XDS Linux (Gentoo 2006.0) H.100 / SCSA Driver Package Reference Manual

Driver Version 1.0
September 2006

American Tel-A-Systems, Inc.

Contents

XDS Linux (Gentoo 2006.0) PCI (H.100) / ISA SCSA Driver Reference Manual

American Tel-A-System, Inç.
608-838-4194
4800 Curtin Drive, McFarland, WI 53558, USA
http://www.amtelco.com/
257M023

Driver Software Package Installation and Removal

This page was intentionally left blank.

1.0 Introduction

The XDS Linux (Gentoo 2006.0) PCI (H.100)/ISA SCSA Driver comes in the form of one CD-ROM or if downloaded – an install script, the driver package in the form of a tar file, and a removal script.

This software driver package contains the low-level device driver, SCSA-based libraries with H.100 board functions, a simple demonstration program, a text-based program for sending and receiving messages which employs the signaling mechanism of the driver, the timeslot assignment routines for use with Dialogic software, all of the source code for the included programs, a downloader that allows the user to "flash" their XDS board with different firmware releases, the low-level device driver, a user function library, an install package script, and a remove package script.

At the time of release for version 1.0 of the XDS driver, the XDS driver has only been tested on Gentoo Linux 2006.0, kernel 2.6.

1.1 Hardware Installation Procedure

Each XDS PCI (H.100) board informs Linux what resources it requires to operate. Each XDS H.100 board is assigned a number in the order found starting at 16 decimal and going up to 31. Each ISA board is assigned the same board number as the "SW1" switch setting (from '0' to 'F'). Each ISA board must use a different "SW1" number - no two boards may share one. Before starting the system, you will need to be sure that the appropriate system resources are available for the XDS board(s) to be installed, i.e.: IRQ and memory. For details on the hardware interface, consult the appropriate XDS board technical manual.

1.2 Software / Low-level Driver Installation Procedure

To install the driver package, log into the system as root. Copy the driver package (tar file), install script, and removal script to a temporary directory on your local drive. Once this is done, in the temporary directory, run the command "./install.sh".
The user will be asked for the XDS ISA board settings (if no XDS ISA board is to be used, then enter '0' for the IRQ number).

The first setting the user will be asked will be what IRQ to use for all of the XDS ISA boards. The current XDS ISA boards allow the following IRQs: 3, 4, 5, 6, 7, 10, 11, 12, 14, or 15. The default is an IRQ setting of 15. Enter the number of the IRQ used from the above list and then press the "**Enter**" key. Enter a '**0**' and then press the "**Enter**" key, if you do not have any ISA boards in the system.

The following setting will be the "SW2" number. The current XDS ISA boards allow the following any "SW2" setting between '**0**' and '**F**'. The default value is '**F**'. All XDS ISA boards must have the same "SW2" setting. Enter the "SW2" number to be used and then press the "**Enter**" key.

The last setting asked is the DPRAM address to be used by all of the boards. XDS ISA boards require a 32K block of contiguous memory starting at one of the ten listed locations. The default address will be 0xD000. Enter the number, which corresponds to the address to be used, and then press the "**Enter**" key.

The user will then be asked if the (above) information is correct (it will be displayed for the user). Simply type in a '**Y**' or '**y**' for Yes, or '**N**' or '**n**' for No and then press the "**Enter**" key. If you answer yes to the above question, the system will then complete the installation to activate any and all XDS H.100 and XDS ISA boards (if configured) present in the system. The user may now communicate with the hardware installed. There is no need to reboot.

1.3 Software / Low-level Driver Removal Procedure

Should the user desire to remove the driver package, simply go to the temporary directory where the driver package was copied to, and run the command "./remove.sh".

1.4 Boot Time Initialization

At boot time, for each XDS board in the chassis, Linux will create a new device instance of the XDS driver. For each XDS driver instance started, the driver will test the board for functionality and activate the board and its associated device file, if successful. A list of which boards are present will be displayed as they come up. If a board is detected as present but not functioning, an error message will be displayed. In the case of an error, this display will be retained in the system log file (typically /var/log/messages).

This page was intentionally left blank.

Driver Package
Programs and Source Code

This page was intentionally left blank.

