CONDUANT

FPDP Digital I/O Board

Installation and User's Guide

Copyright and Trademarks

Printed in the United States.

Version: 8.7

Publication date: July 5, 2007

Table of Contents

License Agreement and Limited Warranty

IMPORTANT. CAREFULLY READ THE TERMS AND CONDITIONS OF THIS AGREEMENT BEFORE USING THE PRODUCT. By installing or otherwise using the StreamStor Product, you agree to be bound by the terms of this Agreement. If you do not agree to the terms of this Agreement, do not install or use the StreamStor Product and return it to Conduant Corporation.

GRANT OF LICENSE. In consideration for your purchase of the StreamStor Product, Conduant Corporation hereby grants you a limited, non-exclusive, revocable license to use the software and firmware which controls the StreamStor Product (hereinafter the "Software") solely as part of and in connection with your use of the StreamStor Product. If you are authorized to resell the StreamStor Product, Conduant Corporation hereby grants you a limited non-exclusive license to transfer the Software only in conjunction with a sale or transfer by you of the StreamStor Product controlled by the Software, provided you retain no copies of the Software and the recipient agrees to be bound by the terms of this Agreement and you comply with the RESALE provision herein.

NO REVERSE ENGINEERING. You may not cause or permit, and must take all appropriate and reasonable steps necessary to prevent, the reverse engineering, decompilation, reverse assembly, modification, reconfiguration or creation of derivative works of the Software, in whole or in part.

OWNERSHIP. The Software is a proprietary product of Conduant Corporation which retains all title, rights and interest in and to the Software, including, but not limited to, all copyrights, trademarks, trade secrets, know-how and other proprietary information included or embodied in the Software. The Software is protected by national copyright laws and international copyright treaties.

TERM. This Agreement is effective from the date of receipt of the StreamStor Product and the Software. This Agreement will terminate automatically at any time, without prior notice to you, if you fail to comply with any of the provisions hereunder. Upon termination of this Agreement for any reason, you must return the StreamStor Product and Software in your possession or control to Conduant Corporation.

LIMITED WARRANTY. This Limited Warranty is void if failure of the StreamStor Product or the Software is due to accident, abuse or misuse.

Hardware: Conduant's terms of warranty on all manufactured products is one year from the date of shipment from our offices. After the warranty period, product support and repairs are available on a fee paid basis. Warranty on all third party materials sold through Conduant, such as chassis, disk drives, PCs, bus extenders, and drive carriers, is passed through with the original manufacturer's warranty. Conduant will provide no charge service for 90 days to replace or handle repair returns on third party materials. Any charges imposed by the original manufacturer will be passed through to the customer. After 90 days, Conduant will handle returns on third party material on a time and materials basis.

Software: The warranty on all software products is 90 days from the date of shipment from Conduant's offices. After 90 days, Conduant will provide product support and upgrades on a fee paid basis. Warranties on all third party software are passed through with the original manufacturer's warranty. Conduant will provide no charge service for 90 days to replace or handle repair returns on third party software. Any charges imposed by the manufacturer will be passed through to the customer.

DISCLAIMER OF WARRANTIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, CONDUANT CORPORATION DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT, WITH REGARD TO THE STREAMSTOR PRODUCT AND THE SOFTWARE.

SOLE REMEDIES. If the StreamStor Product or the Software do not meet Conduant Corporation's Limited Warranty and you return the StreamStor Product and the Software to Conduant Corporation, Conduant Corporation's entire liability and your exclusive remedy shall be at Conduant Corporation 's option, either (a) return of the price paid, if any, or (b) repair or replacement of the StreamStor Product or the Software. Any replacement Product or Software will be warranted for the remainder of the original warranty period.

LIMITATION OF LIABILITIES. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL CONDUANT CORPORATION BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE STREAMSTOR PRODUCT AND THE SOFTWARE. IN ANY CASE, CONDUANT CORPORATION'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL BE LIMITED TO THE AMOUNT ACTUALLY PAID BY YOU FOR THE STREAMSTOR PRODUCT AND THE SOFTWARE. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.

RESALE. If you are authorized to resell the StreamStor Product, you must distribute the StreamStor Product only in conjunction with and as part of your product that is designed, developed and tested to operate with and add significant functionality to the StreamStor Product; you may not permit further distribution or transfer of the StreamStor Product by your end-user customer; you must agree to indemnify, hold harmless and defend Conduant Corporation from and against any claims or lawsuits, including attorneys' fees, that arise or result from the use or distribution of your product; and you may not use Conduant Corporation's name, logos or trademarks to market your product without the prior written consent of Conduant Corporation.

ENTIRE AGREEMENT; SEVERABILITY. This Agreement constitutes the complete and exclusive agreement between you and Conduant Corporation with respect to the subject matter hereof and supersedes all prior written or oral agreements, understandings or communications. If any provision of this Agreement is deemed invalid under any applicable law, it shall be deemed modified or omitted to the extent necessary to comply with such law and the remainder of this Agreement shall remain in full force and effect.

GOVERNING LAW. This Agreement is governed by the laws of the State of Colorado, without giving effect to the choice of law provisions therein. By accepting this Agreement, you hereby consent to the exclusive jurisdiction of the state and federal courts sitting in the State of Colorado.

About This Manual

This manual is intended to serve the following purposes:

* to act as a guide for hardware installation

* to act as a reference for the programmer

It is suggested that you periodically check the Conduant web site for the most recent software updates, application notes, and technical bulletins.

If you are unable to locate the information you need, contact us by phone or submit an electronic request for support. To submit a request for support, go to our website, www.conduant.com. Then click on the support link.

Chapter 1

Introduction

About the FPDP Digital I/O Board

Thank you for purchasing Conduant's FPDP Digital I/O Board. Your purchase includes the FPDP Digital I/O board that plugs into the PCI bus, device drivers, software development tools, and additional utility software.

The FPDP Digital I/O Board can be used to capture one or two FPDP input streams into a standard PC. It can capture 32 bits/4 bytes of parallel data at up to 50 MHz via standard FPDP protocol. For FPDP details, please read the chapter "External Port" in this manual.

The PCI bus is a high performance I/O bus designed for attaching peripheral devices to computer systems. It is found in computing systems from many different manufacturers and is supported by most major operating systems. PCI data acquisition cards (digital oscilloscopes, frame grabbers, telemetry interfaces, etc) are available from many manufacturers to collect data and record it to system memory in real time (as it is collected).

The device drivers and API (Application Programming Interface) provide for a smooth integration of the FPDP Digital I/O Board with the data acquisition device and/or analysis software. Please feel free to offer suggestions and request new features.

What you need to get started

To set up and use the FPDP Digital I/O Board, you will need the following:

- The FPDP Digital I/O Board
- The StreamStor Software Development Kit
- A computer and chassis
- An empty full-length PCI slot or 3U CompactPCI slot (depending on model).
- This manual

Software Programming Choices

The StreamStor Software Development Kit (SDK) includes a Windows DLL library, a Linux function library and drivers providing control and data retrieval functions necessary for using the FPDP Digital I/O Board. Application software can be developed in any environment capable of utilizing these library functions. This includes the various Windows programming languages such as Visual C++ and Visual Basic as well as graphical programming environments such as LabVIEW.

Unpacking

Carefully inspect all shipping packages for any sign of damage. In particular, look for wrinkled or bent corners, holes, or other signs of bad handling or abuse. If you notice any damage to the packaging, immediately open the boxes and inspect the contents for damage. Pay close attention to the components near the area where the packing material was damaged. Report any damage to the carrier and Conduant immediately.

FPDP Digital I/O Board

The FPDP Digital I/O board is shipped in a specially designed antistatic box to prevent electrostatic damage to the board. To avoid damage in handling the board, take the following precautions:

* Ground yourself with a grounding strap or grasp a conductive, grounded object to dissipate any static charge while handling the board.

* Always store the board in its antistatic box when not installed in a computer system.

* Inspect the board carefully before installing in the computer. Notify Conduant immediately if the board appears damaged. Do not install a damaged board into your computer.

* Never touch any exposed connector pins or component leads.

* Avoid bending or twisting the board.

Chapter 2

Installation

Components

Your FPDP Digital I/O Board is shipped with this user manual and installation software (on CD-ROM).

Planning Your Installation

☠ **CAUTION:** ***Please read the entire installation section of this manual before starting to install the FPDP Digital I/O Board. This manual assumes that the user is knowledgeable and comfortable with basic computer cabling, power connections, inserting cards into the PCI bus, and use of the computer operating system. If you are unsure as to how to proceed, please contact Conduant.***

The cables supplied with your system are the maximum recommended length. Avoid the use of longer cables since they may cause intermittent data loss. Avoid pinching or routing over sharp edges to prevent cable damage.

☠ **CAUTION:** ***When removing cables from the FPDP Digital I/O Board, ALWAYS use the ejector tabs to gently free the cables from the board. NEVER pull on the cables to free them from the board.***

Hardware Installation

FPDP Digital I/O Board

Installation requires a PCI slot that can accommodate a full size card and has a card support guide. The following are general instructions for installing your FPDP Digital I/O Board. You should also consult your computer user manual or technical reference for more specific instructions and warnings.

☠ **CAUTION:** **Over flexing the FPDP Digital I/O Board will damage it. Be careful to prevent damage to any components on the backside of the board if you lay the card down.**

1. Turn off and unplug your computer.

2. Remove the top cover or access port to the I/O bus.

3. Remove the expansion slot cover on the back panel of the computer for the slot into which you intend to install the FPDP Digital I/O Board.

4. Insert the FPDP Digital I/O Board into the chosen PCI slot. Gently rock the board to ease it into place. It may be a tight fit but do not force the board into place. Make sure that the card support bracket lines up correctly with the support provided in the computer chassis.

5. Screw the mounting bracket to the back panel of the computer chassis.

Installing the Software

Your FPDP Digital I/O Board was shipped with the Software Development Kit on CD-ROM. This section describes how to install it.

After you have installed the FPDP Digital I/O Board, power up your computer. On Windows systems, when ready, run the `setup.exe` program on the CD-ROM to start the installation process. On Linux systems, refer to the file `linux/docs/install.txt` on the CD-ROM for installation instructions.

Plug and play operating systems such as Windows will detect the installation of a FPDP Digital I/O Board and will attempt to configure it using the hardware plug and play wizard program. The required installation information file for plug and play installation is included on the CD-ROM. Make sure the plug and play wizard includes the CD-ROM drive in its search so that the FPDP Digital I/O Board drivers will be properly installed. You should not cancel the plug and play wizard since this can create hardware conflicts in the system when using the FPDP Digital I/O Board. Note that the `setup.exe` program must still be executed to install the StreamStor SDK onto your system.

The software installation procedure will install the device drivers, library files, example programs and all other components of the SDK onto your system.

The StreamStor SDK does not include software interfaces or drivers used for the control of data acquisition cards made by other manufacturers. However, it does include some sample programs to help in your software development efforts. Other drivers and examples may be available depending on your choice of data acquisition hardware. Contact Conduant support for more information.

Always review the **`readme.html`** file included with the SDK for the latest information not included in this manual. Also, check the Conduant web site periodically for software updates. Software updates may include new features and capabilities as well as important fixes and improved hardware support. Users who do not have access to the Internet can request updates by calling Conduant Technical Support

Chapter 3

Software Development Kit (SDK)

Introduction

Conduant makes it easy for system designers to use the FPDP Digital I/O Board by providing an Application Programming Interface (API) library. This library provides the control software for the FPDP Digital I/O Board in the form of DLLs (Dynamic Link Libraries) for Windows and an archive library for Linux that can be accessed by user application software.

The following pages define the functions provided by the library for controlling the FPDP Digital I/O Board. It is suggested that you periodically check the Conduant Web Site for updates. If you do not have Internet access, feel free to call and ask for technical support. We'll be happy to send you the latest updates.

