



SL100/SL240
API Guide

Document No. F-T-ML-S2AP1###-A-0-A9

FOREWORD

The information in this document has been carefully checked and is believed to be accurate; however, no responsibility is assumed for inaccuracies. Curtiss-Wright Controls, Inc. reserves the right to make changes without notice.

Curtiss-Wright Controls, Inc. makes no warranty of any kind with regard to this printed material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.

©Copyright 2008, Curtiss-Wright Controls, Inc. All Rights Reserved.

FibreXtreme[®] is a registered trademark of Curtiss-Wright Controls, Inc. All Rights Reserved.

VxWorks[®] is a registered trademark of Wind River Systems.

Tornado[®] is a registered trademark of Wind River Systems.

Windows[®] is a registered trademark of the Microsoft Corporation.

Sun, Sun Microsystems, Sun WorkShop Compilers C, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc.

Any reference made within this document to equipment from other vendors does not constitute an endorsement of their product(s).

Revised: July 9, 2008

Curtiss-Wright Controls Embedded Computing
Data Communications Center
2600 Paramount Place Suite 200
Fairborn, OH 45324 USA
(800) 252-5601(U.S. only)
(937) 252-5601

TABLE OF CONTENTS

1. INTRODUCTION.....	1-1
1.1 How to Use This Manual.....	1-1
1.1.1 Purpose	1-1
1.1.2 Scope	1-1
1.1.3 Style Conventions.....	1-1
1.2 Related Information.....	1-2
1.3 Quality Assurance	1-2
1.4 Technical Support.....	1-3
1.5 Ordering Process	1-3
2. API OVERVIEW	2-1
2.1 SL240 API Introduction	2-1
2.1.1 Point-to-Point Topology	2-1
2.1.2 Ring Topology.....	2-3
2.1.3 SL240 Software Layers	2-5
2.1.4 SL240 Device Driver.....	2-5
2.1.5 SL240 API Library	2-5
2.1.6 Applications.....	2-5
3. INSTALLATION.....	3-1
3.1 Overview	3-1
3.2 Application Directories	3-1
3.3 Tornado 2.2 Applications on a Windows Host.....	3-1
3.3.1 Load the SL240 Applications	3-1
3.3.2 Build SL240 Applications	3-2
3.4 UNIX Applications.....	3-3
3.4.1 Load SL240 Applications	3-3
3.4.2 Build SL240 Applications	3-3
3.5 Windows Applications	3-4
3.5.1 Load the SL240 Applications	3-4
3.5.2 Build the SL240 Applications.....	3-4
3.5.3 Set Up the Visual C++ Environmental Variables	3-4
4. API DESCRIPTION	4-1
4.1 Introduction	4-1
4.2 Constants	4-1
4.2.1 Function Return Values	4-1
4.2.2 Send and Receive Flags	4-1
4.2.3 FPD P Constants	4-2
4.3 Primitive Types	4-2
4.4 Structures.....	4-3
4.4.1 fxsl_configstruct	4-3
4.4.2 fxsl_statusstruct	4-5
4.5 Time-out Operation	4-5
4.6 API Functions.....	4-6
4.6.1 fxsl_open	4-7
4.6.2 fxsl_close.....	4-8
4.6.3 fxsl_recv	4-9
4.6.4 fxsl_send.....	4-11
4.6.5 fxsl_status	4-12
4.6.6 fxsl_get_config	4-13
4.6.7 fxsl_set_config.....	4-14
4.6.8 fxsl_fpdp_get.....	4-15
4.6.9 fxsl_fpdp_put.....	4-16
4.6.10 fxsl_read_CR.....	4-17
4.6.11 fxsl_write_CR.....	4-18
4.7 Return Codes	4-19

5. API EXAMPLES	5-1
5.1 Installing the SL240 Device Driver	5-1
5.2 Opening and Closing a Handle to the Driver	5-1
5.3 Configuring the Driver	5-2
5.4 Sending Data	5-3
5.5 Receiving Data	5-4
5.6 Using Status Information	5-6
5.7 Using FPDP Lines	5-7
5.8 Using SL240 Run-Time Registers	5-8
5.9 API Procedure Flow	5-9
5.10 Additional Topics	5-10
5.10.1 Using Multiple Threads	5-10
5.10.2 Synchronization using FPDP Lines	5-10
6. UTILITY APPLICATIONS	6-1
6.1 Application Overview	6-1
6.1.1 Running SL240 Applications Under Tornado 2.2 and VxWorks	6-1
6.2 Monitor Application	6-2
6.2.1 Link Error Handling Mode	6-4
6.3 Throughput Application	6-5
6.3.1 SLTP	6-5
6.3.2 SL240 Versatile Exerciser (slvex)	6-8
6.3.3 SL240 Register (slreg)	6-11
6.4 Firmware Programming Application	6-14