2.0 XDS General Demo Programs

Several demonstration programs are included with this package and may be found in the **/usr/amtelco/scsa/demos** directory.

This release of the driver does not include Voice Playback / Resource support.

All command message strings sent to any board, using any one of the provided utilities, must be in CAPITAL letters.

demo1 checks for the presence of XDS boards using the *xds_id()* XDS library function. If a board is found it sends a version request message using the *xds_msg_send()* XDS library function and receives the response using the *xds_query_receive()* XDS library function. It then displays the response message(s). If a board is not present, the phrase "board is not present" is displayed on that line. This program also demonstrates how to open and close the driver and how to enable interrupts from the board with the *xds_msg_on()* XDS library function.

tstsig is a text-based interactive utility that allows the user to send and receive messages from any XDS board. Built in to this program is an option to reset any PCI-based XDS board. It also demonstrates the signaling mechanism (SIGPOLL) of the low-level driver. This program displays the menu options when it is run.

xdspcires is a text-based utility that lists each and every PCI-based XDS board in the system along with its respective ID code, device (board) number, PCI bus number, and PCI slot number.

octest simply opens a handle to the XDS device and then closes it while displaying the status of each step.

3.0 XDS Board Flash Downloader

Most XDS boards are equipped with flash memory, which contains a downloadable version of the board program (firmware). However, you may want to check the board technical manual to be sure or consult an Amtelco service technician. New revisions of the board program (firmware) can be downloaded to the board using the downloader program **lx386dlc**. To use this program, the driver must be started and recognize the board. The .hex to be downloaded file should include a header identifying the board type so that it can only be loaded onto a compatible board.

The syntax for the downloader is:

```
lx386dlc <hexfile.hex> <segment> <board number>
```

where the segment specifier is either a ‘C’ for the control processor or “D” for the DSP processor. For example:

```
lx386dlc 257h000b.hex c 16
```

would load the hex file “257h000b.hex” into the control processor program on board 16.

4.0 XDS Timeslot Assignment Routine

If the user wishes to set up their H.100 or SC Bus for SC-based timeslot usage, they will want to use the SC Bus timeslot assignment helper applications described in this chapter.

There are three steps involved in setting up the XDS hardware/software for use with the Dialogic software release. These include configuring the XDS hardware, setting up the XDS service daemons to run at startup, and rebooting the machine.

The user must first decide how many timeslots out of the total timeslots available on the CT bus that they would like to reserve for XDS boards (total for all XDS boards). To help automate the process of timeslot selection on the XDS boards, the utility **xdstscfg** has been provided to the user.

This will need to be run if there are any XDS PCI MC3 and/or ISA SCx multi-chassis boards in the system and/or an XDS board will be providing the master clock to the CT bus. If this is not the case, skip to section 4.2.

4.1 xdstscfg

First, run **xdstscfg**. This will allow the user to configure their hardware:
1) Configure any SCx (SCSA Multi-chassis) boards. (see chart ‘A’ for available modes)
2) Configure any H.100 MC-3 Multi-chassis boards. (see chart ‘B’ for available modes)
3) Select an SCSA board to be the master clock (if it is not an SCx board). If the user selects a BRI interface board to provide the master clock to the SC bus, they will have the option to specify a port on the board to use for the clock (if they desire). Enter the port number (i.e. 0 – 11) or ‘N’ to use the local oscillator to provide the clock to the SC bus.
4) Select the encoding mode (A-Law or μ-law) that will be used for all of the boards. A-Law is most commonly used in Europe and Asia, and μ-Law is most commonly used in North America and Japan.
5) Select the H.100 CT bus bit rate to be used for all of the XDS H.100 boards (SCSA 4MHz or H.100 8MHz). If the H.100 board’s CT bus will be connected to another H.100 board’s CT bus, then the user will want to select H.100. If it will be connected to an SCSA bus, then select SCSA.

Once all of the information has been entered, a configuration file (in ASCII format) will be created, named **xdscfg** in the /usr/amtelco/scsa/config directory. This file will contain all of the board information needed for the rest of the timeslot assignment routine.

