XDS Linux (Redhat 7.x / 9) H.110 Driver Package Reference Manual

Version 3.0
February 2005

American Tel-A-Systems, Inc.
258M015C ©

This page was intentionally left blank.

Contents

XDS Linux (Redhat 7.x / 9) H.110
Driver Reference Manual

Author: Brian Riek

American Tel-A-System, Inc.
608-838-4194
4800 Curtin Drive, McFarland, WI 53558
http://www.amtelco.com
258M015C

Driver Software Package Installation and Removal

This page was intentionally left blank.

1.0 Introduction

The XDS Linux (Redhat v7.x / 9) cPCI (H.110) Driver comes in the form of one CD-ROM. This CD-ROM contains the driver, H.110 function library, a GUI utility, simple demonstration programs, all of the source code for the included programs, driver, and library, an install package script, and a remove package script. Information on the contents of the disk can be obtained by running **File Manager** from any GUI desktop environment, such as KDE or GNOME.

At the time of release for version 3.0 of the XDS driver, Redhat 8 had not been tested. The use of this driver is at the user's discretion on Redhat 8.

NOTE: Redhat 9, by default, does NOT install the kernel source. The user may have to install the appropriate source for the kernel they are running on their Redhat 9 system. This will need to be done before the XDS device driver and package can be installed.

1.1 Hardware Installation Procedure

Each XDS H.110 board uses 8K of memory and comes in the cPCI form factor. The resources for each PCI device in the system can be viewed in the system BIOS at boot-up.

You will need to be sure that there is a PCI interrupt available for the cPCI board(s).

As with all device drivers in most operating systems, the user must have administrator privileges in order to install/remove a device driver.

You will need to power down the system that the board(s) will be installed in. Make sure to save any work that you may have been doing. Follow the board's hardware manual precisely for the board installation portion. When this step is completed, power the system back on.

1.2 Software / Low-level Driver Installation Procedure

Each XDS H.110 cPCI board informs the system what resources it requires to operate automatically (at boot time). When the driver first loads, it assigns a number that is equal to the physical slot number in which it resides (Geographic addressing). If the Geographic address is not available, the board will be assigned the next available unused number from the driver. For details on the hardware interface, consult the appropriate XDS technical manual.

To install the driver package, simply log into the system as root using your preferred GUI. Insert the XDS H.110 Linux Driver CD-ROM into the CD-ROM drive. Now the user will need to mount the CD-ROM drive (i.e.: by typing "mount /dev/cdrom /mnt/cdrom"). If you have downloaded the RPM and other files associated with it, from the Amtelco FTP site, copy them to a temp directory on your system. Now, run the **install** script from a terminal window. This will install the Amtelco XDS driver package and build / load the XDS driver module. If you are running an older version of Redhat, such as 7.1, you may need to install the Openmotif RPM in order to use the XDS user GUI demo **xdsutil**.

The first question asked is what version of Linux Redhat is being used. If Redhat 7.2 and lower is the O/S that you are using, select '**1**'. If Redhat 7.3 (kernel 2.4.x) is the O/S that you are using, select '**2**'. If Redhat 9 is being used, select '**3**'. When your selection is made, select '**Y**' to proceed.

The next question is which cPCI architecture the user desires to use the XDS driver on. The two options are hot-swap* and hot-plug. Hot-swap is when the user will install a third party hot-swap software package (from the CompactPCI chassis vendor). Hot-plug will be when the user will use the XDS board stand-alone and let the XDS driver manage the board removal and replacement of the XDS boards. The install script will prompt the user to enter a 'Y' or a 'y' for hot-swap or an 'N' or 'n' for hot-plug. It will then display the user's choice and ask to confirm it. If the choice displayed is correct, enter a 'Y' or 'y'. If the hot-swap version was selected, the system will need to be restarted before the XDS driver can be used. If hot-plug is chosen, this is not necessary.

*this requires a third-party hot-swap controller, that is provided by the chassis vendor, package to be installed. This driver is known to be supported and has been tested on the Motorola CPX2000 series and the CPX8216T cPCI chassis.

1.3 Software / Low-level Driver Removal Procedure

Should the user desire to remove the driver package, simply log into any X-Windows environment (such as KDE) as "root". Insert the same XDS CD-ROM into the CD-ROM drive, as before. Again, you may need to mount the cdrom. When it is mounted properly, run the command "./remove".

If the user installed the Openmotif package (ie: on Redhat 7.1 or 7.2), they may want to uninstall if they so desire by doing an rpm –e openmotif-2.1.30-1_MLI.i386.rpm

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.