APPENDICES

GLOSSARY	GLOSSARY-1
INDEX	INDEX-1

FIGURES

Figure 2-1 Point-to-Point Configuration	2-1
Figure 2-2 Hardware Configuration for Point-to-Point Topology	2-2
Figure 2-3 Ring Configuration	2-3
Figure 2-4 Software-Selectable SL240 Hardware Configurations for Multiple-Node Ring Topology	2-4
Figure 2-5 SL240 Software Layers	2-5
Figure 3-1 Directory Structure for SL240 Applications CD-ROM	3-1
Figure 3-2 Tornado Project Options	3-2
Figure 5-1 API Procedure Call Sequence	5-9
Figure 6-1 Tornado 2.2 Window	6-1
Figure 6-2 sltp Online Help Output	6-6
Figure 6-3 SL240 Throughput Data	6-7
Figure 6-4 slvex Online Help output	6-9
Figure 6-5 slvex Runtime Output	6-9
Figure 6-6 slvex 'print' Option Output	6-10
Figure 6-7 slreg Online Help Display	6-12
Figure 6-8 Firmware Programming Example	6-14
Figure 6-9 Online help Display	6-15

TABLES

Table 4-1 API Functions	4-6
Table 4-2 Return Codes	4-19
Table 6-1 Link Error Modes	6-4

1. INTRODUCTION

1.1 How to Use This Manual

1.1.1 Purpose

This manual describes the operation of the FibreXtreme SL100/SL240 driver and illustrates how to interact with the driver via the Application Program Interface (API).



NOTE: Both the FibreXtreme SL100 and SL240 hardware will be referred to throughout this manual as SL240. The software that supports both the SL100 and SL240 hardware will also be referred to as SL240, including the driver and API. Anything that is exclusive to the SL100 or the SL240 will be described as such.

1.1.2 Scope

This manual contains the following information pertinent to all platforms.

- Introduction to the SL240 API.
- Description of constants provided by the SL240 API.
- Description of data structures provided by the SL240 API.
- Description of functions provided by the SL240 API.
- Sample code using the SL240 API.
- Description of the utility applications included.

Operating-system-specific information is included in the various *FibreXtreme SL240 Software Installation Manuals*.

1.1.3 Style Conventions

- Called functions are italicized. For example, *OpenConnect()*.
- Data types are italicized. For example, *int*.
- Function parameters are bolded. For example, **Action**.
- Path names are italicized. For example, *utility/sw/cfg*.
- File names are bolded. For example, **config.c**.
- Path file names are italicized and bolded. For example, ***utility/sw/cfg/config.c***.
- Hexadecimal values are written with a “0x” prefix. For example, 0x7e.
- For signals on hardware products, an ‘Active Low’ is represented by prefixing the signal name with a slash (/). For example, */SYNC*.
- Code and monitor screen displays of input and output are boxed and indented on a separate line. Text that represents user input is bolded. Text that the computer displays on the screen is not bolded. For example:

```
ls
file1          file2          file3
```

- Large samples of code are Courier font, at least one size less than context, and are usually on a separate page or in an appendix.

1.2 Related Information

- *FibreXtreme SL100/SL240 Hardware Reference for PCI, PMC, and CPCI Cards*, (Doc. Nr. F-T-MR-S2PCIPMC), Curtiss-Wright Controls, Inc.
- *FibreXtreme SL100/SL240 Software Installation Manual for Linux*, (Doc. Nr. F-T-MI-LIXXDS21), Curtiss-Wright Controls, Inc.
- *FibreXtreme SL100/SL240 Software Installation Manual for Windows NT 4.0*, (Doc. Nr. F-T-MI-NTXXPS21), Curtiss-Wright Controls, Inc.
- *FibreXtreme SL100/SL240 Software Installation Manual for Solaris 7-9*, (Doc. Nr. F-T-MI-SLSUPS21), Curtiss-Wright Controls, Inc.
- *FibreXtreme SL100/SL240 Software Installation Manual for Tornado 2.2 and VxWorks 5.3.1 and 5.5*, (Doc. Nr. F-T-MI-VWXXGS21), Curtiss-Wright Controls, Inc.
- *FibreXtreme SL100/SL240 Software Installation Manual for Windows 2000 Using PCI, PMC, and CPCI Cards*, (Doc. Nr. F-T-MI-WMXXGS21), Curtiss-Wright Controls, Inc.
- Curtiss-Wright Controls, Inc. – web site : www.cwembedded.com

1.3 Quality Assurance

Curtiss-Wright Controls' policy is to provide our customers with the highest quality products and services. In addition to the physical product, the company provides documentation, sales and marketing support, hardware and software technical support, and timely product delivery. Our quality commitment begins with product concept, and continues after receipt of the purchased product.

Curtiss-Wright Controls' Quality System conforms to the ISO 9001 international standard for quality systems. ISO 9001 is the model for quality assurance in design, development, production, installation and servicing. The ISO 9001 standard addresses all 20 clauses of the ISO quality system, and is the most comprehensive of the conformance standards.

Our Quality System addresses the following basic objectives:

- Achieve, maintain and continually improve the quality of our products through established design, test, and production procedures.
- Improve the quality of our operations to meet the needs of our customers, suppliers, and other stakeholders.
- Provide our employees with the tools and overall work environment to fulfill, maintain, and improve product and service quality.
- Ensure our customer and other stakeholders that only the highest quality product or service will be delivered.

The British Standards Institution (BSI), the world's largest and most respected standardization authority, assessed Curtiss-Wright Controls' BSI's Quality Assurance division certified we meet or exceed all applicable international standards, and issued Certificate of Registration, number FM 31468, on May 16, 1995. The scope of Curtiss-Wright Controls' registration is: "Design, manufacture and service of high technology hardware and software computer communications products." The registration is maintained under BSI QA's bi-annual quality audit program.