Software Components

The SDK software components include operating system device drivers, support files, programming libraries and utility programs.

Device Driver

The StreamStor SDK provides device driver support for Windows 2000, Windows XP and Linux operating systems. The drivers are installed automatically by the supplied setup program. On Windows systems, the device driver is named `windrvr6.sys`. The Linux device driver is installed as a kernel module named `windrvr6.o`. On Linux systems, refer to the file `linux/docs/install.txt` on the CD-ROM for driver installation instructions.

Support files

The FPDP Digital I/O Board support files (`sspxf.bib,sspxf-1.bib, ssatap3.bib`) located in the installation directory are required for proper initialization of the board after power-on or reset. On Windows computers, the location of these files is defined by a registry entry created by the installation program that specifies the installation directory where these files are installed by default. This registry setting may be changed if these files are moved to an alternate directory. The registry path is:

"`HKEY_LOCAL_MACHINE\SOFTWARE\Conduant\StreamStor SDK\BibPath`"

On Linux, the environment variable `STREAMSTOR_BIB_PATH` is used to specify this directory path.

The Windows DLL for the StreamStor API is named `xlrapi.dll`. This file is installed into the main directory where FPDP Digital I/O Board files are located. When developing custom applications you must make sure this file is available in a directory where the operating system searches for DLL files.

The Linux library is named `libssapi.a` and all functions are statically linked into the user application from this library archive.

☠ **CAUTION: Modifying the Windows registry incorrectly can irreparably damage your Windows installation.**

Windows Uninstall

The StreamStor SDK can be easily uninstalled in Windows by using the "Add/Remove Software" wizard in the control panel. Simply select "StreamStor SDK" and all installed components will be automatically removed. You can also select "Remove StreamStor SDK" in the StreamStor menu.

Windows Library

The software development kit includes a DLL library for Windows based user applications. The required DLL file is `xlrapi.dll`. The library file `xlrapi.lib` is also included for linking the DLL functions to a user program. The required include files are `xlrapi.h` and `xlrtypes.h`. Only the `xlrapi.h` file needs to be included in a user program. Example programs are included in the SDK. All of the include files are installed automatically by the installation software in the "Include" directory. The library file for linking user programs is installed in the "Lib" directory and the DLL is installed in the StreamStor installation directory.

Linux Uninstall

The StreamStor SDK can be easily uninstalled on Linux by removing the installation directory and the WinDriver module. To do so, enter the following commands as root where `<InstallDir>` is the full path name where the StreamStor SDK is installed and `<WinDriverModule>` is the name of the WinDriver module (i.e., windrvr6).

1. Remove the SDK installation directory as follows:

   ```
   rm –rf <InstallDir>
   ```

   For example, to remove the entire SDK:

   ```
   rm –rf /usr/local/streamstor
   ```

2. Remove the WinDriver module as follows:

   a) Verify that the WinDriver module is not in use.

   b) Unload the WinDriver module by entering:

   ```
   rmmod <WinDriverModule>
   ```

c) Remove the old device node by entering:

```
rm –rf /dev/<WinDriverModule>
```

d) Remove the system startup file (if it exists) by entering:

```
rm –rf /etc/.windriver.rc
```

e) Remove the user startup file (if it exists) by entering:

```
rm –rf $HOME/.windriver.rc
```

Linux Configuration/Test Utilities

Two Linux utility programs are included with the SDK to test the FPDP Digital I/O Board for proper configuration and functionality. If you have just received your FPDP Digital I/O Board or if you are experiencing problems, running these programs will perform configuration and confidence tests to insure that your system is working properly.

Linux programs that use the StreamStor SDK (such as the utilities below) require that the environment variable `STREAMSTOR_BIB_PATH` be set and exported to the SDK directory containing the StreamStor `*.bib` files. For example:

```
STREAMSTOR_BIB_PATH=/usr/local/streamstor/linux/bib

export STREAMSTOR_BIB_PATH
```

The program `ssopen` simply attempts to open the StreamStor and then closes it. To execute it:

1. `cd <InstallDir>/linux/util`

2. `./ssopen`

If your system can communicate with the StreamStor board, you should see this output (note that if your system has no drives, you can ignore the values displayed for DriveFail and DriveFailNumber):

```
Attempting to open StreamStor...
StreamStor opened successfully!
Device Status:
   SystemReady-> 1
   MonitorReady-> 0
   DriveFail-> 0
   DriveFailNumber-> 0
   SysError-> 0
   SysErrorCode-> 0
   CtlrError-> 0
```

Linux Library

When the SDK is installed on a Linux system, a static function library is installed named `libssapi.a.` It contains all the StreamStor API functions. The required header files are `xlrapi.h` and `xlrtypes.h`. Only the `xlrapi.h` file must be included by the user application. The library must be supplied to the linker to create a final executable program. An example C program that shows how to call the SDK library functions and a corresponding `gcc` makefile are in the directory `<InstallDir>/Linux/example`.

API Functions

Chapter 4 describes each API command. Table 1 is a summary of the API functions.

Table 1 - API Function Summary

FUNCTION	DESCRIPTION
XLRApiVersion	Report version of API library in use.
XLRArmFPDP	Move StreamStor from a ready to record state when a synch pulse is received.
XLRBindInputChannel	Binds a channel for input into FPDP Digital I/O Board.
XLRBindOutputChannel	Binds a channel for output from FPDP Digital I/O Board.
XLRCardReset	Reset an FPDP Digital I/O Board card.
XLRClearChannels	Unbinds all input and output channels.
XLRClose	Close device and release exclusive access.
XLRDeviceFind	Report number of FPDP Digital I/O Boards present in system.
XLRGetBaseAddr	Get base address (physical) of FPDP Digital I/O Board data window.
XLRGetBaseRange	Get size of FPDP Digital I/O Board data window.
XLRGetDeviceInfo	Retrieve hardware configuration information.
XLRGetDeviceStatus	Get status of device.
XLRGetErrorMessage	Get error string for supplied error code.
XLRGetFIFOLength	Return the amount of data in the FIFO.
XLRGetLastError	Return error code of last failure.
XLRGetMode	Return the input/output mode of the board.
XLRGetSystemAddr	Return the kernel address of the FPDP Digital I/O Board data window.

XLRGetVersion	Report version of the FPDP Digital I/O Board firmware components.
XLRGetWindowAddr	Get user virtual address of the FPDP Digital I/O Board data window.
XLROpen	Open the device for exclusive access.
XLRReadFifo	Read data during a FIFO operation.
XLRRecord	Start FIFO data transfer.
XLRReset	Reset and close an open device.
XLRSelectChannel	Select the channel for subsequent commands.
XLRSetFPDPMode	Set the operating mode of the FPDP data port.
XLRSetMode	Set input/output mode of the board.
XLRSetPortClock	Set the clock speed of the external port.
XLRSetReadLimit	Set the range of any read accesses performed from an outside bus master.
XLRStop	Stop recording.

Data Structures

StreamStor API functions use the following structures. Refer to the end of the Function Reference section for details on each structure and its members.

`S_DEVINFO`	-	Device info parameters
`S_DEVSTATUS`	-	Device status flags
`S_READDESC`	-	Parameters defining read/write requests
`S_XLRSWREV`	-	Various device version strings

Chapter 4

Function Reference

XLRApiVersion

Syntax:

```
char *XLRApiVersion( char *versionstring )
```

Description:

`XLRApiVersion` returns the API version as a string formatted as a *major.minor* version number.

Parameters:

- *`versionstring`* is a pointer to a character string to hold the returned version. It must be of minimum length `XLR_VERSION_LENGTH.`

Return Value:

The API version is returned in *`versionstring`*.

Usage:

```
/* Read XLR API version into string */
char xlrstring[XLR_VERSION_LENGTH];

XLRApiVersion( xlrstring );
printf( “StreamStor API version is %s”, xlrstring );
```

See Also:

`XLRGetVersion.`

XLRArmFPDP

Syntax:

`XLR_RETURN_CODE XLRArmFPDP( SSHANDLE` *`xlrDevice`* `)`

Description:

`XLRArmFPDP` moves StreamStor from a ready to record state, to recording when an FPDP SYNC* pulse is received. StreamStor must already be in record mode, and `SS_OPT_FPDPSYNCARM` must be set. If no SYNC* pulse is received, no data will be recorded.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;

xlrStatus = XLROpen( 1, &xlrDevice );
xlrStatus = XLRSetMode(xlrDevice, SS_MODE_PASSTHRU);

// … Bind and configure channels as desired …
// … Configure FPDP as desired …

if ( XLRRecord ( xlrDevice, 0, 0 ) != XLR_SUCCESS ) {
  XLRClose( xlrDevice );
  exit(1);
}
if( XLRArmFPDP( xlrDevice ) != XLR_SUCCESS )
{
  XLRClose (xlrDevice );
  exit(1);
}

// Waiting for SYNC pulse – data will be recorded to disk as soon
// as SYNC is received.
```

See Also:

`XLRSetFPDPOption, XLRRecord` and `XLRAppend.`

XLRBindInputChannel

Syntax:

```
XLR_RETURN_CODE XLRBindInputChannel( SSHANDLE xlrDevice, UINT
channel )
```

Description:

XLRBindInputChannel binds a channel for input INTO the Digital FPDP I/O Board. In other words, "input" is relative to the board. To use a channel, that channel must be bound to the board via this command.

XLRClearChannels must be called to unbind the channel(s) before calling XLRBindInputChannel.

Details on channel selection are in the "Channel Description and Selection" chapter of this manual.

Parameters:

- *xlrDevice* is the device handle returned from a previous call to XLROpen.
- *channel* is the channel number to bind – this is card specific. Valid channels are:
    - 0 – the PCI Bus
    - 30 – the top FPDP connector
    - 31 – the front FPDP connector

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Note: **CHANGING MODES CLEARS ALL INPUT AND OUTPUT CHANNELS. CHANNELS MUST BE BOUND AFTER THE MODE IS SELECTED.**

Usage:

```
SSHANDLE          xlrDevice;

xlrStatus = XLROpen( 1, &xlrDevice );
xlrReturnCode = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );

xlrStatus = XLRClearChannels( xlrDevice );

// For input over the PCI bus, bind to channel zero.
xlrStatus = XLRBindInputChannel( xlrDevice, 0 );
if( xlrStatus != XLR_SUCCESS )
{
  return(1);
}
```

See Also:

`XLRClearChannels`, `XLRBindOutputChannel`, and `XLRSelectChannel`.

XLRBindOutputChannel

Syntax:

```
XLR_RETURN_CODE XLRBindOutputChannel( SSHANDLE xlrDevice, UINT
channel )
```

Description:

`XLRBindOutputChannel` binds a channel for output FROM the FPDP Digital I/O Board. In other words, "output" is relative to FPDP Digital I/O Board. To read from FIFO or send data over a particular channel, that channel must be bound to the board via this command. Only one channel at a time can be selected to output data.

`XLRClearChannels` must be called to unbind the channel(s) before calling `XLRBindOutputChannel.`

Details on channel selection are in the "Channel Description and Selection" chapter of this manual.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`channel`* is the channel number to bind – this is card specific. Valid channels are:
  - 0 – the PCI Bus
  - 30 – the top FPDP connector
  - 31 – the front FPDP connector

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Note: **CHANGING MODES CLEARS ALL INPUT AND OUTPUT CHANNELS. CHANNELS MUST BE BOUND AFTER THE MODE IS SELECTED.**

Usage:

```
SSHANDLE          xlrDevice;
S_READDESC        readDesc;

xlrStatus = XLROpen( 1, &xlrDevice );
xlrReturnCode = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );
xlrStatus = XLRClearChannels( xlrDevice );
xlrStatus = XLRSelectChannel ( xlrDevice, 0);

// Bind the PCI Bus channel for output.
xlrStatus = XLRBindOutputChannel( xlrDevice, 0 );
if( xlrStatus != XLR_SUCCESS )
{
  return(1);
}
```

See Also:

XLRClearChannels, XLRBindInputChannel, and XLRSelectChannel.

XLRCardReset

Syntax:

```
XLR_RETURN_CODE XLRCardReset( UINT index )
```

Description:

`XLRCardReset` will attempt to reset an FPDP Digital I/O Board and re-initialize the hardware and firmware. This function should be used only as a last resort.

Parameters:

- *`index`* is the FPDP Digital I/O Board index number.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
xlrReturnCode = XLRCardReset( 1 );
```

See Also:

`XLROpen` and `XLRReset`.

