an API component. Second, configure the compiler for multithreaded operation, as applicable. Third, expand the include file search path to search the directories where the library header and its included files are located. This includes the following list of subdirectories under the directory tree where the SDK was installed.

Subdirectory
`dmi32/common/api`
`dmi32/common/api/linux`
`dmi32/dmi32/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 instructions refer to section 6 on page 18. The second argument is to link the applications with both the API Library and the pthread library. This can typically be done by adding the arguments `-ldmi32_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 DMI32 devices to be accessed. For instructions refer to section 4 on page 14. 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 instructions refer to section 6 on page 18.

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 in section 2.4 on page 10. The instruction for building, loading and shutting down the driver are given in the paragraphs that follow.

5.1.1. Build

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

Follow the below steps to build the driver.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the makefile, as shown in the below command line.

   ```
   cd dmi32/dmi32/driver/linux/gcc
   ```

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

   ```
   make clean
   ```

4. 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 the 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 superuser privileges.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the startup script, as shown in the below command line.

   ```
   cd dmi32/dmi32/driver/linux/gcc
   ```

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.

```
./dmi32_start
```

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

NOTE: The DMI32 device node major number is assigned dynamically by the kernel. The minor numbers and the device node suffix numbers are index numbers beginning with zero, and increase by one for each additional board installed.

4. Verify that the device module has been loaded by issuing the below command and examining the output. The module name `dmi32` 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/dmi32*
   ```

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 superuser 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 DMI32 SDK archive was installed.

   ```
   /usr/src/linux/drivers/dmi32/dmi32/driver/linux/gcc/dmi32_start
   ```

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.

5.1.3. 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/dmi32
   ```

   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 superuser privileges.

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

   ```
   rmmod dmi32
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `dmi32` should not be in the list.

   ```
   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 the subdirectories as described briefly in the directory structure in section 2.4 on page 10. 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. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the makefile, as shown in the below command line.

   ```
   cd dmi32/dmi32/api/linux/gcc
   ```

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

   ```
   make clean
   ```

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

   ```
   make all
   ```

5. 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 superuser privileges.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the library and the startup script, as shown in the below command line.

   ```
   cd dmi32/dmi32/api/linux/gcc
   ```

3. 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.

4. 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 that portion of the library file name that precedes the second period (i.e. `libdmi32_api.so`). The name of the second soft link should be that portion of the library file name that precedes the third period (i.e. `libdmi32_api.so.6`).

```
ls –l /usr/local/lib/
```

5. 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 dmi32
   ```

6.3. Uninstall

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

NOTE: These steps require superuser privileges.

1. Remove the DMI32 shared library and links from the `/usr/src/lib` directory using the below commands.

   ```
   rmmod /usr/local/lib/*dmi32*
   ldconfig
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `dmi32` should not be in the list.

   ```
   lsmod
   ```

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 DMI32 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 DMI32 `docsrc` directory. The sources include a makefile to build the library file `dmi32_dsl.a`.

File	Description
`docsrc\*.c`	These are the C source files.
`docsrc\dmi32_dsl.h`	This is the library header file.
`docsrc\linux\gcc\makefile`	This is a `gcc` makefile.
`docsrc\linux\gcc\makefile.dep`	This is an automatically generated make dependency file.

7.1. dmi32_dsl.a

The build target for these sample files is the statically linkable library `dmi32_dsl.a`. This library can be used with customer applications as is if so desired. These files also provide building blocks upon which DMI32 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. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the makefile, as shown in the below command line.

   ```
   cd dmi32/dmi32/docsrc/linux/gcc
   ```

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

   ```
   make clean
   ```

4. Build the library by issuing the below command. This should take less than about 30 seconds.

   ```
   make all
   ```

7.3. Using the Library

7.3.1. Compile Time Use

Compile time use has two requirements. First, include the header file `dmi32_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 `dmi32/dmi32/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 `dmi32_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 `dmi32/dmi32/docsrc` under the directory tree where the SDK was installed.

8. Sample Applications

8.1. Transmit

This sample application provides a command line driven Linux application that transmits a block of data out the cable interface. This application makes no assumptions about a cable being attached, but will report if a clock is present at the cable interface. The data is written blindly to the board. This sample can be used as the starting point for application development or as an aid to one's learning curve. The application's sources are located under the `dmi32/dmi32/tx` subdirectory tree under the directory where the SDK was installed. The application also uses some of the utility sources located under the `dmi32/dmi32/utils` subdirectory.

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

8.1.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the makefile, as shown in the below command line.

   ```
   cd dmi32/dmi32/tx/linux/gcc
   ```

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

   ```
   make clean
   ```

4. Build the application by issuing the below command. This should take about 10 seconds.

   ```
   make all
   ```

8.1.2. Execute

> **CAUTION:** Damage may occur to the DMI32 or any attached equipment if either the cable or the attached equipment are not compatible with the DMI32. Damage may result because the application drives various external output signals. No damage will result if no cable at all is attached.