Customer feedback is integral to our quality and reliability program. We encourage customers to contact us with questions, suggestions, or comments regarding any of our products or services. We guarantee professional and quick responses to your questions, comments, or problems.

1.4 Technical Support

Technical documentation is provided with all of our products. This documentation describes the technology, its performance characteristics, and includes some typical applications. It also includes comprehensive support information, designed to answer any technical questions that might arise concerning the use of this product. We also publish and distribute technical briefs and application notes that cover a wide assortment of topics. Although we try to tailor the applications to real scenarios, not all possible circumstances are covered.

Although we have attempted to make this document comprehensive, you may have specific problems or issues this document does not satisfactorily cover. Our goal is to offer a combination of products and services that provide complete, easy-to-use solutions for your application.

If you have any technical or non-technical questions or comments, contact us. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: **(937) 252-5601** or **(800) 252-5601**
- E-mail: **DTN_support@curtisswright.com**
- Fax: **(937) 252-1465**
- World Wide Web address: www.cwembedded.com

1.5 Ordering Process

To learn more about Curtiss-Wright Controls' products or to place an order, please use the following contact information. Hours of operation are from 8:00 a.m. to 5:00 p.m. Eastern Standard/Daylight Time.

- Phone: **(937) 252-5601** or **(800) 252-5601**
- E-mail: **DTN_info@curtisswright.com**
- World Wide Web address: www.cwembedded.com

This page intentionally left blank.

2. API OVERVIEW

2.1 SL240 API Introduction

The SL240 API is intended to help users develop their application software and utilize the SL240 hardware. The SL240 API is a set of calls that allows access to the features and functions of the SL240 cards. The SL240 API calls provide an easy-to-use interface that will abstract the complex details of device drivers, DMA transactions, etc. Section 4.6 describes each of the API functions in detail. The SL240 can be used in two different topology configurations: point-to-point and ring.

2.1.1 Point-to-Point Topology

The data flow in a point-to-point configuration would be as follows:

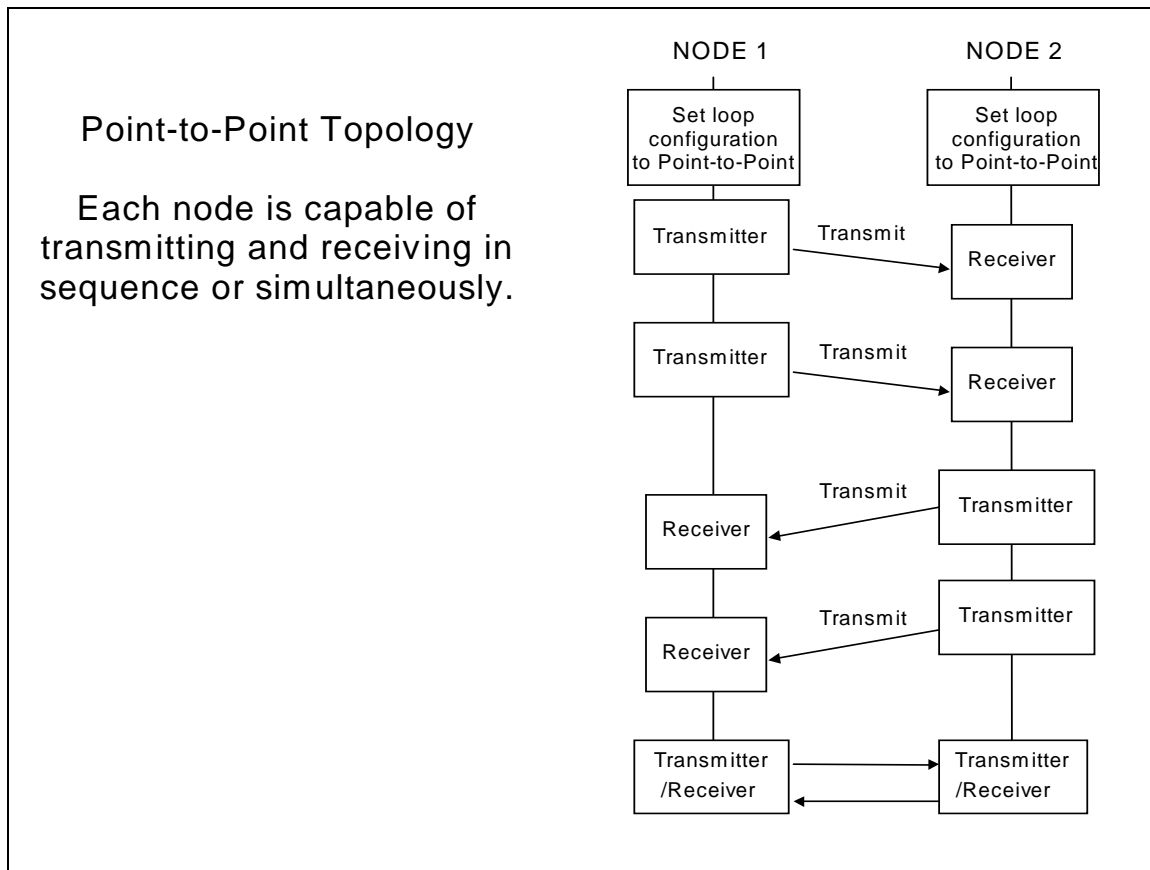


Figure 2-1 Point-to-Point Configuration

The software can be used to control the functionality of the SL240. To make the node function in a point-to-point configuration, the software would put each of the two boards into the Transmitter-Receiver configuration as shown below.