1.0 XDS Source Code Description

All of the source code and makefiles used to build the programs, library, and driver has been included for the user's convenience. They were built by using the native tool set included with Redhat Linux. If any or all of the code is "re-used", the American Tel-A-Systems, Inc. copyright information must be included with it.

2.0 Demos / Utilities

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

A simple demonstration program, **demo1,** is included with this package and can be found in the **/usr/amtelco/h110/demos** directory. It checks for the presence of XDS boards using the **xds_id** XDS library function. If a board is not present, the phrase "board is not present" is displayed on that line.

The program **tstled** is a text-based interactive utility that allows the user to control the AUX LED on some XDS H.110 BRI interface boards*. This program will also display the menu options when it is run.

*** please check the board's technical manual for this capability**

The **tstchs** program is a text-based interactive utility that allows the user to send and receive messages from any XDS board. It also allows the user to extract and insert XDS H.110 boards as well as reset the board. The user options are displayed on the screen at runtime. It also demonstrates the signaling mechanism (SIGPOLL) of the low-level driver.

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 is demonstration to the user on how to open and close a handle to the XDS device driver.

The program **xdsutil** may be used to send and receive messages from any XDS board in the system. The program can be found in the directory **/usr/amtelco/h110/util**.

This program has an easy to use Graphical User Interface. When the program is running, it displays several boxes for transmitting messages, showing a past history of transmitted and received messages, and displaying the port states. Each box shows information for only one board at a time, but the active board may be changed by using the select arrow for each box. To determine which boards are present in the system, click on the "boards" button.

3.0 Downloader

Downloader Program

All XDS H.110 boards are equipped with flash memory, which contains the board's firmware program. The program can be found in the directory **/usr/amtelco/h110/dloader**. New revisions of the program can be downloaded to this memory using the downloader program **lx386dlc**. To use this program, the driver must be started and recognize the board. The program to be downloaded is contained in a file of type .hex. This file will 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> <board number>

where the segment specifier is either a 'C' for the control processor or 'D' for the DSP processor. For example

lx386dlc 258H000B.HEX C 1

would load the control program onto the board in physical slot 1.

4.0 XDS Function Library Description

A function library (libxds110.a) has been provided to access XDS native board functions. These include proprietary functions for use with only XDS H.110 boards. Many of the demos and other programs in this package use this library. When creating a new application, be sure to link in this library in the makefile if you plan to call any functions from it. Details of the functions included in this library may be found in the document *XDS H.110 Library Reference Manual, 258M013.*

This page was intentionally left blank.

XDS Linux Driver
IOCTL Description

This page was intentionally left blank.

Linux Driver

Overview

The XDS Linux driver is designed to provide an interface between XDS boards and applications running under the operating system. A companion library is also provided that allows easy control of a set of XDS boards from a ‘C’ Language program.

The XDS Linux H.110 driver is a loadable driver module. A loadable driver has a wrapper around it, which makes it a loadable module. It acts just like a kernel driver and has the same permissions and entry points. As soon as a program does an open of “/dev/xds” the driver will automatically be loaded. The following files comprise the loadable driver module:

For the purposes of these commands, the board is specified by board_number.

For cPCI/H.110 boards, this number will correspond to the cPCI device number. These numbers will range from 1-30.

The transmit command writes messages directly to the mailbox of the appropriate board. The driver places received messages on one of two queues. Acknowledgments, 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. Each queue is shared by all of the XDS boards in the system. A driver command is provided for reading each queue. The receive queue can handle up to 31 messages while the query queue can handle 7. 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.

The driver can be set to notify the application when a new message has arrived from an XDS board using the driver signaling mechanism (sigpol). This facility eliminates the need for an application to continuously poll the driver.

Commands are provided for reading and writing the dual-ported RAM, which each board shares 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.

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.: fd1 = open("/dev/xds", 0_RDRW);

If the driver can't be opened, a (-1) will be returned. Before closing an application, the user should use the **close** command 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.**

The commands available to an application are:

XMT - xds transmit message function
RCV - xds receive message function
RCV_QUERY - xds receive query response message function
READ_DPRAM - read from the dual-ported RAM
WRITE_DPRAM - write to the dual-ported RAM
PROC_REF - enable signaling on received messages
PROC_UNREF - disable signaling on received messages
XDS_BOARD_ID - obtain information on an XDS board
XDS_RESET - reset specified device (ISA High Density Line Boards, ISA BRI, all H.100, and all H.110 boards)
XDS_SLEEP – put board to sleep function
XDS_RESUME – reactivate a sleeping board function
XDS_GET_BUS_DEVICE_NUM – query the PCI bus and slot number
XDS_GET_BOARD_INFO - get board ID, version, and number of ports
XDS_QUEUE_USER_MSG – copy a message to a message queue

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 number of the board
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 number of the intended board. 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 “LC” or “LR”.