XLRClearChannels

Syntax:

`XLR_RETURN_CODE XLRClearChannels( SSHANDLE *xlrDevice* )`

Description:

`XLRClearChannels` unbinds all input and output channels from the FPDP Digital I/O Board. The FPDP Digital I/O Board cannot be reading or writing, and new input and output channels must be bound before any data transfer operation is started. `XLRClearChannels` must be called before calling `XLRBindInputChannel` or `XLRBindOutputChannel`.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

// Open the device
xlrStatus = XLROpen( 1, &xlrDevice );
      …
xlrStatus = XLRClearChannels( xlrDevice );
xlrStatus = XLRBindInputChannel( xlrDevice, 0 );
      …
// Close device before exiting.
XLRClose( xlrDevice );
```

See Also:

`XLRBindInputChannel`, `XLRBindOutputChannel`, and `XLRSelectChannel`.

XLRClose

Syntax:

`void XLRClose( SSHANDLE *xlrDevice* )`

Description:

`XLRClose` closes the FPDP Digital I/O Board. This should be called before exiting an application that has opened an FPDP Digital I/O Board with `XLROpen`. No other application can open the FPDP Digital I/O Board until this function has been called.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

none

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

// Open the device.
xlrStatus = XLROpen( 1, &xlrDevice );
.
.
.
// Close device before exiting.
XLRClose( xlrDevice );
```

See Also:

`XLROpen.`

XLRDeviceFind

Syntax:

```
UINT XLRDeviceFind( )
```

Description:

`XLRDeviceFind` searches the PCI bus(es) and returns the number of FPDP Digital I/O Boards present in the system.

Parameters:

None.

Return Value:

This function returns the number of FPDP Digital I/O Boards in the system. If the driver has not been installed properly, this function returns zero.

Usage:

```
UINT  NumCards;

if( NumCards = XLRDeviceFind() )
{
   // There are FPDP Digital I/O Boards on this system.
   printf(“FPDP Digital I/O Boards found: %d\n”, NumCards );
}
else
{
   // No FPDP Digital I/O Boards on the system.
   printf(“No FPDP Digital I/O Boards detected!\n”);
}
```

See Also:

`XLROpen.`

XLRGetBaseAddr

Syntax:

`ULONG XLRGetBaseAddr( SSHANDLE *xlrDevice* )`

Description:

`XLRGetBaseAddr` returns the physical address of the recording data window. This address can be used to program PCI hardware devices for direct card-to-card data transfer. The address returned from this function is NOT a valid user address.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

This function returns the physical PCI address as a 32 bit unsigned integer.

Usage:

```
ULONG              xlrAddress;
SSHANDLE           xlrDevice;
XLR_RETURN_CODE    xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );
if( xlrStatus != XLR_SUCCESS )
{
   // Error opening FPDP Digital I/O Board
}
else
{
   xlrAddress = XLRGetBaseAddr( xlrDevice );
}
```

XLRGetBaseRange

Syntax:

ULONG XLRGetBaseRange(SSHANDLE *xlrDevice*)

Description:

XLRGetBaseRange returns the size (in bytes) of the FPDP Digital I/O Board data window. This range of addresses is intended to be used by hardware transferring data that cannot be programmed to write with a non-incrementing address. Note that the address used to write to FPDP Digital I/O Board does not effect the storage location of the data; FPDP Digital I/O Board always stores data sequentially in the order it is written regardless of the address.

Parameters:

- *xlrDevice* is the device handle returned from a previous call to XLROpen.

Return Value:

This function returns the window size in bytes.

Usage:

```
ULONG             xlrAddress, xlrRange;
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );
if( xlrStatus != XLR_SUCCESS )
{
   // Error opening FPDP Digital I/O Board
}
else
{
   xlrAddress = XLRGetBaseAddr( xlrDevice );
   xlrRange = XLRGetBaseRange( xlrDevice );
}
// DMA Hardware may now be programmed to write to any address from
// xlrAddress to (xlrAddress + xlrRange)
```

XLRGetDeviceInfo

Syntax:

`XLR_RETURN_CODE XLRGetDeviceInfo( SSHANDLE` *`xlrDevice`*`, PS_DEVINFO` *`pDevInfo`* `)`

Description:

`XLRGetDeviceInfo` retrieves information from the FPDP Digital I/O Board about its physical configuration.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`pDevInfo`* is a pointer to an `S_DEVINFO` structure.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE            xlrDevice;
S_DEVINFO           devInfo;
XLR_RETURN_CODE     xlrReturn;

xlrReturn = XLROpen( 1, &xlrDevice );
if( xlrReturn != XLR_SUCCESS )
      return(1);
xlrReturn = XLRGetDeviceInfo( xlrDevice, &devInfo );
if( xlrReturn != XLR_SUCCESS )
      return(1);
printf(“FPDP Digital I/O Board serial number is: %d”, devInfo.SerialNum );
```

XLRGetDeviceStatus

Syntax:

XLR_RETURN_CODE XLRGetDeviceStatus(SSHANDLE *xlrDevice*,
PS_DEVSTATUS *pDevStatus*)

Description:

XLRGetDeviceStatus retrieves the status of the FPDP Digital I/O Board.

Parameters:

- *xlrDevice* is the device handle returned from a previous call to XLROpen.
- *pDevStatus* is a pointer to an S_DEVSTATUS structure.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
S_DEVSTATUS       devStatus;
XLR_RETURN_CODE   xlrReturn;

xlrReturn = XLROpen( 1, &xlrDevice );
if( xlrReturn != XLR_SUCCESS )
      return(1);
xlrReturn = XLRGetDeviceStatus( xlrDevice, &devStatus );
if( xlrReturn != XLR_SUCCESS )
      return(1);
if( devStatus.FifoFull )
      printf(“Fifo is full.”);
else
      printf(“Fifo is not full.”);
```

XLRGetErrorMessage

Syntax:

`XLR_RETURN_CODE XLRGetErrorMessage(char *string, XLR_ERROR_CODE err)`

Description:

`XLRGetErrorMessage` returns the error message of the most recent API failure.

Parameters:

- *`string`* is a pointer to a string to accept the error message of at least `XLR_ERROR_LENGTH` size.
- *`err`* is an error code returned from `XLRGetLastError.`

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrHandle;
S_DEVSTATUS       devStatus;
XLR_RETURN_CODE   xlrReturn;
XLR_ERROR_CODE    xlrError;
char              temp[XLR_ERROR_LENGTH];

xlrStatus = XLROpen( 1, &xlrDevice );
      …
xlrReturn = XLRGetDeviceStatus( xlrDevice, &devStatus );
if( xlrReturn != XLR_SUCCESS )
{
   Printf ( “Cannot get device status.\n”);
   xlrError = XLRGetLastError(  );
   XLRGetErrorMessage( temp, xlrError );
   printf( “Error message: %s\n”, temp );
   exit(1);
}
```

See Also:

`XLRGetLastError.`

XLRGetFIFOLength

Syntax:

`DWORDLONG XLRGetFIFOLength( SSHANDLE *xlrDevice* )`

Description:

`XLRGetFIFOLength` returns the amount of data currently in the FIFO. This function is only valid when the FPDP Digital I/O Board is in pass through mode (`SS_MODE_PASSTRHU`). If the FPDP Digital I/O Board is not in pass through mode, or is not currently moving data, `XLRGetFIFOLength` will return 0.

If you retrieve the error code after the `XLRReadFifo` call, you may get the `XLR_ERR_EMPTY` ("No Data") error. This indicates that `XLRReadFifo` was not able to return the requested number of bytes. This does not necessarily indicate an error condition. Rather, it may indicate that transfer of data has ended normally but that the last transfer into the FIFO did not fill it to capacity. In this case, you may want to call `XLRGetFIFOLength` to get the number of bytes left in the FIFO and then call `XLRReadFifo` with that length. See Example 1 in the "Channel Description and Selection" chapter of this manual.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

Usage:

```
SSHANDLE         xlrDevice;
DWORDLONG        length = 0;

xlrStatus = XLROpen( 1, &xlrDevice );
xlrReturnCode = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );

// … Bind and configure channels as desired …
// … Configure FPDP as desired …

// Start data transfer by calling XLRRecord.
xlrReturnCode = XLRRecord( xlrDevice, 0, 0 );

length = XLRGetFIFOLength( xlrDevice );
```

See Also:

`XLRSetMode, XlrReadFIFO` and `XLRGetLength.`

XLRGetLastError

Syntax:

```
XLR_ERROR_CODE XLRGetLastError( void )
```

Description:

`XLRGetLastError` returns the error code of the most recent API failure.

Parameters:

None.

Return Value:

This function returns the error code (see Appendix A).

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;
XLR_ERROR_CODE    xlrError;
char              temp[XLR_ERROR_LENGTH];

xlrStatus = XLROpen( 1, &xlrDevice );
if( xlrStatus != XLR_SUCCESS )
{
   printf ( “Cannot open FPDP Digital I/O Board one.\n” );
   xlrError = XLRGetLastError(  );
   XLRGetErrorMessage( temp, xlrError );
   printf( “Error Message:  %s\n”, temp );
   exit(1);
}
```

See Also:

`XLRGetErrorMessage.`

XLRGetMode

Syntax:

`XLR_RETURN_CODE XLRGetMode( SSHANDLE` *`xlrDevice`*`, SSMODE` *`pMode`* `)`

Description:

`XLRGetMode` returns the input/output path (or "port mode") on the FPDP Digital I/O Board where the mode was previously set with the `XLRSetMode` command.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`pmode`* is a pointer to an `SSMODE` variable that will receive the mode.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrHandle;
XLR_RETURN_CODE   xlrStatus;
SSMODE            portMode;

xlrStatus = XLROpen( 1, &xlrDevice );
xlrStatus = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );
xlrStatus = XLRGetMode(xlrDevice, &portMode);

if ( portMode == SS_MODE_PASSTHRU )
{
   printf (“FPDP Digital I/O Board is in PASSTHRU mode.\n”);
}
```

See Also:

`XLRSetMode.`

XLRGetSystemAddr

Syntax:

```
ULONG XLRGetSystemAddr( SSHANDLE xlrDevice )
```

Description:

`XLRGetSystemAddr` returns the kernel address of the recording data window. This address can be used from device drivers or other kernel level software. The address returned from this function is NOT a valid user address.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

This function returns the physical PCI address as a 32 bit unsigned integer.

Usage:

```
ULONG             xlrAddress;
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );
if( xlrStatus != XLR_SUCCESS )
{
   // Error opening FPDP Digital I/O Board
}
else
{
   xlrAddress = XLRGetSystemAddr( xlrDevice );
}
```

XLRGetVersion

Syntax:

`XLR_RETURN_CODE XLRGetVersion( SSHANDLE *xlrDevice*, PS_XLRSWREV *pVersion* )`

Description:

`XLRGetVersion` gets the API and firmware version information from an FPDP Digital I/O Board.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`pVersion`* is a pointer to an `S_XLRSWREV` structure to hold the version strings returned.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE     xlrDevice;
S_XLRSWVER   swVersion;
char         temp[XLR_ERROR_LENGTH];

xlrStatus = XLROpen( 1, &xlrDevice );
      …
xlrReturnCode = XLRGetVersion( xlrDevice, &swVersion );
if( xlrReturnCode != XLR_SUCCESS )
{
   xlrError = XLRGetLastError(  );
   XLRGetErrorMessage( temp, xlrError );
   printf( “%s\n”, temp );
   exit(1);
}
printf(“Firmware version: %s\n”, swVersion.FirmwareVersion );
```

See Also:

`XLRApiVersion.`

XLRGetWindowAddr

Syntax:

PULONG XLRGetWindowAddr(SSHANDLE *xlrDevice*)

Description:

XLRGetWindowAddr returns the user virtual address of the recording data window. This address can be used to directly write data to the FPDP Digital I/O Board from a user program.