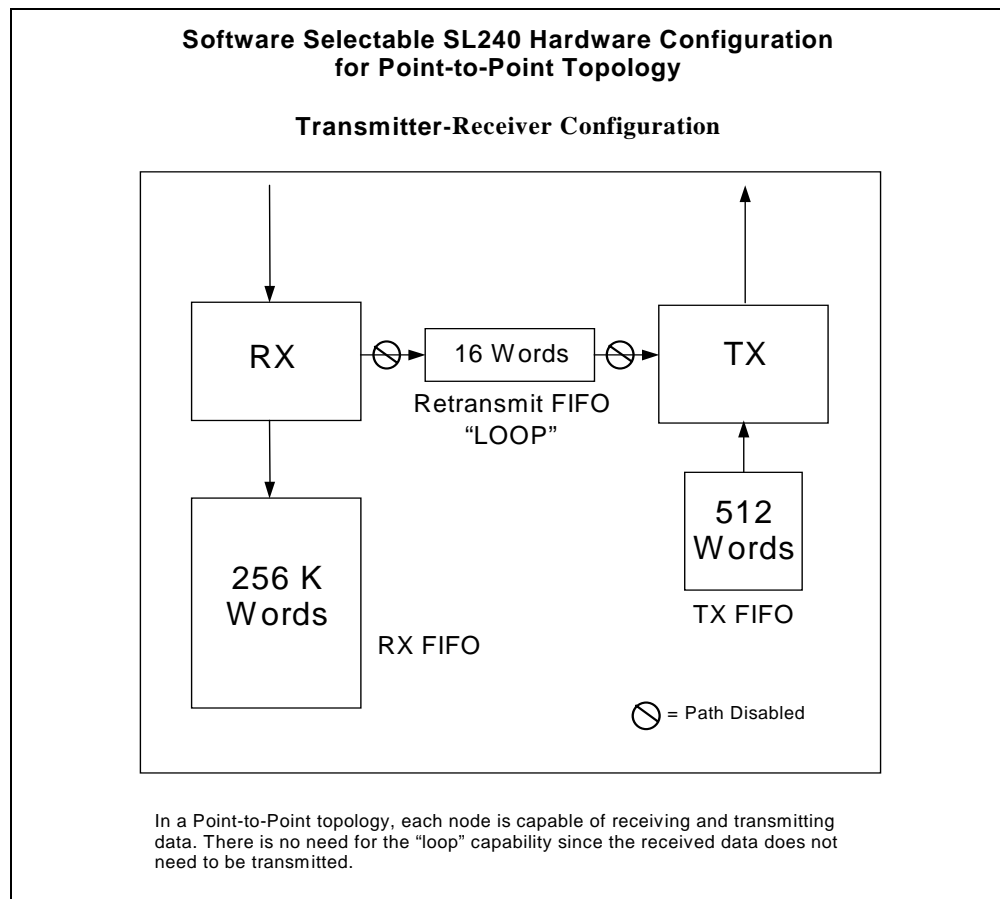


Figure 2-2 Hardware Configuration for Point-to-Point Topology

2.1.2 Ring Topology

A ring configuration consists of three or more nodes. One node becomes a transmitter while the remaining nodes act as re-transmitters, gathering the data and then passing it on to the next node. When the data reaches the transmitting node, it is removed.