4.2 xdsdaemons

Next, run **xdsdaemons**. This will setup the XDS service daemons for the user. One of the XDS service daemons will be named **xdsclocksvc**, which controls setting the clocks, bit rates (on H.100 boards), and encoding modes on all of the XDS boards every time the system starts. It also creates one named **xdsreqsvc** for setting up the timeslots on the XDS boards. In addition **xdsreqsvc** will calculate the total number of timeslots and enter it into Dialogic’s configuration file **/usr/dialogic/cfg/.sctscfg**.

4.3 reboot

When the boards have been configured and the XDS service daemons have been setup, the user will need to re-boot for the changes to take effect.

4.4 xdstsa

When the system starts up, the Dialogic service daemon will now start the Dialogic portion of the timeslot assignment routine and then run xdstsa (from the XDS entry that is now in the /usr/dialogic/cfg/.sctscfg.sav file). Dialogic's software will then merge **.sctscfg** with **.sctscfg.sav**, we have no control over that.

5.0 XDS Source Code Description

All of the source code and makefiles used to build the all of the XDS programs, library, and driver have been included for the user's convenience. They have all been written in the 'C' programming language, and were built using the native tool set included with Gentoo Linux 2006.0. There will be a standard Linux makefile, named **makefile**, in each directory. If any or all of the code is "re-used" and/or "re-distributed", the American Tel-A-Systems, Inc. copyright information must be included with it. The source code is provided as-is.

5.1 driver

The source code for the XDS low-level Linux driver is located in the /usr/amtelco/scsa/source/driver directory. To re-build the driver, simply type "make" at the command line. If the user wishes to change the XDS ISA board(s) settings, you will first type "./Configure", then "make".

5.2 library

The source code for the XDS user function library is located in the /usr/amtelco/scsa/source/library directory. To re-build the library, run "make" at the command prompt.

5.3 demos

The source code for all of the XDS board demos may all be located in the /usr/amtelco/scsa/source/demos directory. To re-build any of the demos included, run "make <program name>" at the command prompt. Or to build all of the demos, run "make".

5.3 lx386dlc

The source code for the XDS board downloader is located in the /usr/amtelco/scsa/source/dloader directory. To re-build this, run “make” at the command prompt.

5.4 timeslot assignment programs

The source code for the XDS timeslot assignment routines is located in the /usr/amtelco/scsa/source/tsa directory. To re-build any of these, run “make <program name>” at the command prompt. Or to build all of them, run “make all”.

Chart ‘A’ – ISA SCx clock modes

Mode 0 -
No bus clocks

Mode 1 -
Clock source: SCbus
Supplied to: SCxbus right clock

Mode 2 -
Clock source: SCbus
Supplied to: SCxbus left clock

Mode 3 -
Clock source: SCx board
Supplied to: SCbus and SCxbus right clock

Mode 4 -
Clock source: SCx board
Supplied to: SCbus and SCxbus left clock

Mode 5 -
Clock source: 8Kref
Supplied to: SCbus and SCxbus right clock

Mode 6 -
Clock source: 8Kref
Supplied to: SCbus and SCxbus left clock

Mode 7 -
Clock source: Right clock
Supplied to: SCbus

Mode 8 -
Clock source: Right clock
Supplied to: SCbus and SCxbus left clock

Mode 9 -
Clock source: Left clock
Supplied to: SCbus

Mode A –
Clock source: Left clock
Supplied to: SCbus and SCxbus right clock

Chart ‘B’ – H.100 clocks and MC-3 ring modes

Mode 0 -
No clocks provided
No ring control provided

Mode 1 -
Clock mode: Slave clock mode from the CT bus, 4 MHz
Ring mode: extended (both available)
Ring failure mode: no failure ring mode set

Mode 2 -
Clock mode: Slave clock mode from the CT bus, 4 MHz
Ring mode: Redundant rings, ring 0 is primary
Ring failure mode: ring 1

Mode 3 -
Clock mode: Slave clock mode from the CT bus, 4 MHz
Ring mode: Redundant rings, ring 1 is primary
Ring failure mode: ring 0

Mode 4 -
Clock mode: Master clock mode, freerun, no compatibility clocks
Ring mode: extended (both available)
Ring failure mode: no failure ring mode set

Mode 5 -
Clock mode: Master clock mode, freerun, no compatibility clocks
Ring mode: Redundant rings, ring 0 is primary
Ring failure mode: ring 1