RCV

ioctl(fd1, RCV, 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 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 “LC” or “LR”.

If the queue becomes full, a “FULL QUEUE” message is placed on the queue with the board_number for that message set to -1. 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 = 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 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 ‘V’. 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 -1. 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.

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 at an offset before the beginning of the board
3 - attempt to read past the end of the 2K (ISA) or 8K (PCI) limit

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.

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 to first 256 bytes of an ISA board and the last 256 bytes on a PCI board
3 - attempt to write beyond the end of the 2K (ISA) or 8K (PCI) limit

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.

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 boards.

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_BOARD_ID

ioctl(fd1, XDS_BOARD_ID, NULL);

int fd1;	device file handle returned by **open**
int cmd = XDS_BOARD_ID	obtain information on a board
NULL	no arguments

Purpose
This command is used to identify an XDS board.

Returns
The ioctl function will return the following codes:

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

Comments
This routine scans down the XDS Board list. If it does not find an XDS board, it returns XDS_BOARD_NOT_PRESENT. If the XDS Board is found, the Board's ID, Version Number and number of ports are read in, formatted into a string and passed in the message buffer. Then the routine returns XDS_SUCCESS.

XDS_RESET

ioctl(fd1, XDS_RESET, NULL);

int fd1;	device file handle returned by **open**
int cmd = XDS_RESET	obtain information on a board
NULL	no arguments

Purpose
This command is used to reset an entire board.

Returns
The ioctl function will return the following codes:

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

Comments
This function does not replace the xds_reset_all() function in the XDS library. This will reset entire board. It is valid for the ISA High Density Boards, all ISA BRI boards, all PCI/H.100 boards, and all of the cPCI/H.110 boards.

XDS_SLEEP

ioctl (fd1, XDS_SLEEP, board_number);

int fd1;	Utility device file handle returned by **open**
int cmd = XDS_SLEEP;	Place XDS board into sleep state in preparation to swap out
unsigned char board_number;	Board Number (Physical Slot Number) to put to sleep

Purpose
This command is used to allow a given XDS board to be swapped out for another board of same type 32 port BRI S/T for a bad 32 port BRI S/T board. This allows the replacement of a defective board by a good replacement board while the rest of the system continues to be in use. This command places a designated XDS board into a sleep state. This state turns on the blue LED on the board giving a physical indication of which board to remove. **XDS_RESUME** is used to restart the newly inserted board back into a usable state.

Returns
The ioctl function returns the following codes:

0 - success
1 - no board present
3 - unknown error
4 - illegal argument
-1 - board is not able to be suspended at this time

Comments
To be able to put a board to sleep, the following conditions must be met:

1) The board must be in active state. This means that the board can't be in the sleep state or failed the initialization tests.

2) The board must not be busy. This means that the board is not open by any application. This is why all **XDS_SLEEP** commands must be sent through the XDS utility device "xds0" or the channel assigned to physical slot number 0.

3) The driver must be loaded for that board. This means that the board is not idle for more than idle time threshold. This threshold can be set to infinity through the configuration file.

XDS_RESUME

ioctl (fd1, XDS_RESUME, board_number);

int fd1;	Utility device file handle returned by **open**
int cmd = XDS_RESUME;	Return XDS board from sleep state after swap in
unsigned char board_number;	Board Number (Physical Slot Number) to resume

Purpose
This command is used to allow a given XDS board to be swapped out for another board of same type 32 port BRI S/T for a bad 32 port BRI S/T board. This allows the replacement of a defective board by a good replacement board while the rest of the system continues to be in use. This command returns a designated XDS board from a sleep state entered by **XDS_SLEEP**. This state configures the new board to be like the old board and initializes the board.

Returns
The ioctl function returns the following codes:

0 - success
1 - no board present
3 - unknown error
4 - illegal argument
-1 - board is not able to be reactivated at this time

Comments
This command requires you to send the request through the utility channel (/dev/xds0).

XDS_GET_BUS_DEVICE_NUM

ioctl(fd1, XDS_GET_BUS_DEVICE_NUM, xds_id *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
XDS_ID *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.