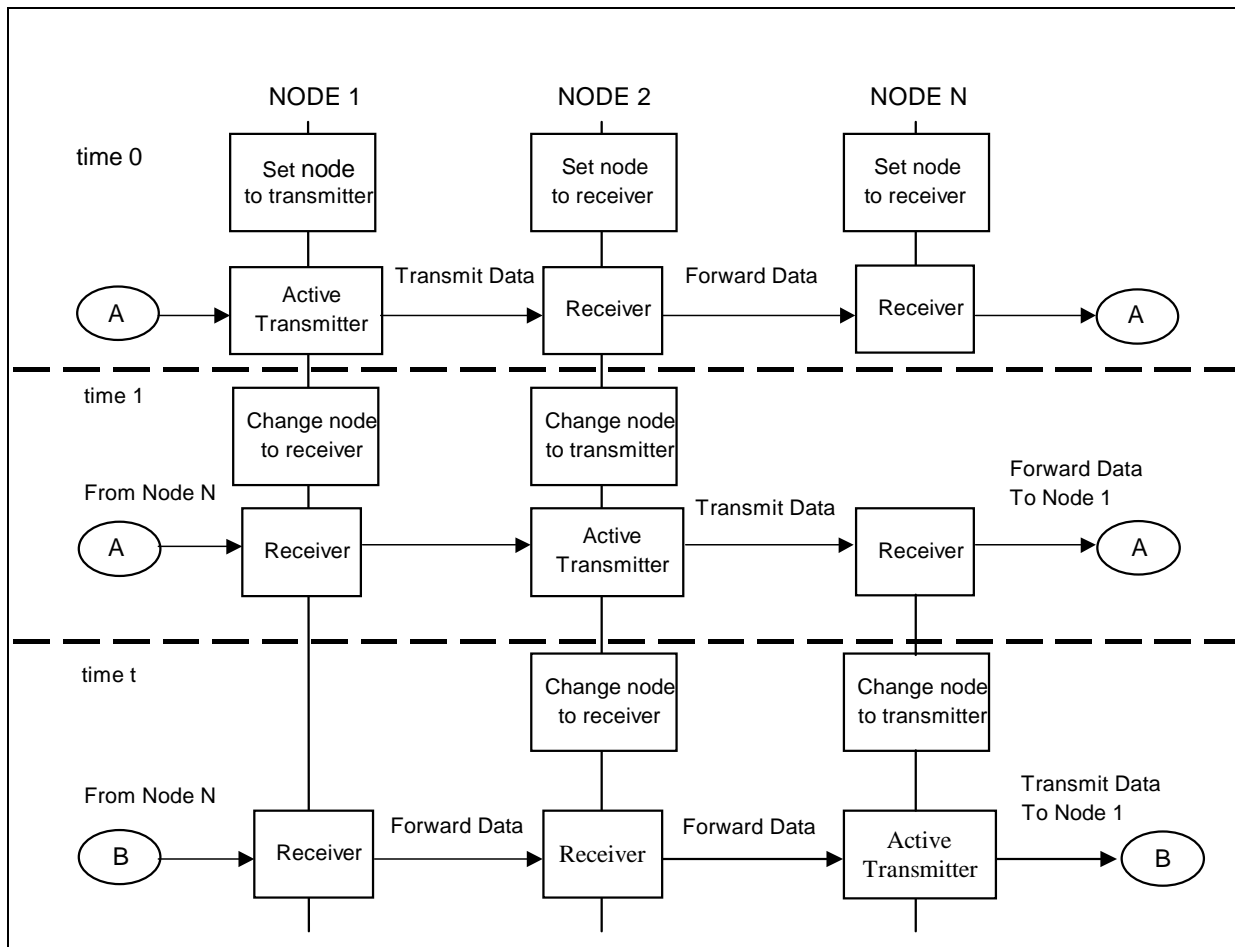


Figure 2-3 Ring Configuration

Only one node can be actively transmitting at a time. The other nodes receive and re-transmit the data. Software on each host computer controls whether the node is a transmitter or receiver. The criteria for which node is a transmitter and which are receivers can be based on a scheduler, the data being received, or some other user-defined method. The software can control which node is the designated transmitter and which nodes are to be the re-transmitters on a host-by-host basis.