Mode 6 -
Clock mode: Master clock mode, freerun, no compatibility clocks
Ring mode: Redundant rings, ring 1 is primary
Ring failure mode: ring 0

Mode 7 -
Clock mode: Master clock mode, local network, no compatibility clocks
Ring mode: extended (both available)
Ring failure mode: no failure ring mode set

Mode 8 -
Clock mode: Master clock mode, local network, 4 MHz CT bus clock
Ring mode: extended (both available)
Ring failure mode: no failure ring mode set

Mode 9 -
Clock mode: Master clock mode, local network, no compatibility clocks
Ring mode: Redundant rings, ring 0 is primary
Ring failure mode: ring 1

Mode A -
Clock mode: Master clock mode, local network, no compatibility clocks
Ring mode: Redundant rings, ring 1 is primary
Ring failure mode: ring 0

Linux Gentoo Driver IOCTL Description

This page was intentionally left blank.

This PCI (H.100)/ISA Driver is designed to provide an interface between XDS boards and applications running under the Linux (Gentoo 2006.0) operating system. A companion library is also provided that allows easy control of a set of XDS boards from a ‘C’ Language program.

A common interface mechanism is used for all XDS boards regardless of type. Control of the boards is accomplished thorough commands which are in the form of NULL terminated ASCII strings. Responses, acknowledgements, state change and error information are also passed from the XDS boards in the form of ASCII strings. All boards share a transmit and receive mailbox and a corresponding flag. Each board also provides a limited amount of buffering (eight messages deep) in either direction.

The XDS Basic Rate ISDN Board and T1 / E1 Boards share two additional mailboxes that are used for passing Layer 3 messages for the ‘D’ channel messages. These mailboxes are used in conjunction with the main transmit and receive mailboxes. Details can be found in the appropriate hardware reference manual.

The driver consists of a streams device driver using the normal **open** and **close** functions and uses the **ioctl** function to actually exchange information with the board. The **ioctl** function supports nine commands to allow for the sending and receiving of messages, the writing and reading of the dual-ported RAM on the boards, identifying the board, and resetting the board.

The driver “transmit” command (XMT) writes messages directly to the mailbox on the appropriate board. The driver places received messages on one of two queues. Acknowledgements, state change messages, and error messages are passed through the receive queue. Query responses and Version Request responses are passed through a separate receive query queue. One queue for each purpose is defined for all XDS boards in a system. An **ioctl** command is provided for reading each queue. Each of these queues is capable of buffering up to 63 messages. If the queue is full, the driver will discard additional messages. It is therefore the responsibility of the application to check the queues frequently enough so that they do not fill up.

Commands are provided for reading and writing the dual-ported RAM which each board uses to communicate with the host processor. These commands include protection to prevent reading or writing outside of the dual ported memory on a particular board or for overwriting the mailboxes or configuration information on each board.

The identify board command asks the driver to fill in the requested board’s identity information into the identity structure provided by the caller. The process reference and un-reference commands tell the driver to send or stop sending a signal when a message is received from the board. The reset command tells the driver to reset the requested board.

Application Interface

Applications can interface directly to the driver by using the **ioctl** function call. Through this function, the application can send and receive messages directly to and from XDS boards. It is also possible to directly read or write to the Dual-Ported Ram on the XDS boards.

Open & Close

Before the **ioctl** function can be used by an application, it must first obtain a file handle. This is done by an **open**, i.e.

```
strcpy(device_name, “/dev/xds”);
fd1 = open(device_name, 0_RDRW);
```

If the driver can not be opened, a (-1) will be returned. Before shutting down, the application should use **close** to close the file handle, i.e.

```
close(fd1);
```

The **open** and **close** functions require the header file **<fcntl.h>**.