Parameters:

- *xlrDevice* is the device handle returned from a previous call to XLROpen.

Return Value:

This function returns a pointer to the data window mapped into the user virtual address space.

Usage:

```
PULONG            xlrAddress;
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrReturn;

xlrReturn = XLROpen( 1, &xlrDevice );
if( xlrReturn == XLR_SUCCESS )
{
   xlrAddress = XLRGetWindowAddr( xlrDevice );
   *xlrAddress = someData;

   /* someData has been written to the FPDP Digital I/O Board.
    * Note that xlrAddress does not need to be incremented
    * for subsequent writes.
    */
}
```

XLROpen

Syntax:

XLR_RETURN_CODE XLROpen(UINT *devIndex*, SSHANDLE **pXlrHandle*)

Description:

XLROpen opens an FPDP Digital I/O Board and initializes the hardware and firmware. The device is transitioned to system ready state if required. This function must be called before any other API function. After successful completion of this function, the handle pointed to by *pXlrHandle* can be used for all subsequent API calls.

NOTE: You should call XLRClose even if XLROpen returns XLR_FAIL.

Parameters:

- *devIndex* identifies the desired FPDP Digital I/O Board to open when multiple FPDP Digital I/O Boards are in use. Use 1 for single board systems. Use XLRDeviceFind to get the number of FPDP Digital I/O Boards installed.
- *pXlrHandle* is a pointer to a system handle for initialization. Successful completion loads this parameter with a valid handle to the hardware device to use in subsequent API calls. **pXlrHandle* is assigned the value INVALID_SSHANDLE on failure.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE           xlrHandle;
XLR_RETURN_CODE    xlrReturnCode;
ULONG              xlrError;
char               errString[XLR_ERROR_LENGTH];

xlrReturnCode = XLROpen( 1, &xlrHandle );
  // … Do FPDP Digital I/O work …
XLRClose( xlrHandle );
```

See Also:

XLRClose and XLRDeviceFind.

XLRReadFifo

Syntax:

```
XLR_RETURN_CODE XLRReadFifo( SSHANDLE xlrDevice, PULONG Buffer,
ULONG Length, BOOLEAN Direct )
```

Description:

`XLRReadFifo` reads data from the FPDP Digital I/O Board during a FIFO operation. Data can continue to be read with this function until the FIFO is empty or `XLRStop` is called. Note that `XLRRecord` must be called prior to calling `XLRReadFifo`. A second call to `XLRStop` is required to take the board out of record mode.

See Example 1 in the "Channel Description and Selection" chapter of this manual to see how this command can be used.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`Buffer`* is the address of the buffer to receive the read data.
- *`Length`* is the length of data to transfer in bytes.
- *`Direct`* is a flag that indicates if the supplied Buffer address is a physical address for direct transfer. For normal transfer to a user memory buffer this flag should be FALSE (0).

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;
ULONG             myBuffer[40000];

xlrStatus = XLROpen( 1, &xlrDevice );
xlrStatus = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );
// … Bind and configure channels as desired …
// … Configure FPDP as desired …

xlrStatus = XLRRecord( xlrDevice, 0, 0 );
      …
xlrStatus = XLRReadFifo(xlrDevice, myBuffer, sizeof(myBuffer), FALSE);

// Stop the transfer of data.
xlrStatus = XLRStop( xlrDevice );

// Take the board out of record mode.
xlrStatus = XLRStop( xlrDevice );
```

See Also:

XLRGetFifoLength, XLRRecord, XLRSetMode and XLRSetFPDPMode.

XLRRecord

Syntax:

`XLR_RETURN_CODE XLRRecord( SSHANDLE` *`xlrDevice`*`, BOOLEAN` *`WrapEnable`*`, SHORT` *`ZoneRange`* `)`

Description:

`XLRRecord` initiates the transfer of data from the FIFO. When called, the FPDP Digital I/O board will start reading data or will start writing over a previously specified channel.

You must call `XLRSetMode` to put the FPDP Digital I/O board in pass through mode prior to calling `XLRRecord`.

See the examples in the "Channel Description and Selection" chapter of this manual.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`WrapEnable`* is ignored.
- *`ZoneRange`* is ignored.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE xlrDevice;
XLR_RETURN_CODE   xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );
xlrStatus = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );

// … Bind and configure channels as desired …
// … Configure FPDP as desired …
      …
// Start data transfer.
xlrStatus = XLRRecord( xlrDevice, 0, 0 );
      …
// End data transfer.
XLRStop( xlrDevice );
```

See Also:

`XLRWrite, XLRWriteData, XLRReadFifo, XLRGetFIFOLength.`

XLRReset

Syntax:

`XLR_RETURN_CODE XLRReset( SSHANDLE *xlrDevice* )`

Description:

`XLRReset` will attempt to reset an FPDP Digital I/O Board and re-initialize the hardware and firmware. This function should be used only as a last resort.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;
XLR_RETURN_CODE   xlrReturnCode;

xlrStatus = XLROpen( 1, &xlrDevice );
      …
xlrReturnCode = XLRReset( xlrDevice );
```

See Also:

`XLRCardReset.`

XLRSelectChannel

Syntax:

`XLR_RETURN_CODE XLRSelectChannel(SSHANDLE` *`xlrDevice`*`, UINT` *`channel`*`)`

Description:

`XLRSelectChannel` selects the channel that future commands will operate on. A channel can be selected and operated on regardless of whether or not it is bound.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`channel`* is the number of the channel to select, where *`channel`* is one of:
  - 0 – to select the PCI channel.
  - 30 – to select the FPDP channel over the FPDP top connector.
  - 31 – to select the FPDP channel over the FPDP front connector.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );.
xlrStatus = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );
xlrStatus = XLRClearChannels( xlrDevice );

// Bind TOP port (connector) as input channel.
xlrStatus = XLRBindInputChannel( xlrDevice, 30 );

// Select and set FPDP options on TOP port.
xlrStatus = XLRSelectChannel( xlrDevice, 30 );
xlrStatus =
  XLRSetFPDPMode( xlrDevice, SS_FPDP_RECVMASTER,  SS_OPT_FPDPNRASSERT );
```

See Also:

`XLRClearChannels`, `XLRBindInputChannel`, `XLRBindOutputChannel`, `XLRSetMode` and `XLRSetFPDPMode`.

XLRSetFPDPMode

Syntax:

`XLR_RETURN_CODE XLRSetFPDPMode( SSHANDLE` *`xlrDevice,`* `FPDPMODE` *`Mode,`* `FPDPOP` *`option`* `)`

Description:

`XLRSetFPDPMode` is used to set the operating mode of the external port. For details on using FPDP, refer to the "External Port" chapter of this manual.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`Mode`* is a constant that defines the mode of operation. Possible values are:
  - `SS_FPDP_RECV` – Sets the FPDP Digital I/O Board port to FPDP/R mode.
  - `SS_FPDP_RECVMASTER` – Sets the FPDP Digital I/O Board to FPDP/RM mode.
  - .`SS_FPDP_XMIT` – Sets the FPDP Digital I/O Board to FPDP/T mode.
  - `SS_FPDP_XMITMASTER` – Sets the FPDP Digital I/O Board to FPDP/TM mode.
  - `SS_FPDP_RECVMASTER_CLOCKS` – Sets the FPDP Digital I/O Board to FPDP/RMCM mode.
- *`option`* is used to specify various options that modify the operation of the FPDP port. Possible values are:
  - `0` (zero) – Disables all options.
  - `SS_OPT_FPDPNRASSERT` – Assert the "Not ready" signal on the FPDP bus when not transferring data. This prevents data flow on FPDP when the FPDP Digital I/O Board is not in transfer mode.
  - `SS_OPT_FPDPSTROB` – Enables the data strobe clock (TTL strobe signals). Default is pstrob clock (PECL strobe signals).
  - `SS_OPT_FPDPNOPLL` – This option should be set when the FPDP clock will be operating at less than 10 MHz while transferring data. Default is this option is off.

- SS_OPT_FPDPSINGLEFRAME – When transferring data in, the FPDP port on the FPDP Digital I/O Board will wait for an FPDP SYNC* pulse before any data on the FPDP interface is placed in the FIFO. All valid data prior to and during the first cycle of the SYNC* pulse will be discarded. When transferring data out, the FPDP Digital I/O Board will generate a single cycle FPDP SYNC* pulse a few clocks prior to asserting DVALID* and transmitting data. The default state of this option is off.
- SS_OPT_FPDPSYNCARM - This option is used when transferring data in. When set, the FPDP Digital I/O Board will be configured and ready to transfer data into the FIFO. However, data will not be transferred until XLRArmFPDP is called and an FPDP SYNC* pulse is received.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );
     …
// Configure channel 30(the top connector) for FPDP.
xlrStatus = XLRSelectChannel( xlrDevice, 30 );

// Example 1: Set the FPDP port mode to FPDP/R and use the default
// options on channel 30.
xlrStatus = XLRSetFPDPMode( xlrDevice, SS_FPDP_RECV, 0 );

// Example 2: Enable the data strobe clock and "Not Ready"
// assert options on channel 30.
xlrStatus = XLRSetFPDPMode( xlrDevice,
   FPDP_RECV, SS_OPT_FPDPSTROB|SS_OPT_NRASSERT );

// Example 3: Enable data strobe clock on channel 30.
xlrStatus = XLRSetFPDPMode( xlrDevice, FPDP_RECV, SS_OPT_FPDPSTROB );

// Configure channel 31(the front connector) for FPDP.
xlrStatus = XLRSelectChannel( xlrDevice, 31 );

// Example 4: Enable FPDP front connector to FPDP/RM mode
// on channel 31.
xlrStatus = XLRSetFPDPMode( xlrDevice, SS_FPDP_RECVMASTER, 0 );
```

See Also:

XLRSetMode and XLRSelectChannel.