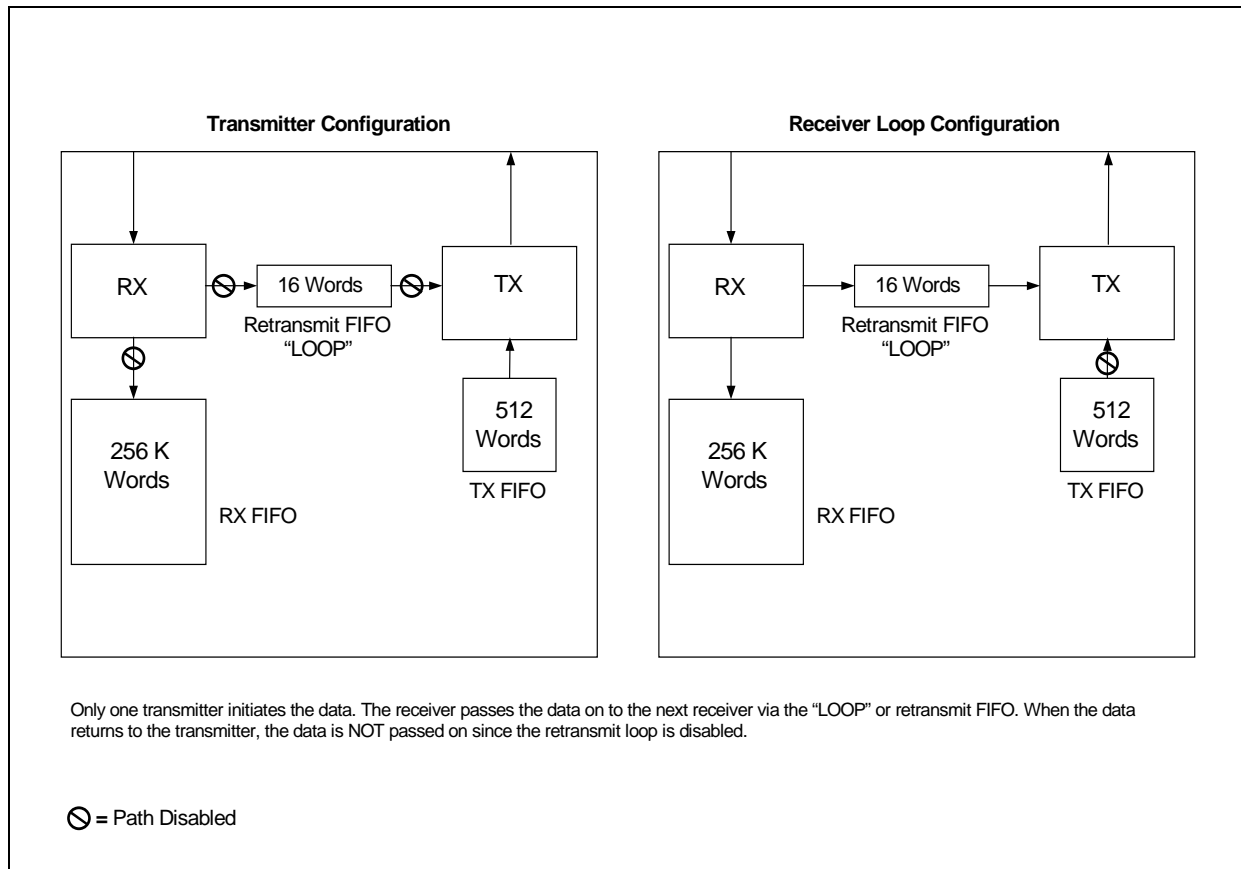


Figure 2-4 Software-Selectable SL240 Hardware Configurations for Multiple-Node Ring Topology

2.1.3 SL240 Software Layers

The SL240 software is composed of “layers.” The software layers interaction is shown in Figure 2-5.

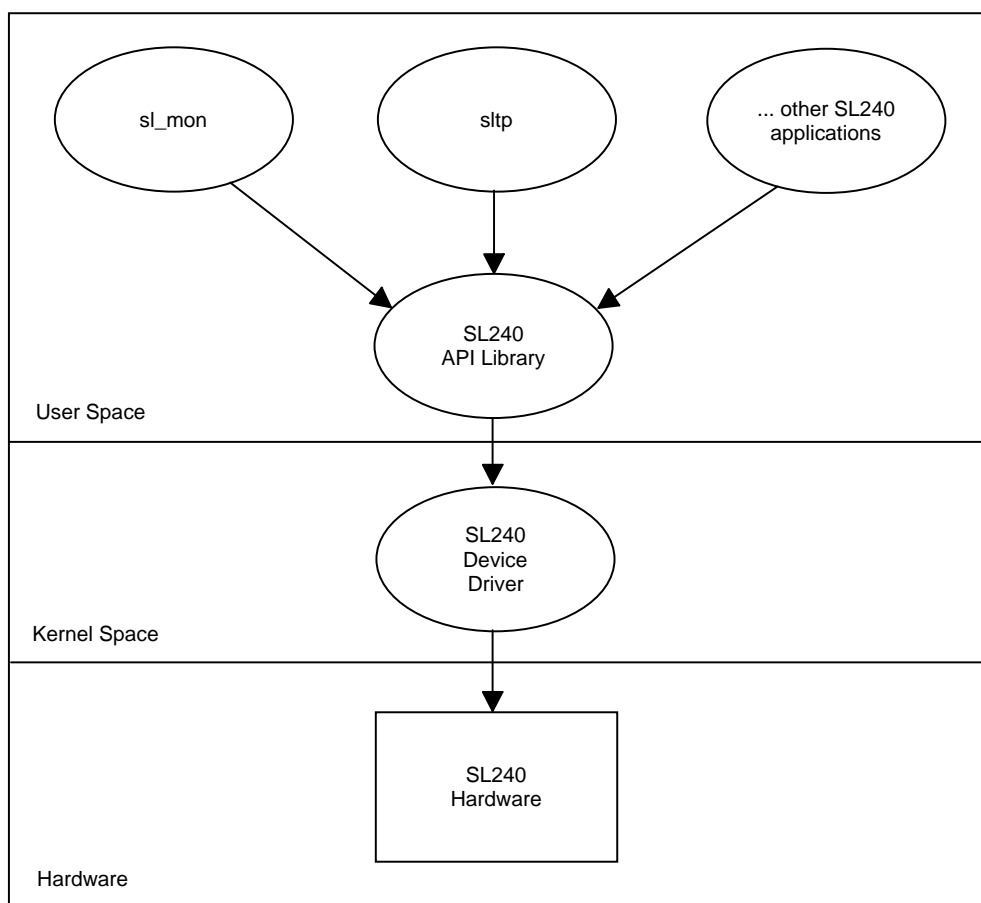


Figure 2-5 SL240 Software Layers

2.1.4 SL240 Device Driver

The SL240 device driver is the lowest software layer. It receives operation requests from the API library and performs the operations using the SL240 hardware.

2.1.5 SL240 API Library

The SL240 API library fits between the applications and the driver. The SL240 API allows applications to access the driver using a set of defined functions. These functions are described in detail later in this manual.

2.1.6 Applications

Applications are typical user-level processes or programs. Applications can communicate with the SL240 hardware through the SL240 API library. The details of the SL240 device driver and hardware are hidden from the applications. This allows applications to operate the same regardless of the operating system used and hardware version. For information about the SL240 applications see Chapter 5.

This page intentionally left blank.

3. INSTALLATION

3.1 Overview

SL240 applications are distributed on a single CD-ROM (part number FSD6#240APP1#0A1). To install the SL240 API and Applications:

- Install the SL240 hardware. See SL100/SL240 Hardware Reference Manual for PCI and PMC cards (on manuals CD).
- Install the SL240 driver according to the driver installation manual.