IOCTL
The **ioctl** call takes the form:

```
ioctl(fd1, cmd, msgp);
```

where fd1 is the file handle obtained by the **open** function, cmd is the ioctl funtion to be preformed, and msgp is a pointer to a structure for the arguments. The templates for these structures are contained in **xdsioctl.h.** Eleven commands are available to an application. These are:

XMT - XDS transmit message function
RCV - XDS receive message function
RCV_QUERY - XDS receive query response message function
WRITE_DPRAM - write to the dual-ported RAM
READ_DPRAM - read from the dual-ported RAM
PROC_REF - enable signaling on received messages
PROC_UNREF - disable signaling on received messages
XDS_GET_BUS_DEVICE_NUM - obtain information on an XDS board
XDS_GET_BOARD_INFO - get board ID, version, and number of ports
XDS_QUEUE_USER_MSG – copy a message to a message queue
XDS_RESET – reset board function
XDS_DRIVER_VERSION – return the low-level driver version number and sub-version number

XMT

ioctl(fd1, XMT, msgp);

int fd1;	device file handle returned by **open**
int cmd = XMT;	transmit message command
struct xds_msg *msgp {	
unsigned char board_number;	the Board Number
char msg[32];	the ASCII text of the message, NULL terminated
unsigned short augTxRxLen;	length of Layer 3 message
unsigned char augTxRxMesg[260];	body of Layer 3 message
}	

Purpose
This command is used to send messages to an XDS board. The board is specified in board_number which corresponds to the intended Board Number. The message is contained in the character array msg, and consists of a NULL terminated character string.

Returns
The ioctl function will return the following codes:

0 - success
1 - board not present
2 - board not responding

Comments
Transmit messages are not queued, but sent directly to the board. If the mailbox is full, XMT will wait up to a tenth of a second before reporting a failure. Note that **augTxRxLen** and **augTxRxMesg** are only when sending a Layer 3 message on the XDS SCSA Basic Rate ISDN Board when the message in **msg** is of the format “ALC” or “ALR”.

RCV

ioctl(fd1, RCV, msgp);

int fd1;	device file handle returned by **open**
int cmd = RCV;	receive message command
struct xds_msg *msgp {	
unsigned char board_number;	the Board Number
char msg[32];	the ASCII text of the message, NULL terminated
unsigned short augTxRxLen;	length of Layer 3 message
unsigned char augTxRxMesg[260];	body of Layer 3 message
}	

Purpose
This command is used to receive normal messages from boards. Query and version request messages are returned on the query response queue and read with the RCV_QUERY command. The board sending the message is contained in board_number, while the text of the message is in the character array msg in the form of a NULL terminated ASCII string.

Returns
If a message is available, ioctl will return a 0, otherwise it will return a 1.

Comments
This command checks to see if there is any message on the receive queue. If there is, it will return with the message. If no message is present, it will return immediately with a return value of 1.

Normal messages are placed on the receive queue. These include acknowledgements, state change messages, and error messages. Version request and query responses are placed on the query response queue and can be read using the RCV_QUERY command.
The elements **augTxRxLen** and **augTxRxMesg** are only valid when receiving Layer 3 messages on the XDS SCSA Basic Rate ISDN Board and the message in **msg** is of the format “ALC” or “ALR”.

If the queue becomes full, a “FULL QUEUE” message is placed on the queue with the board_number for that message set to 255. If this message is received it indicates the possibility that messages may have been lost. It is the responsibility of the application to check for messages often enough to prevent this.

RCV_QUERY

ioctl(fd1, RCV_QUERY, msgp);

int fd1;	device file handle returned by **open**
int cmd = RCV_QUERY;	receive query message command
struct xds_msg *msgp {	
unsigned char board_number;	the Board Number
char msg[32];	the ASCII text of the message, NULL terminated
unsigned short augTxRxLen;	length of Layer 3 message
unsigned char augTxRxMesg[260];	body of Layer 3 message
}	

Purpose
This command is used to receive version request responses and query responses which are placed on the query response queue by the driver. The board sending the message is contained in board_number, while the text of the message is in the character array msg as a NULL terminated ASCII string.

Returns
If a message is available, ioctl will return with a 0. If no message is available, ioctl will return with a 1.

Comments
Unlike the RCV command, the RCV_QUERY command does not return immediately if there is no message available. It will wait up to a tenth of a second for a message to be on the queue. This implementation was made because of the finite time that it takes a board to respond to a version request or a query. By doing so, it eliminates the need for the application to implement a timeout mechanism.