XLRSetMode

Syntax:

`XLR_RETURN_CODE XLRSetMode( SSHANDLE` *`xlrDevice,`* `SSMODE` *`Mode`* `)`

Description:

`XLRSetMode` is used to set the input/output path and functionality of the FPDP Digital I/O Board.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`Mode`* is a constant that defines the mode of operation. The only valid mode for this release is `SS_MODE_PASSTHRU`. In this mode, data comes in on one channel and then is sent out ("passed through") a different channel.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Note: **CHANGING MODES CLEARS ALL INPUT AND OUTPUT CHANNELS. CHANNELS MUST BE BOUND AFTER THE MODE IS SELECTED.**

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;

xlrStatus = XLROpen( 1, &xlrDevice );

xlrReturnCode = XLRSetMode( xlrDevice, SS_MODE_PASSTHRU );

// Channels must be cleared prior to binding.
xlrStatus = XLRClearChannels( xlrDevice );

// Input will be done over the PCI Bus, which is channel zero.
xlrStatus = XLRBindInputChannel( xlrDevice, 0 );
      -
// Select channel zero.
xlrStatus = XLRSelectChannel( xlrDevice, 0 );

// Begin transfer of data over channel zero.
xlrStatus = XLRRecord( xlrDevice, 0, 0 );
```

See Also:

`XLRGetMode, XLRSetFPDPMode, XLRBindInputChannel` and `XLRBindOutputChannel.`

XLRSetPortClock

Syntax:

`XLR_RETURN_CODE XLRSetPortClock( SSHANDLE` *`xlrDevice,`* `UINT` *`clock`* `)`

Description:

`XLRSetPortClock` is used to set the operating frequency of the external port if applicable.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`clock`* is a constant that defines the desired clock frequency. Possible values are defined in the header file xlrapi.h as `SS_PORTCLOCK_`*`x`*`MHz` values.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
// Set the external clock frequency
SSHANDLE xlrDevice;

xlrStatus = XLROpen( 1, &xlrDevice );
      …

xlrReturnCode = XLRSetPortClock( xlrDevice, SS_PORTCLOCK_40MHZ );
```

XLRSetReadLimit

Syntax:

`XLR_RETURN_CODE XLRSetReadLimit( SSHANDLE` *`xlrDevice,`* `ULONG` *`Limit`* `)`

Description:

`XLRSetReadLimit` sets the size of the address range an outside device will be using when reading data. This is required to prevent StreamStor hardware from discarding cached read data when an external DMA engine recycles to a new starting read address on the PCI bus.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.
- *`Limit`* is the address range size that the outside device will use when reading from StreamStor during playback operations.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE xlrDevice;
ULONG  DMA_size = 0x2000;
PULONG pBuffer;
PULONG pSSAddr;

xlrStatus = XLROpen( 1, &xlrDevice );
      …
xlrReturnCode = XLRSetReadLimit( xlrDevice, DMA_size );
      …
// Outside device can now DMA data from StreamStor within an
// address range size defined by DMA_size.
// The following simulates this by reading from StreamStor to memory.
pBuffer = (PULONG)malloc(DMA_size);
pSSAddr = XLRGetWindowAddr( xlrDevice );

for( j = 0; j < loops; j++ )
{
   for( i = 0; i < DMA_size; i += 4 )
   {
      *pBuffer++ = *pSSAddr++;
   }
}
```

XLRStop

Syntax:

```
XLR_RETURN_CODE XLRStop( SSHANDLE xlrDevice )
```

Description:

`XLRStop` will halt a FIFO operation. This function should always be used to end FIFO transfers that were started with `XLRRecord`.

Parameters:

- *`xlrDevice`* is the device handle returned from a previous call to `XLROpen`.

Return Value:

On success, this function returns XLR_SUCCESS.
On failure, this function returns XLR_FAIL.

Usage:

```
SSHANDLE          xlrDevice;
XLR_RETURN_CODE   xlrStatus;


xlrStatus = XLROpen( 1, &xlrDevice );
       …
xlrStatus = XLRStop( xlrDevice );
```

See Also:

`XLRRecord`.

Structure S_DEVINFO

```
typedef struct _DEVINFO
{
   char     BoardType[XLR_MAX_NAME];
   UINT     SerialNum;
   UINT     NumDrives;
   UINT     NumBuses;
   UINT     TotalCapacity;
   UINT     MaxBandwidth;
   UINT     PciBus;
   UINT     PciSlot;
   UINT     NumExtPorts;
}S_DEVINFO, *PS_DEVINFO;
```

Purpose

This structure is used by the `XLRGetDeviceInfo` function to return data about the FPDP Digital I/O Board configuration.

Members

- *BoardType* - a string holding the board type (model name) of the FPDP Digital I/O Board.
- *SerialNum* - the serial number of the FPDP Digital I/O Board.
- *NumDrives* – not used.
- *NumBuses* – not used.
- *TotalCapacity* – not used.
- *MaxBandwidth* – not used.
- *PciBus* - the PCI bus number to which the FPDP Digital I/O Board is connected.
- *PciSlot* – the PCI slot number to which the FPDP Digital I/O Board is connected.
- *NumExtPorts* – the number of external ports.

Structure S_DEVSTATUS

```
typedef struct _DEVSTATUS
{
   BOOLEAN SystemReady;
   BOOLEAN BootmonReady;
   BOOLEAN Recording;
   BOOLEAN Playing;
   BOOLEAN Reserved1;
   BOOLEAN Reserved2;
   BOOLEAN Reserved3;
   BOOLEAN Reserved4;
   BOOLEAN RecordActive[XLR_MAX_VRS];
   BOOLEAN ReadActive[XLR_MAX_VRS];
   BOOLEAN FifoActive;
   BOOLEAN DriveFail;
   UINT    DriveFailNumber;
   BOOLEAN SysError;
   UINT    SysErrorCode;
   BOOLEAN CtlrError;
   BOOLEAN FifoFull;
   BOOLEAN Overflow[XLR_MAX_VRS];
}S_DEVSTATUS, *PS_DEVSTATUS;
```

Purpose

This structure holds various system status flags as returned by the `XLRGetDeviceStatus` function.

Note: The array index value is always 0 for RecordActive, ReadActive, VRActive, and Overflow.

Members

- *SystemReady* – System ready flag, indicates the system firmware and hardware have been initialized successfully.
- *BootmonReady* – Power on boot flag, indicates that the system boot succeeded and the system is ready for initialization (`XLROpen`).
- *Recording* – Indicates that the system is currently in FIFO transfer mode.
- *Playing* – not used.
- *Reserved1, Reserved2, Reserved3 and Reserved4* – not used.
- *RecordActive* – Element 0 indicates that the system is currently transferring into FIFO. Element 1 is not used.
- *ReadActive* – Element 0 indicates that the system is currently reading.
- *FifoActive* – Indicates that the system is currently in FIFO mode.
- *DriveFail* – not used.
- *DriveFailNumber*– not used.

- *SysError* – Indicates that system initialization failed.
- *SysErrorCode* – Holds initialization error code if SysError is TRUE.
- *CtlrError* – Indicates an ATA controller has failed.
- *FifoFull* – Indicates the system is at capacity while in FIFO mode.
- *Overflow* – When in mode SS_MODE_PASSTHRU (see XLRSetMode), Overflow gets set when the external port data has overflowed the available FIFO space.

Structure S_READDESC

```
typedef struct _READDESC{
   PULONG   BufferAddr;
   ULONG    AddrHi;
   ULONG    AddrLo;
   ULONG    XferLength;
}S_READDESC, *PS_READDESC;
```

Purpose

This structure is used to define the parameters for a read or write from the FPDP Digital I/O Board.

Members

- *BufferAddr* – Address of buffer to hold data from the FPDP Digital I/O Board. Must be at least *XferLength* bytes.
- *AddrHi* – High word (32 bit) of starting byte address.
- *AddrLo* – Low word (32 bit) of starting byte address.
- *XferLength* – Number of bytes to transfer from the FPDP Digital I/O Board.

Structure S_XLRSWREV