3.2 Application Directories

The SL240 applications CD-ROM contains the directories shown in Figure 3-1.

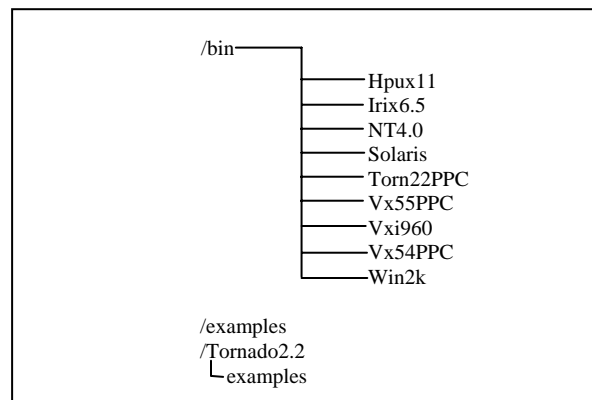


Figure 3-1 Directory Structure for SL240 Applications CD-ROM

The */examples* directory contains the source for the applications.

3.3 Tornado 2.2 Applications on a Windows Host

3.3.1 Load the SL240 Applications

To load the pre-compiled Tornado 2.2 application binaries:

- Insert the SL240 Application CD-ROM in the drive.
- Using Windows Explorer, browse the Tornado 2.2 folder containing the target binaries

Example:

D:\bin\tonr22PPC\dy4182

- From within the folder containing the target binaries, using the Windows explorer menu select Edit->Select All to highlight the files in the folder. Next, right click and select Copy.
- Browse to the bin folder on the host (default is c:\tornado2.2\target\proj\sl240\bin), right click on the folder and select Paste.

3.3.2 Build SL240 Applications

To build the SL240 applications from within the Tornado 2.2 IDE:

- Insert the SL240 Application CD-ROM in the drive.
- Using Windows Explorer, browse to the folder containing the Tornado 2.2 project files.

Example:

D:\Tornado2.2

- From the Windows Explorer menu select Edit->Select All, right click the highlighted files and folders and select Copy.
- Browse the SL240 apps directory on the host (default is: c:\tornado2.2\target\proj\sl240\app). Right click inside the folder and select Paste from the menu.
- From within the apps folder, use Windows Explorer to select the SL240 Tornado 2.2 Application workspace, sl240apps.wsp. Next, right click on the file and select Cut.
- Browse to the Tornado 2.2 project directory (C:\tornado2.2\target\proj) and select Paste.
- Load the Tornado 2.2 IDE and select File->Open workspace, and then choose Browse. Locate the file sl240apps.wsp and then click OK.
- From within the Tornado 2.2 IDE right click on the application and select either Build or Rebuild All from the project menu. Figure 3-2 shows the Tornado 2.2 options menu.

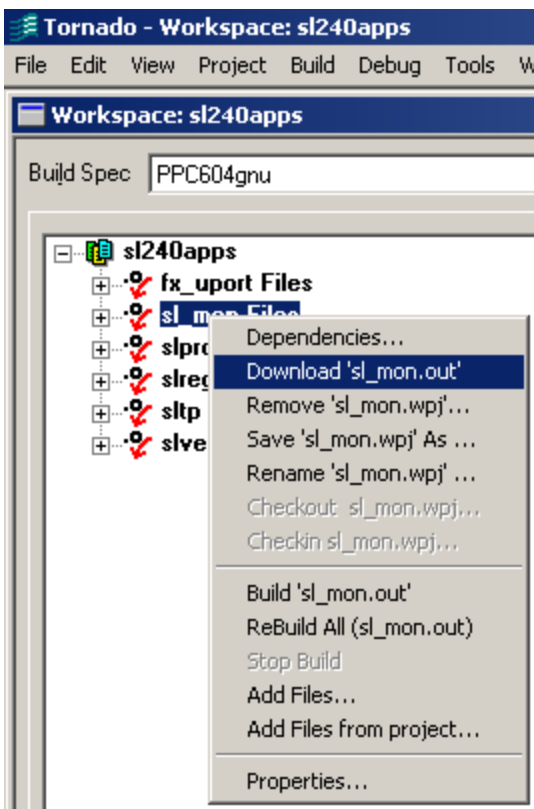


Figure 3-2 Tornado Project Options

3.4 UNIX Applications

3.4.1 Load SL240 Applications

To load the SL240 application binaries:

- Insert the CD-ROM in the drive.
- Log to the host as root (su).
- Mount the CD-ROM device if it isn't already mounted. For example:

```
mount /dev/cdrom
```

Change to the *root* directory on the application CD-ROM:

```
cd /cdrom/
```

Change to the platform-specific directory for the host systems. For example:

```
cd /bin/Solaris
```

Copy the contents from the platform-specific directory to the SL240 driver directory:

```
cp * sl240 driver directory/bin
```

3.4.2 Build SL240 Applications

Load the SL240 application source code:

- Insert the CD-ROM in the drive
- Log on to the host as root (su)
- Mount the CD-ROM if it isn't already mounted.