Version request response messages always begin with the letter “Q”. Query responses always begin with the letter “Q” or have a “Q” as the second letter.. These messages are always placed on the query response queue and must be read using the RCV_Query command.
The elements augTxRxLen and **augTxRxMesg** never contain valid data when using RCV_QUERY.

If the queue becomes full, a “FULL QUEUE” message is placed on the queue with the board_number for that message set to 255. If this message is received it indicates the possibility that messages may have been lost. It is the responsibility of the application to check for messages often enough to prevent this.

WRITE_DPRAM

ioctl(fd1, WRITE_DPRAM, dpramp);

int fd1;	device file handle returned by **open**
int cmd = WRITE_DPRAM	write to dual-ported RAM command
struct xds_dpram *dpramp {	
unsigned char board_number;	the Board Number
int offset;	the offset in bytes into dual-ported RAM
int size;	the number of bytes to be written
unsigned char *buffer;	a pointer to the bytes to be written
}	

Purpose
This command is used to write information into the dual-ported RAM on the XDS board specified in board_number. This is normally not necessary as the XMT command can be used to control the board. However, for diagnostic purposes or for downloading firmware, this command may be used.

Returns
The ioctl function will return the following codes:

0 - success
1 - no board present
2 - attempt to write before the alterable area of the board
3 - attempt to write beyond the alterable area of the board

Comments
The WRITE_DPRAM is included in the ioctl commands to facilitate writing a downloader. It normally will not be necessary for an application to use this command directly. WRITE_DPRAM prevents writing to the first 256 bytes of the dual-ported RAM which contain the mailboxes, flags, and configuration information in H.100 boards and the last 256 bytes in ISA boards.

READ_DPRAM

ioctl(fd1, READ_DPRAM, dpramp);

int fd1;	device file handle returned by **open**
int cmd = READ_DPRAM	read to dual-ported RAM command
struct xds_dpram *dpramp {	
unsigned char board_number;	the Board Number
int offset;	the offset in bytes into dual-ported RAM
int size;	the number of bytes to be read
unsigned char *buffer;	a pointer to the buffer to contain the data
}	

Purpose
This command can be used to read directly the contents of a portion of the dual-ported RAM. This may be done to obtain configuration information or for diagnostic purposes. The information read is placed in a buffer supplied by the application.

Returns
The ioctl function returns the following codes:

0 - success
1 - board not present
2 - attempt to read before the alterable area of the board
3 - attempt to read beyond the alterable area of the board

Comments
This command may be used to obtain configuration information on the board, such as the board type, port states, etc. However, there also exist library functions that will achieve the same thing which may be easier to use. It is also possible to use this command for diagnostic purposes to display the contents of the mailboxes and the state of the transmit and receive flags.

PROC_REF

ioctl(fd1, PROC_REF, NULL);

int fd1;	device file handle returned by **open**
int cmd = PROC_REF	enable signaling command
NULL	no arguments

Purpose
This command is used to enable the signaling mechanism. When enabled, the driver will notify the calling function or application when a message arrives from an XDS board.

Returns
The ioctl function will return the following codes:

0 - success
1 - no board present

Comments
To use the signaling mechanism, the application must use the function call **sigset(SIGPOLL, handle_pollsig)**. This sets the function **handle_pollsig()** as the handler for incoming SIGPOLL signals. The application must also include the following header file: #include <signal.h>. After the **sigset** call, the PROC_REF command may be issued to enable signaling. Signaling is disabled with the PROC_UNREF command.

PROC_UNREF

ioctl(fd1, PROC_UNREF, NULL);

int fd1;	device file handle returned by **open**
int cmd = PROC_UNREF	disable signaling command
NULL	no arguments

Purpose
This command is used to disable signaling. The driver will no longer notify the calling function or application when a message is received from an XDS board.

Returns
The ioctl function will return the following codes:

0 - success
1 - no board present

Comments
This command is used to disable the signaling feature of the driver. Signaling may be re-enabled by issuing a PROC_REF command. This command should be issued before the driver is closed. Note: if this call is not made before the **close(),** a safeguard has been added to the driver that will disable the signaling mechanism automatically on a close(). It is the responsibility of the developer to close any open. To cause an application to ignore the signal, the **sigignore(SIGPOLL)** function can be used. The application must include the following header file: #include <signal.h>.