```
typedef struct _XLRSWREV
{
   char     ApiVersion[XLR_VERSION_LENGTH];
   char     ApiDateCode[XLR_DATECODE_LENGTH];
   char     FirmwareVersion[XLR_VERSION_LENGTH];
   char     FirmDateCode[XLR_DATECODE_LENGTH];
   char     MonitorVersion[XLR_VERSION_LENGTH];
   char     XbarVersion[XLR_VERSION_LENGTH];
   char     AtaVersion[XLR_VERSION_LENGTH];
   char     UAtaVersion[XLR_VERSION_LENGTH];
   char     DriverVersion[XLR_VERSION_LENGTH];
}S_XLRSWREV, *PS_XLRSWREV;
```

Purpose

This structure is used by `XLRGetVersion` to return software/hardware version strings.

Members

- *`ApiVersion`* – Version of the StreamStor API library.
- *`ApiDateCode`* – Build date of the StreamStor API library.
- *`FirmwareVersion`* – FPDP Digital I/O Board firmware version.
- *`FirmDateCode`* – Build date of the firmware.
- *`MonitorVersion`* – Boot monitor firmware version.
- *`XbarVersion`* – Controller logic version.
- *`AtaVersion`* – ATA controller version.
- *`UAtaVersion`* – Ultra ATA controller version.
- *`DriverVersion`* – Driver version.

Chapter 5

PCI Integration

PCI Integration

To allow maximum bandwidth transferring digital data over the PCI bus, the FPDP Digital I/O Board is designed for direct card-to-card data transfers. Since many data acquisition cards already perform DMA operations directly to system memory, the FPDP Digital I/O Board uses this capability for the direct transfer of data. The software development kit provides the necessary control functions for integration of the FPDP Digital I/O Board into user applications.

Initialization and Setup

Initialization requires a call to the `XLROpen` function. This function will lock the FPDP Digital I/O board for exclusive access and initialize the board. The initialization routine includes locating the FPDP Digital I/O Board on the PCI bus, downloading software and initializing required data structures, etc.

PCI Bus Interfacing

Although the PCI bus itself has been designed for card-to-card transactions, most operating systems have no provisions for this functionality. In addition, most operating systems do not have provisions for real-time event management, which is required when transferring data at high bandwidths. For these reasons, there may be a requirement to modify existing device drivers for the PCI card that is to send data to the FPDP Digital I/O Board.

The FPDP Digital I/O Board requests a memory mapped window during computer booting providing a memory space for writing data to be transferred. The default size of this window is 8MB although you should use the `XLRGetBaseRange` to verify this in your application. The StreamStor SDK provides two functions that return the physical and logical addresses of this window.

The address returned by `XLRGetBaseAddr` is the physical address that is assigned to the FPDP Digital I/O Board data window during the boot process. The FPDP Digital I/O Board PCI interface chip will respond to any memory writes on the PCI bus in this address range. Note, however, that the FPDP Digital I/O Board does not utilize the address to determine where to store the data. Any data transferred in the order they are received. This physical address can be used directly for programming DMA hardware on the PCI data source device. Various techniques can be used for programming the DMA hardware but generally you will need to set up a DMA block transfer that continuously recycles back to the original starting address. If the DMA hardware supports chaining (scatter/gather) then a looping transfer can be set up. Consult the documentation for your PCI data acquisition card for more information.

☠ **CAUTION:** ***The physical address returned by XLRGetBaseAddr cannot be used in place of a buffer memory address. Use XLRGetWindowAddr instead.***

The address returned by XLRGetWindowAddr is a logical address created by the operating system to "map" the physical address space of the FPDP Digital I/O Board into the application memory space. This address can sometimes be used with software provided by PCI card vendors in place of the address of a memory buffer. Check with Conduant about your specific environment for more details

Multi-Card Operation

Multiple FPDP Digital I/O Boards can be used in a single system either on the same bus or on "bridged" PCI buses. If multiple FPDP Digital I/O Boards are installed into the same bus there will be contention for ownership of the bus during data transfers and the effective bandwidth will be reduced. If multiple FPDP Digital I/O Boards are installed on opposite sides of a PCI-PCI bridge than there is no loss in bandwidth as long as the data capture card is co-located on the same bus as the FPDP Digital I/O Board it is streaming data to.

Software applications gain exclusive access to an FPDP Digital I/O Board after calling the XLROpen function. Until the application exits or calls XLRClose, no other application may connect to that FPDP Digital I/O Board. A single application can connect to and control multiple FPDP Digital I/O Boards but must manage the unique handles returned from multiple calls to the XLROpen function. The index number passed into XLROpen determines which card is to be controlled by the handle returned. If multiple applications (or multiple instances of the same application) are used to control FPDP Digital I/O Boards, they must each connect to a unique FPDP Digital I/O Board. The XLRDeviceFind function returns the number of FPDP Digital I/O Boards found in the system. The index number cannot be larger than this number. In most cases, the higher value index indicates a card that is on a bus or slot further from the main bus. The PCI bus number and slot number are available from the XLRGetDeviceInfo command. The command can be used to identify the appropriate card in a multi-card system.

Chapter 6

Operation

Operation

FPDP Digital I/O Boards have the capability of real time passing of data streams. This mode of operation is called "pass through." In pass through mode, the board is receiving input data over one channel (the PCI bus, the top FPDP port or the Front FPDP port) and simultaneously outputting that data over a different channel.

In general, to transfer data, your application will follow these steps:

1. Open the FPDP Digital I/O Board (`XLROpen`).
2. Set the mode of the board to pass through (`XLRSetMode`).
3. Clear all channel bindings (`XLRClearChannels`).
4. Bind the input channel (`XLRBindInputChannel`).
5. If using FPDP on the input channel, select that channel and configure it for FPDP (`XLRSelectChannel` and `XLRSetFPDPMode`).
6. Bind the output channel (`XLRBindOutputChannel`).
7. If using FPDP on the output channel, select that channel and configure it for FPDP (`XLRSelectChannel` and `XLRSetFPDPMode`).
8. Start the transfer of data into the FPDP Digital I/O Board FIFO (`XLRRecord`).
9. If transferring data into the FPDP Digital I/O Board FIFO, read data from the FIFO (`XLRGetFIFOLength` and `XLRReadFifo`). See Example 1 in the Channel Description and Selection chapter of this manual.
10. If transferring data out of the FPDP Digital I/O Board FIFO, initiate data writes to the board (`XLRWrite` or `XLRWriteData`). See Example 2 in the "Channel Description and Selection" chapter of this manual.
11. When the desired amount of data has been transferred, stop the transfer of data (`XLRStop`).
12. Close the FPDP Digital I/O Board (`XLRClose`).

Details on selecting channels can be found in the "Channel Description and Selection" of this manual. Details on configuring FPDP options can be found in the "External Port" chapter of this manual.

Data Transfer

After getting the base address of the data window using `XLRGetBaseAddr`, it is used to setup the DMA hardware on the data acquisition card for direct slave writing to the FPDP Digital I/O Board. Because the capacity available on the FPDP Digital I/O Board is much larger than the 32 bit PCI address scheme (4 GB) will allow, the system is designed to ignore PCI addressing and assume any data written within the PCI address range is data to be transferred sequentially. The actual size of the data window can be found with a call to XLRGetBaseRange (default: 8MB). The PCI data source card is required to maintain a destination address within this range. This can easily be accomplished with DMA chaining or other techniques. For example, the data acquisition card can be programmed to start at the base address, write 64kB, than start over again at the base address for the next 64kB, etc.

Transferring Data into the FIFO

To start the transfer of data into the FPDP Digial I/O Board FIFO, the user application must call the XLRRecord function. Once XLR_SUCCESS status has been returned from this function, the FPDP Digital I/O Board will transfer all data written to its data address range into the FIFO. This function should be called BEFORE starting the flow of data to prevent overflow on the data source device. The user application can periodically sample the device status using XLRGetDeviceStatus to check for errors that occurred during transfer. Note that this function call generates PCI traffic and can impact data transfer bandwidth if used excessively.

Many data acquisition cards have operating modes that allow the capture of a specific number of data points. Unfortunately, the software does not usually allow specifying a number larger than a 32-bit integer (4,294,967,295). For this reason it may be necessary to use the data acquisition card in a “pre-trigger” mode where data is captured continuously until the trigger and then a specified number of data points are captured after the trigger. The data acquisition card will then continuously cycle through its “memory buffer” until receiving the trigger. The FPDP Digital I/O Board will continuously transfer all of the data.

Note that he data must be output at the same speed as it is coming in. If not, an overflow condition will be signaled (see Overflow section below) and the data order of the output stream can no longer be guaranteed.

Ending the Transfer

The FPDP Digital I/O Board will continue to transfer data until the XLRStop function has been called.

Also, note that a data acquisition system can stop filling the FIFO by simply ceasing any writes to the FPDP Digital I/O Board data address range. The XLRStop function should be used to flush all data.

Reading Data from the FIFO

XLRReadFIFO retrieves data from FPDP Digital I/O Board’s FIFO. You pass XLRReadFIFO a pointer to a buffer to receive the data and the length of data to read. XLRReadFifo will automatically begin the transfer of data when the FIFO has the requested amount of data in it. If, after five seconds, there is not enough data in the FIFO to fulfill the request, `XLRReadFIFO` will time out.

Checking the FIFO length

`XLRGetFifoLength` tells you how much data is available for reading from the FIFO. `XLRGetFIFOLength` is most useful in a case where the last buffer of data received into the FIFO is less than the requested amount. For example, say you were transferring buffers of one MG and that the last buffer to be transferred did not contain a full MG. In this case, `XLRReadFifo` would time out (since the full MG was never received). If you then retrieve the error code after the `XLRReadFifo` call, you would get the `XLR_ERR_EMPTY` ("No Data") error. This indicates that `XLRReadFifo` was not able to return the requested number of bytes.

If you get the "No Data" error and want to get the data from the partially filled buffer, you would:

1. Call `XLRStop` (to stop the FIFO transfer).

2. Call `XLRGetFifoLength` to get the length of data in the FIFO.

3. If the length returned by `XLRGetFifoLength` is not zero, call `XLRReadFifo` with the length parameter set to the amount returned from `XLRGetFifoLength`.

Ending a FIFO Operation

Stopping data forking or passthru requires the use of **two** calls to `XLRStop`. The first `XLRStop` will shutdown the receiving hardware, but leave the sending operation (over the PCI bus) still running. After the first stop, call `XLRGetFIFOLength` to find out exactly how much data is left in the FIFO to read. Next, call `XLRReadFIFO` (with the amount returned from `XLRGetFIFOLength` – **make sure the buffer is big enough**) to read out the remaining data. Finally, call `XLRStop` to take the StreamStor out of record mode.

Overflows

Data pass through operates in a real time fashion. If data is coming in faster than it is leaving, the FPDP Digital I/O Board's on-board RAM buffer will eventually fill and an overflow condition will arise. Overflow conditions are signaled by the `Overflow` member of the `S_DEVSTATUS` structure. This structure is filled by calls to `XLRGetDeviceStatus`. See the function reference for more information.

☠ **CAUTION:** ***Once an overflow condition arises, the integrity and order of output data can no longer be guaranteed. The only way to "recover" from an overflow situation is to stop and restart the FPDP Digital I/O Board.***

Chapter 7

External Port

External Port

The FPDP Digital I/O Board has connectors and electronics to transfer data into and out of the FPDP Digital I/O Board. Use of these connectors (or " external ports") offer several advantages:

- freedom from interaction with other devices on an arbitrated bus such as PCI;
- the reduction or elimination of bus FIFOs that may otherwise be required to interface with an arbitrated bus;
- full isolation of data path from operating system and computer hardware facilitates predictable and repeatable behavior;
- better or additional control over timing and other parameters;
- higher bus utilization efficiency due to non-arbitrated nature;
- access to interface signals without risk of crashing host computer;
- higher data rates than the most common PCI buses support; and
- the potential for dual-port operation (simultaneous transfers on both PCI bus and external ports).

FPDP

Overview

FPDP is a 32-bit synchronous data bus that allows data to be transferred at high speeds between devices. Simple and low-cost in its implementation, FPDP supports the necessary flow controls to manage transfers between devices of different speeds. Sustained speeds up to 200Mbytes/sec are supported on the FPDP Digital I/O Board FPDP interface.

In reading the following sections on using this feature, it is important to be familiar with the American National Standard for Front Panel Data Port Specifications (ANSI/VITA 17-1998). This manual is intended to clarify FPDP Digital I/O Board's operation as it relates to the standard, not to educate one on the standard itself. For additional information about the standard, other FPDP products and manufacturers, and other technical details regarding FPDP, please visit www.fpdp.com.

The FPDP Digital I/O Board FPDP interface is designed to meet and exceed the basic capabilities of FPDP as defined in the FPDP ANSI standard. The following sections describe:

- any optional FPDP features FPDP Digital I/O Board has implemented;
- any features that FPDP Digital I/O Board has implemented as a superset to the standard;
- any known deviations form the ANSI standard;
- any clarifications that might otherwise be left open to interpretation; and
- the API functions necessary to configure an external port.

Interface Electronics

Interface electronics and termination values on the FPDP Digital I/O Board are those recommended by the ANSI standard, though some signals and terminations can be electronically connected or isolated with crossbar switching devices in order to support electronic reconfiguration.

Data Formats

The FPDP is a multi-drop bus intended to carry either framed or unframed data. The FPDP Digital I/O Board currently supports only the unframed data mode. The SYNC* (Sync Pulse) signal is driven to an inactive state while FPDP Digital I/O Board is a data transmitter on the FPDP bus.

Contact Conduant for more information on using framed data.

PIO Signals

PIO signals are programmable lines for I/O for user-defined functions. These are ancillary signals and are not required for the FPDP function. The FPDP Digital I/O Board currently does not drive or act on received PIO signals. Contact Conduant for more information on using PIO signals.

Interface Functions

To ready the FPDP Digital I/O Board to transfer data using FPDP, the API routine `XLRBindxxxChannel` must be called. The FPDP port's channel number will depend on the board type. (For details on channel numbers, see the `XLRSelectChannel` function in the Function Reference section of this manual.) The bind function is called as follows (*xxx* stands for "Input" or "Output" depending on intended usage):

```
XLRBindxxxChannel ( device, 0 );
```

After The FPDP Digital I/O Board binds and selects a channel, an API call to XLRSetFPDPMode is used to configure the port. This command allows you to set the mode to one of:

- FPDP Transmit Master (FPDP/TM)
- FPDP Transmit (FPDP/T, FPDP Digital I/O Board unique)
- FPDP Receive (FPDP/R)
- FPDP Receive Master (FPDP/RM).
- FPDP Receive Master Clock Master (FPDP/RMCM, FPDP Digital I/O Board unique)

In FPDP/T mode, the FPDP Digital I/O Board drives the FPDP DATA, DVALID* (Data Valid), DIR* (direction), and SYNC* (Sync Pulse) signals but uses the FPDP clock that is driven to the FPDP bus by some other source. In this mode, the FPDP Digital I/O Board does not provide any termination for signals other than DATA[1]. To use this mode properly, the FPDP Digital I/O Board should NOT be positioned at either end of the FPDP bus. Note also that the maximum useable frequency in this mode will decay more rapidly as the cumulative distance from the clock source to the data source to the data destination increases.

In FPDP/RMCM mode, the FPDP Digital I/O Board acts as a Receive Master and drives the FPDP clock signals on the FPDP bus. In addition, the FPDP Digital I/O Board terminates the clock signals (PSTROBE, PSTROBE*, and STROB) as would a traditional FPDP/TM while terminating the remaining signals as would a FPDP/RM. To use this mode, the FPDP Digital I/O Board should be physically positioned at an end of the FPDP bus. Note also that the maximum useable frequency in this mode will decay more rapidly as the cumulative distance from the clock source to the data source to the data destination increases.

When configuring the FPDP Digital I/O Board for data transfer, it may be desirable to prevent a transmitter from sending data until the FPDP Digital I/O Board transfer function is fully enabled. XLRSetFPDPMode can be used to assert the FPDP NRDY* (Not Ready) signal when the FPDP Digital I/O Board is activated as an FPDP receiver. NRDY* will remain asserted until the FPDP Digital I/O Board data transfer is ready to proceed. An example of this is:

```
XLRSetFPDPMode( device, FPDP_RECVMASTER, SS_OPT_FPDPNRASSERT );
```

[1] FPDP Digital I/O Board always provides series termination on the DATA signals as described in Permission 6.4.1 of the ANSI specification.

PSTROBE/PSTROBE* and STROB Signals

When in FPDP/TM or FPDP/RMCM modes, the FPDP Digital I/O Board will drive and terminate both the differential clock pair of PSTROBE, PSTROBE* (± PECL Data Strobe) and the single-ended STROB (Data Strobe) TTL clock. When in any other mode, the user will select which of the two FPDP clock sources the FPDP Digital I/O Board should use from the FPDP bus. The clock can be selected by calling `XLRSetFPDPMode` with the desired clock option. For example, to enable the data strobe clock (TTL):

```
XLRSetFPDPMode( device, FPDP_RECV, SS_OPT_FPDPSTROB );
```

Refer to the FPDP ANSI standard for recommendations and observations about the use of these signals.

Operating Frequency Range

In either FPDP/TM or FPDP/RMCM mode, the FPDP Digital I/O Board can be programmed to synthesize a bus clock in the range from 6 to 50MHz. The FPDP Digital I/O Board can operate from FPDP clocks supplied by other sources at frequencies down to DC. Note, however, that the ANSI specification limits the clock to 20MHz if a receiver is using the STROB (Data Strobe) clock. To program the clock, use the API function `XLRSetPortClock`. By default, the clock frequency is set to 8MHZ. Clock frequencies can be found in the header file `xlrapi.h.`

Chapter 8

Channel Description and Selection

Channel Description and Selection

There are three data paths or channels that can be used to input and output data to /from the FPDP Digital I/O Board. These channels are: the PCI Bus, the FPDP top connector, and the FPDP front connector. A single channel or multiple channels may be selected to receive data. Only one channel at a time can be selected to output data. This section describes the commands that should be used to correctly set up the FPDP Digital I/O Board channels for data transfer.

Channel Description

The FPDP Digital I/O Board currently supports three channels: the PCI Bus, the FPDP top connector, and the FPDP front connector. The PCI Bus is defined as channel 0, the FPDP top connector is defined as channel 30 and the FPDP front connector is defined as channel 31. The default channel is the PCI Bus channel 0. (There are plans to increase the number of PCI channels in the future.)

Selecting an Operating Mode

The FPDP Digital I/O Board operating mode should be set before binding or selecting a channel. The function XLRSetMode is used to set the operating mode by passing the Mode parameter of SS_MODE_PASSTHRU to set pass through mode.

Binding and Selecting Channels

The user application must identify the data input and output channels to be used by the FPDP Digital I/O Board. The process of choosing a channel is called *binding* a channel. Binding a channel is analogous to choosing the data path. The function XLRBindInputChannel is used to bind a channel for input into the FPDP Digital I/O Board and the function XLRBindOutputChannel is used to bind a channel for output from FPDP Digital I/O Board. These functions should be called before data is transferred to or from the FPDP Digital I/O Board is initiated.

XLRClearChannels should be called to clear the default channels prior to calling XLRBindInputChannel and XLRBindOutputChannel, or these functions *may* return an error. The default channel is the PCI Bus channel 0.

The XLRSelectChannel function is also used to select a channel that future functions will act on. For example, XLRSelectChannel needs to be called to select the FPDP channel before a call to XLRSetFPDPMode is made.

Example 1

```
/*
 * This example shows how you can use the FPDP Digital I/O Board buffer
 * as a FIFO to read data from an external port.  In this example,
 * we use channel 30 (the top FPDP port) as the external port.
 * XLRRecord initiates the transfer of data into the
 * FPDP Digital I/O Board buffer in a first in-first out mode.
 * Then XLRReadFifo is used to retrieve data into system memory
 * from this FIFO buffer.
 */
#include <stdio.h>
#include <stdlib.h>
#include "xlrapi.h"

void errorExit(SSHANDLE xlrDevice);

#ifdef WIN32
#define LONGLONGFMT  "%I64u"
#else
#include <unistd.h>  // for sleep
#define Sleep(x)     (sleep((UINT)(x/1000)))
#define LONGLONGFMT  "%llu"
#endif

#define  BUFFS_TO_READ     10
#define  BUFFSIZE          131072
#define  WAIT_LIMIT        5

int main(int argc, char * argv[])
{
   SSHANDLE             xlrDevice;
   DWORDLONG            fifoLength = 0;
   ULONG                lengthToRead = 0;
   ULONG                readBuff[249856];
   XLR_RETURN_CODE      xlrStatus;
   UINT                 iterations=0;
   UINT                 buffCount=0;
   char                 errorMessage[XLR_ERROR_LENGTH];
   char                 prtBuf[256];
   char                 notReadyMsg[256];

   xlrStatus = XLROpen (1, &xlrDevice);
   if (xlrStatus != XLR_SUCCESS) {
      XLRGetErrorMessage(errorMessage, XLRGetLastError());
      printf ("Could not open FPDP Digital I/O Board. Error = %s\n",
            errorMessage);
      exit(1);
   }

   xlrStatus = XLRSetMode(xlrDevice, SS_MODE_PASSTHRU);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not set mode to pass through.\n");
      errorExit(xlrDevice);
   }
```

```
  xlrStatus = XLRClearChannels(xlrDevice);
  if (xlrStatus != XLR_SUCCESS) {
     printf ("Could not clear channels.\n");
     errorExit(xlrDevice);
  }

  xlrStatus = XLRBindInputChannel (xlrDevice, 30);
  if (xlrStatus != XLR_SUCCESS) {
     printf ("Could not bind input channel to top port.\n");
     errorExit(xlrDevice);
  }

  xlrStatus = XLRSelectChannel (xlrDevice, 30);
  if (xlrStatus != XLR_SUCCESS) {
     printf ("Could not select the top port.\n");
     errorExit(xlrDevice);
  }

  xlrStatus = XLRSetFPDPMode(xlrDevice, SS_FPDP_RECVMASTER,
        SS_OPT_FPDPNRASSERT);
  if (xlrStatus != XLR_SUCCESS) {
     printf ("Could not set top port to Receive Master.\n");
     errorExit(xlrDevice);
  }

  xlrStatus = XLRBindOutputChannel (xlrDevice, 0);
  if (xlrStatus != XLR_SUCCESS) {
     printf ("Could not bind output channel to PCI.\n");
     errorExit(xlrDevice);
}

xlrStatus = XLRRecord(xlrDevice, 0, 0);
if (xlrStatus != XLR_SUCCESS) {
   printf ("XLRRecord failed.\n");
   errorExit(xlrDevice);
}

//
// Data is received by the FPDP Digital I/O Board over the top
// FPDP port.
// Read a few buffers worth of data.
//
lengthToRead = (ULONG)BUFFSIZE;
for (buffCount = 0; buffCount < BUFFS_TO_READ; buffCount++) {

     //
     // Get the amount of data that is currently in the fifo.  Loop,
     // checking the fifo until it fills.  Once it fills, we are
     // ready to start reading from it.
     //
     fifoLength = XLRGetFIFOLength(xlrDevice);

     if (fifoLength < lengthToRead) {
        sprintf (notReadyMsg, " Waiting to fill - Fifo Length =  %s\n",
              LONGLONGFMT);
        printf (notReadyMsg, fifoLength);
```

```
         if (iterations > WAIT_LIMIT) {
            printf ("\tTimed out waiting for fifo to fill.\n");
            break;
         }

         //
         // Sleep 3 seconds, which should be enough time for the fifo
         // length to get updated.
         //
         Sleep(3000);
         iterations++;
         continue;
      }

      if (iterations > WAIT_LIMIT) {
         break;
      }

      sprintf (prtBuf, "Bytes in fifo = %s\n", LONGLONGFMT);
      printf (prtBuf, fifoLength);
      if( XLRReadFifo( xlrDevice, readBuff, lengthToRead, 0)
          != XLR_SUCCESS ) {
         printf("\nERROR: Read FIFO failed.\n");
         XLRGetErrorMessage(errorMessage, XLRGetLastError());
         printf ("Exiting because of error:  %s\n", errorMessage);
         XLRClose(xlrDevice);
         exit(1);
      }
   }

   // Stop the transfer of data.
   if( XLRStop(xlrDevice) != XLR_SUCCESS ) {
      printf("XLRStop failed.\n");
      errorExit(xlrDevice);
   }

   //
   // If the last buffer read was only partially full, read it now.
   //
   fifoLength = XLRGetFIFOLength( xlrDevice );
   if (fifoLength == 0) {
      printf ("No partial buffer needs to be read.\n");
   }
   else {
      //
      // Make sure the requested length is an even multiple of 8 bytes.
      //
      if( ( fifoLength % 8 ) != 0 ) {
         fifoLength = (ULONG)( ( fifoLength / 8 ) * 8 );
      }

      lengthToRead = (ULONG)fifoLength;
      //
      // Read the data from the partially filled FIFO.
      //
      if( XLRReadFifo( xlrDevice, readBuff, lengthToRead, 0)
            != XLR_SUCCESS ) {
         printf ("Readfifo of partial buffer failed.\n");
         errorExit(xlrDevice);
      }
   }
   // Take the card out of record mode.
   XLRStop(xlrDevice);

   XLRClose(xlrDevice);
   exit(0);
}

void errorExit(SSHANDLE xlrDevice)
{
   char     errorMessage[XLR_ERROR_LENGTH];

   XLRGetErrorMessage(errorMessage, XLRGetLastError());
   printf ("Exiting because of error:  %s\n", errorMessage);
   XLRClose(xlrDevice);
   exit(1);
}
```

Example 2

```
/*
 *
 * This example shows how you can use the FPDP Digital I/O Board
 * buffer as a FIFO to write data to an external port.  In this
 * example, we use channel 0(the PCI bus) as input and channel 31
 * the front FPDP port) as output.
 *
 * XLRRecord initiates the system and XLRWrite will
 * put data into the fifo for delivery over channel 31.
 *
 */
#include <stdio.h>
#include <stdlib.h>
#include "xlrapi.h"

void errorExit(SSHANDLE xlrDevice);

int main(int argc, char * argv[])
{
   SSHANDLE             xlrDevice;
   XLR_RETURN_CODE      xlrStatus;
   char                 errorMessage[XLR_ERROR_LENGTH];


   xlrStatus = XLROpen (1, &xlrDevice);
   if (xlrStatus != XLR_SUCCESS) {
      XLRGetErrorMessage(errorMessage, XLRGetLastError());
      printf ("Could not open FPDP Digital I/O Board. Error = %s\n",
            errorMessage);
      exit(1);
   }

   xlrStatus = XLRSetMode(xlrDevice, SS_MODE_PASSTHRU);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not set mode to pass through.\n");
      errorExit(xlrDevice);
   }

   xlrStatus = XLRClearChannels(xlrDevice);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not clear channels.\n");
      errorExit(xlrDevice);
   }

   xlrStatus = XLRBindInputChannel (xlrDevice, 0);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not bind input channel to PCI Bus.\n");
      errorExit(xlrDevice);
   }
   //
   // Set up to use FPDP to write data to the external port, which is
   // channel 31 - the front FPDP port - in this example.
   //
   xlrStatus = XLRBindOutputChannel (xlrDevice, 31);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not bind output channel to 31 - the front port.\n");
      errorExit(xlrDevice);
   }

   xlrStatus = XLRSelectChannel (xlrDevice, 31);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not select the top port.\n");
      errorExit(xlrDevice);
   }

   //
   // Make channel 31 the transmit-master.
   //
   xlrStatus = XLRSetFPDPMode(xlrDevice, SS_FPDP_XMITMASTER, 0);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("Could not set top port to Transmit Master.\n");
      errorExit(xlrDevice);
   }

   xlrStatus = XLRRecord(xlrDevice, 0, 0);
   if (xlrStatus != XLR_SUCCESS) {
      printf ("XLRRecord failed.\n");
      errorExit(xlrDevice);
   }