Follow the below steps to execute the sample application.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the executable, as shown in the below command line.

   ```
   cd dmi32/dmi32/tx/linux/gcc
   ```

3. Start the application by issuing the command given below. The application uses the command line arguments given to direct its course of action. Once started the application will automatically execute a series of operations against the specified board. The application will repeat the cycle as called for by the arguments and the accumulated test results. A single cycle should take about one second to complete. The command line arguments are described in the table below.

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

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.
`board`	This is the index of the board to access and is required only if more than one board is in the system.

8.2. Single Board Test

This sample application provides a command line driven Linux application that tests the library interface and the functionality of a DMI32. The application's purpose is to verify the API interface and to test the operation of a single board with a disconnected cable. This sample can be used as the starting point for application development or as an aid to one's learning curve. The application performs an automated test of the board selected. The application includes the below listed files. The application's sources are located under the `dmi32/dmi32/sbtest` subdirectory tree under the directory where the SDK was installed. The application also uses some of the utility sources located under the `dmi32/dmi32/utils` subdirectory.

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

8.2.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the makefile, as shown in the below command line.

   ```
   cd dmi32/dmi32/sbtest/linux/gcc
   ```

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

   ```
   make clean
   ```

4. Build the application by issuing the below command. This should take about 15 seconds.

```
make all
```

8.2.2. Execute

CAUTION: Damage may occur to the DMI32 or any attached equipment if either the cable or the attached equipment are not compatible with the DMI32. Damage may result because the application drives various external output signals. No damage will result if no cable at all is attached.

Follow the below steps to execute the sample application.

1. Change to the directory where the SDK archive was installed. This may have been `/usr/src/linux/drivers`. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the executable, as shown in the below command line.

   ```
   cd dmi32/dmi32/sbtest/linux/gcc
   ```

3. Start the application by issuing the command given below. The application uses the command line arguments given to direct its course of action. Once started the application will automatically execute a series of tests to verify the operation of the board. The application will repeat the test cycle as called for by the arguments and the accumulated test results. A single test cycle should take about 25 seconds to complete. Enabling the memory test will add about three minutes per installed gigabyte to each test cycle. The command line arguments are described in the table below.

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

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`	Perform a simple memory test.
`-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.
`board`	This is the index of the board to access and is required only if more than one board is in the system.

8.3. Board-to-Board Test

This sample application provides a command line driven Linux application that tests the functionality of two DMI32 boards connected by a pass-through cable. The application's purpose is to test the operation of two boards connected back-to-back. The sample can be used as the starting point for application development or as an aid to one's learning curve. The application performs an automated test of the two boards selected. The application includes the below listed files. The sources are located under the `dmi32/dmi32/bbtest` subdirectory tree under the directory where the SDK was installed. The application also uses some of the utility sources located under the `dmi32/dmi32/utils` subdirectory.

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

utils/*.h	These are the headers for the utility sources used by the application.
utils/linux/*.c	These are utility sources used by the application.

8.3.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the SDK archive was installed. This may have been /usr/src/linux/drivers. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the makefile, as shown in the below command line.

   ```
   cd dmi32/dmi32/bbtest/linux/gcc
   ```

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

   ```
   make clean
   ```

4. Build the application by issuing the below command. This should take about 15 seconds.

   ```
   make all
   ```

8.3.2. Execute

CAUTION: When using this sample application the DMI32s and any externally attached equipment may be damaged if the DMI32s' external interface has a cable attached to other than the two boards being accessed. Damage may result because the sample application drives various external output signals. No damage will result if no cable at all is attached.

Follow the below steps to execute the sample application.

1. Change to the directory where the SDK archive was installed. This may have been /usr/src/linux/drivers. (The path name may vary among distributions and kernel versions.)

2. Change to the subdirectory containing the executable, as shown in the below command line.

   ```
   cd dmi32/dmi32/bbtest/linux/gcc
   ```

3. Start the application by issuing the command given below. The application uses the command line arguments given to direct its course of action. Once started the application will automatically execute a series of tests to verify the operation of the boards. The application will repeat the test cycle as called for by the arguments and the accumulated test results. A single test cycle should take about 10 seconds to complete. Additional time will be required if there are data transfer errors while logging is enabled. The command line arguments are described in the table below.

   ```
   ./bbtest <-c> <-C> <-l> <-m#> <-n#> <-v> <#1> <#2>
   ```

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.
-l	Log transfer error data to a file named dmi-XXXX.txt, where XXXX starts at zero.
-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.
-v	Be verbose with some error conditions.
#1	This is the index of the first board to access and is required only if more than two boards are in the system.
#2	This is the index of the second board to access and is required only if more than two boards are in the system.

Document History

Revision	Description
October 29, 2007	Initial release.