XDS_RESET

ioctl(fd1, XDS_RESET, &board_number);

int fd1;	Utility device file handle returned by **open**
int cmd = XDS_RESET;	Return XDS board from sleep state after swap in
unsigned char board_number;	Board Number to reset

Purpose
This command is used to reset through software any given XDS board. This performs the same checks as those done on driver load. Any configuration different than the one after a hardware reset is lost.

Returns
The ioctl function returns the following codes:

0 - success
1 - no board present
3 - unknown error
4 - illegal argument

Comments
This command leaves the board in the same state as hardware reset, except for PCI configuration.

XDS_GET_BUS_DEVICE_NUM

ioctl(fd1, XDS_GET_BUS_DEVICE_NUM, xdsid *info);

int fd1;	Utility device file handle returned by **open**
int cmd = XDS_GET_BUS_DEVICE_NUM;	Return XDS board from sleep state after swap in
XDSID *info	place to put requested board information in
	info->board_number is the Board Number requested

Purpose
This command is used to obtain the PCI bus and slot number of a specified board.

Returns
The ioctl function returns the following codes:

0 - success
1 - no board present
3 - unknown error
4 - illegal argument

Comments
This command is available for PCI-based boards only.

XDS_GET_BOARD_INFO

BOOL DevIoControl(	
hdriver,	device handle
(DWORD) XDS_GET_BOARD_INFO,	board INFO command
pData,	pointer to input structure
sizeof(XDSID),	length of input structure
pData,	pointer to output structure
sizeof(XDSID),	length of output structure
&data_length,	pointer to number of bytes returned
NULL);	

XDSID id;

typedef struct xdsid {	
unsigned char board_number;	board number
char id[5];	board type (ID)
char version[5];	firmware version
int number_ports;	number of ports
UCHAR pci_device_number;	PCI Board device number
UCHAR pci_bus_number;	PCI Board bus number
}XDSID, *PXDS_ID, xiID, *pXdsId;	

Purpose
This command is used to obtain the ID of a specified board.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
STATUS_BUFFER_TOO_SMALL	size of data structure passed in is incorrect
STATUS_DATA_ERROR	board number used, not valid

Comments
This function return the board ID, version, and number of “ports” associated with a specified XDS board.

XDS_QUEUE_USER_MSG

ioctl(fd1, XDS_QUEUE_USER_MSG, msgp);

int fd1;	device file handle returned by **open**
int cmd = XDS_QUEUE_USER_MSG;	queue user message command
struct xds_msg *msgp {	
unsigned char board_number;	the Board Number
char msg[32];	the ASCII text of the message, NULL terminated
unsigned short augTxRxLen;	length of Layer 3 message
unsigned char augTxRxMesg[260];	body of Layer 3 message
}	

Purpose
This command is used to put messages on to a message queue by the user. The board is specified in board_number which corresponds to the intended Board Number. The message is contained in the character array msg, and consists of a NULL terminated character string.

Returns
The ioctl function will return the following codes:

0 - success
1 - board not present
2 - board not responding

Comments
This call is helpful when the application needs to return an error message on an XDS message queue.

XDS_DRIVER_VERSION

BOOL DevIoControl(	
hdriver,	device handle
(DWORD) XDS_DRIVER_VERSION,	board INFO command
pData,	pointer to input structure
sizeof(XDS_DRIVER_REV),	length of input structure
pData,	pointer to output structure
sizeof(XDS_DRIVER_REV),	length of output structure
&data_length,	pointer to number of bytes returned
NULL);	

```
XDS_DRIVER_REV driver_rev;

typedef struct xds_driver_rev
{
        float driver_version;
        float driver_sub_version;
        float library_version;
        float library_sub_version;
        float vr_library_version;
        float vr_library_sub_version;
}       XDS_DRIVER_REV, *PXDS_DRIVER_REV;
```

Purpose
This command is used to obtain the version and sub-version of the low-level driver.

Returns
The function will return the following codes:

STATUS_SUCCESS	success
STATUS_BUFFER_TOO_SMALL	size of data structure passed in is incorrect
STATUS_DATA_ERROR	board number used, not valid

Comments
This function return the board ID, version, and number of “ports” associated with a specified XDS board.

This page was intentionally left blank.