      … Call XLRWrite or XLRWrite data to write
        the FIFO’ed data to the output channel …

   if( XLRStop(xlrDevice) != XLR_SUCCESS ) {
      printf("XLRStop failed.\n");
      errorExit(xlrDevice);
   }

   XLRClose(xlrDevice);
   exit(0);
}

void errorExit(SSHANDLE xlrDevice)
{
   char     errorMessage[XLR_ERROR_LENGTH];

   XLRGetErrorMessage(errorMessage, XLRGetLastError());
   printf ("Exiting because of error:  %s\n", errorMessage);
   XLRClose(xlrDevice);
   exit(1);
}
```

Chapter 9

Technical Support

(303) 485-2721

support@conduant.com

www.conduant.com/support

Technical Support

Conduant wants to be sure that your FPDP Digital I/O Board works correctly and stays working correctly. In the unlikely event, however, that you are unable to get your new system to work properly, or if a working system ceases to function, we will do all that we can to get your system back online.

Solving the problem is largely a matter of data collection and steps that must be taken one at a time. In order for us to better serve you, we ask that you take the time to perform the following steps prior to calling us. This way, you can provide us with the most meaningful information possible that will help us solve the problem.

Is the problem one that obviously requires replacement parts due to physical damage to the system? If yes, then please gather the information described below and report the problem to tech support, by phone or through the Conduant web site.

Have you confirmed that no cabling has been inadvertently disconnected or damaged while working around the equipment?

Is the card properly seated in the PCI slot?

Has the software installation been corrupted? Try re-installing software.

Have you checked the Conduant web site for technical bulletins?

Have you checked the Software Update page in the Conduant web site to be sure that your software is fully up to date? If your software is down level, you may want to update it to determine if this fixes the problem.

Have you recently installed a new Linux kernel or compiler or a new Windows Service Pack?

If the above steps did not resolve the problem, then please call Technical Support or submit a request for support via the Conduant web site. To submit a request for support, go to www.conduant.com, click on “Support” and then on “Submit a Ticket.”

We will do all that we can to resolve the problem as quickly as possible.

Contacting Technical Support

E-mail: support@conduant.com

Phone: (303) 485-2721

Fax: (303) 485-1247

Web: www.conduant.com

Mail: Conduant Corporation
Technical Support
1501 South Sunset Street, Suite C
Longmont, CO 80501

Appendix A – Error Codes

If you are experiencing one of these errors and are unable to determine the cause, please contact Conduant technical support for assistance. Not all error messages will apply to your specific Conduant product.

Number	Error Title	Description
2	XLR_ERR_NODEVICE	FPDP Digital I/O Board was not found in system.
3	XLR_ERR_NOINFO	Undefined error occurred.
4	XLR_ERR_WDOPEN	Cannot open device driver.
5	XLR_ERR_SYSERROR	The controller reported a system error.
6	XLR_ERR_NOXLR	No FPDP Digital I/O Boards located.
7	XLR_ERR_INVALID_CMD	An invalid command was received by the controller.
8	XLR_ERR_HANDLE	Invalid handle.
9	XLR_ERR_DMAREADFAIL	A DMA read failure occurred.
10	XLR_ERR_SYSTATUS	Request is incompatible with current system status.
11	XLR_ERR_NOCMDSTATUS	The command did not complete. Communication with controller timed out.
12	XLR_ERR_DMAINCOMPLETE	The data transfer timed out and did not complete.
13	XLR_ERR_APPSTART	The controller failed to initialize RAM application.
14	XLR_ERR_OUTOFMEMORY	The DLL failed to allocate sufficient memory.
15	XLR_ERR_WIN32FAIL	A Win32 API failure occurred.
16	XLR_ERR_WRITENOTACTIVE	System not ready to receive data.
17	XLR_ERR_WDVERSION	Incorrect driver version detected.
18	XLR_ERR_OPENHANDLE	Device reference by handle already opened.
19	XLR_ERR_INVALIDINDEX	Invalid card index value.
20	XLR_ERR_DEVICELOCK	Could not lock device for exclusive access.
21	XLR_ERR_DETECTCARD	Card configuration invalid.
22	XLR_ERR_BUFLOCK	Could not lock user memory buffer.

23	XLR_ERR_READFAIL	Data read error.
24	XLR_ERR_WRITERAM	Firmware write to device memory failed.
101	XLR_ERR_INVALID_LENGTH	An invalid or unaligned transfer length was requested (must be 64 bit aligned).
102	XLR_ERR_SYSBUSY	System is busy. Use XLRStop to before sending other commands.
103	XLR_ERR_CMDFAIL	The controller has failed to execute the command.
104	XLR_ERR_FILENOTFOUND	A required file was not found.
105	XLR_ERR_LOADKEY	A required registry key was not found.
106	XLR_ERR_DLDCHECKSUM	A required file is corrupted or upload failed.
107	XLR_ERR_DRVFAIL	A disk drive is failing to respond.
108	XLR_ERR_NODRIVER	Device driver not found or device already open.
109	XLR_ERR_FIFO_INACTIVE	Invalid command, FIFO inactive.
110	XLR_ERR_INVALIDVR	An unconfigured or invalid VR was selected.
111	XLR_ERR_NOTENABLED	Optional feature not enabled.
112	XLR_ERR_OUTOFRANGE	Request was not in the recorded data range.
113	XLR_ERR_NOTINFIFO	Command valid only in FIFO mode.
114	XLR_ERR_KERNELMEM	Unable to allocate kernel memory.
115	XLR_ERR_INTENABLE	Unable install device interrupt.
116	XLR_ERR_READCOLLISION	Attempt to start multiple reads from single thread.
117	XLR_ERR_READIDLE	Attempted to check status on non-existent read request.
118	XLR_ERR_FIFODRIVES	Current drive configuration incompatible with FIFO mode.
119	XLR_ERR_FWVERSION	Hardware firmware incompatible with API version.
120	XLR_ERR_OSFAIL	A system call failed.
121	XLR_ERR_THREADCREATE	Process thread creation failed.
122	XLR_ERR_EXPECTEDDISKS_MATCH	The number of expected disks doesn’t equal the actual number of disks.
123	XLR_BOARDTYPE	Unknown board type found.
124	XLR_ERR_FULL	Insufficient disk space.
127	XLR_ERR_INVOPT	Invalid option value.
142	XLR_ERR_INVALID_PORTMODE	Port in wrong mode for this operation.
143	XLR_ERR_NOAPPEND	Attempt to delete non-existent append.
144	XLR_ERR_EMPTY	No data.
145	XLR_ERR_INVALID_BANK	Invalid bank name specified.

146	XLR_ERR_NOTINBANKMODE	Command only valid in bank mode.
148	XLR_ERR_DRIVEMODULE_ NOTREADY	Drive module is not ready.
153	XLR_ERR_CANNOT_RECOVER _DATA	No recovery of data possible.
154	XLR_ERR_NO_RECOVERABLE _DATA	No recoverable data.
155	XLR_ERR_BAD_DISKSET	A disk is missing from a recording or a disk is mounted that was not part of the set when the recording was originally made.
156	XLR_ERR_INVALID_PLAY _LENGTH	Playback length is beyond the end of the recording or is not aligned on an eight-byte boundary.
157	XLR_ERR_INVALID_ WDLICENSE	Invalid driver license.
158	XLR_ERR_WRITE_ PROTECTED	Command invalid on write protected drive modules.
159	XLR_ERR_MAX_CARDS	Maximum number of FPDP Digital I/O Boards exceeded.
160	XLR_ERR_DRVFAIL_BUS0_ MASTER	Master drive on Bus 0 missing or failing.
161	XLR_ERR_DRVFAIL_BUS0_ SLAVE	Slave drive on Bus 0 missing or failing.
162	XLR_ERR_DRVFAIL_BUS1_ MASTER	Master drive on Bus 1 missing or failing.
163	XLR_ERR_DRVFAIL_BUS1_ SLAVE	Slave drive on Bus 1 missing or failing.
164	XLR_ERR_DRVFAIL_BUS2_ MASTER	Master drive on Bus 2 missing or failing.
165	XLR_ERR_DRVFAIL_BUS2_ SLAVE	Slave drive on Bus 2 missing or failing.
166	XLR_ERR_DRVFAIL_BUS3_ MASTER	Master drive on Bus 3 missing or failing.
167	XLR_ERR_DRVFAIL_BUS3_ SLAVE	Slave drive on Bus 3 missing or failing.
168	XLR_ERR_DRVFAIL_BUS4_ MASTER	Master drive on Bus 4 missing or failing.
169	XLR_ERR_DRVFAIL_BUS4_ SLAVE	Slave drive on Bus 4 missing or failing.
170	XLR_ERR_DRVFAIL_BUS5_ MASTER	Master drive on Bus 5 missing or failing.
171	XLR_ERR_DRVFAIL_BUS5_ SLAVE	Slave drive on Bus 5 missing or failing.
172	XLR_ERR_DRVFAIL_BUS6_ MASTER	Master drive on Bus 6 missing or failing.
173	XLR_ERR_DRVFAIL_BUS6_ SLAVE	Slave drive on Bus 6 missing or failing.
174	XLR_ERR_DRVFAIL_BUS7_ MASTER	Master drive on Bus 7 missing or failing.

175	XLR_ERR_DRVFAIL_BUS7_SLAVE	Slave drive on Bus 7 missing or failing.
176	XLR_ERR_NOTIN_RECMODE	Command only valid when in record mode.
177	XLR_ERR_EXT_TO_PCI_OVERFLOW	External port to PCI overflow.
178	XLR_ERR_INVALID_INTERFACE	Command is not available for the currently in use interface (PCI bus, Ethernet, or Serial port).
179	XLR_ERR_INVALID_RETURN_FORMAT	Data returned from command is formatted incorrectly (Ethernet and Serial port interfaces only).
180	XLR_ERR_INVALID_CHANNEL	The channel being selected or bound is invalid.
181	XLR_ERR_INVALID_OP_ON_CHANNEL	Operation is not permitted on this channel.
182	XLR_ERR_USE_SELECT_CHANNEL	SS_OPT_FPDPEXTCONN is no longer valid for selecting the front FPDP port. XLRSelectChannel must be used.
183	XLR_ERR_INVALID_SYSTEM_MODE	Requested mode is invalid.
184	XLR_ERR_TOO_MANY_CHANNELS	Only 1 input or output channel is allowed in this mode.
185	XLR_ERR_NO_INPUT_CHANNELS	Must have at least 1 input channel.
186	XLR_ERR_NO_OUTPUT_CHANNELS	Must have at least 1 output channel.
187	XLR_ERR_NOT_VALID_IN_MULTI	Operation not valid in mutlichannel mode.
188	XLR_ERR_PARTITION_SIZE	Partition size must be multiple of page size.
189	XLR_ERR_INVALID_PARTITION	Invalid partition.
190	XLR_ERR_TOO_MANY_PARTITIONS	Only 256 partitions are permitted.
191	XLR_ERR_NOT_EMPTY	System must be empty for this command.
192	XLR_ERR_UNKNOWN_DIR_VERSION	The directory version found is newer than the current firmware can handle.
193	XLR_ERR_DATA_INTEGRITY	Data integrity check failed.
300	XLR_ERR_PORT_NOT_FOUND	Port is unavailable (Serial/Ethernet interfaces only).
301	XLR_ERR_PORT_ACCESS_DENIED	Port access is denied (Serial/Ethernet interfaces only).
302	XLR_ERR_PORT_TIMEOUT	Port operation has timed out.
303	XLR_ERR_CONNECT_REFUSED	Connection refused by target.

End of Document