For example:

```
mount /dev/cdrom
```

Change to the *apps* subdirectory in the SL240 driver directory:

```
cd /sl240/apps
```

Create an *examples* subdirectory:

```
mkdir examples
```

Change to the *examples* subdirectory on the application CD-ROM:

```
cd /cdrom/examples
```

Copy the SL240 application source to the *sl240/apps/examples* subdirectory:

```
cp *.* /sl240/apps/examples
```

Change to the */apps/examples* subdirectory in the SL240 driver directory:

```
cd /sl240/apps/examples
```

Use the platform-specific makefile to build the SL240 applications:

Example:

```
make -f apps.linux.mak
```



NOTE: Chapter 6, UTILITY APPLICATIONS, contains samples to assist in verifying the functionality of the card, application development, and to show how the SL240 API can be used.

3.5 Windows Applications

3.5.1 Load the SL240 Applications

To load the SL240 applications:

- Insert the CD-ROM in the drive.
- Using Windows Explorer, browse to the top folder on the CD-ROM.
- Browse to the appropriate `\bin` directory.

Example:

```
D:\bin\WIN2K
```

- From the Windows Explorer menu bar select Edit→Select all.
- With the contents of the folder highlighted, right click and select 'Copy.'
- Browse to the SL240 driver directory (default is `c:\program files\sl240`).
- Change to the `\bin` folder by left clicking on it.
- Now right click on the `\bin` folder and select 'Paste' to place the application binaries in the `\bin` subfolder of the SL240 driver folder (default is `c:\Program Files\SL240\Bin`).

3.5.2 Build the SL240 Applications

If it is necessary to modify or rebuild the SL240 applications do the following:

- Using Windows Explorer, browse to the top folder of the CD-ROM, right click on the `\examples` folder and select copy.
- Browse to the SL240 driver folder and left click on the `\apps` folder.
- Right click inside the folder and select paste. This will copy the entire `examples` folder from the SL240 application CD-ROM to your SL240 driver folder.

3.5.3 Set Up the Visual C++ Environmental Variables

Open a command prompt in Windows 2000 by selecting:
start→programs→accessories→command prompt

Open a command prompt in Windows NT 4.0 by selecting:
Select Start→ Programs→command prompt

Issue the following commands:

```
cd \program files\Microsoft Visual Studio\vc98\bin
```

```
vcvars32
```

Change to the SL240 examples folder:

```
cd \program files\sl240\apps\examples
```

Execute the platform-specific makefile:

```
nmake -f apps.nt.mak
```

4. API DESCRIPTION

4.1 Introduction

This chapter describes the constants, data types, and functions provided by the SL240 API. Chapter 5 provides code examples for each function call discussed in this chapter.

4.2 Constants

The SL240 API defines a number of constants to assist in interfacing to the SL240 device driver. This section describes these constants.



NOTE: Using SYNC-with-DVALID transfers with the SL240 requires a firmware revision later than 1C.13. Contact Curtiss-Wright Controls, Inc. Embedded Computing Data Communication Center Customer Support for availability.

4.2.1 Function Return Values

The SL240 API is designed so that all functions return values from a predefined set. The possible error values are fully documented in the header file **fxsl.h** (located in the *inc* directory) and in section 3.6.

4.2.2 Send and Receive Flags

The send and receive entry points currently use a flag parameter. The flag is documented in the header file **fxsl.h** (located in the *inc* directory). This flag parameter is a bit mask with these bits:

FXSL_BUFFER_IS_PHYSADDR (Linux and VxWorks only)

Passes in a pointer to the memory of a device mapped into the PCI bus allowing transfers directly from one PCI device to another.

FXSL_SWAP_BYTES..... Specifies byte-swapping of the data in conjunction with the **swapBytes** driver configuration parameter (See subsection 4.4.1 for the definition of **swapBytes** and section 6.2 for user access to it via the monitor application **sl_mon**). Byte swapping occurs if either the **FXSL_SWAP_BYTES** bit is '1' and the **swapBytes** configuration parameter is '0,' or the **FXSL_SWAP_BYTES** bit is '0' while the **swapBytes** parameter is '1.' Otherwise, bytes are not swapped in the data stream.

FXSL_USE_SYNC..... Specifies the use of SYNC for the requested transaction. If the **FXSL_USE_SYNC** bit is '1' and the operation is a transmission, then a SYNC word is output at the end of the transmission; if it is '0,' then no SYNC word is transmitted. If the **FXSL_USE_SYNC** bit is '1' and the

operation is a receive operation, then the reception is ended if a SYNC word is received before the byte count is satisfied and the **FXSL_USE_SYNC** bit is returned as '1.'

FXSL_USE_SYNC_DVALID. Specifies the use of **SYNC_DVALID** for the requested transaction. If the **FXSL_USE_SYNC_DVALID** bit is '1' and the operation is a transmission, then a **SYNC_DVALID** word is output at the end of the Transmission; if it is '0', then no **SYNC_DVALID** word is transmitted. If the **FXSL_USE_SYNC_DVALID** bit is '1' and the operation is a receive operation, then the reception is ended if a **SYNC_DVALID** word is received before the byte count is satisfied and the **FXSL_USE_SYNC_DVALID** bit is returned as '1'.

4.2.3 FPDP Constants

The front panel data port (FPDP) signals can be read and written using the FXSL API. The following constants are available to aid in this process. These should be used when calling the functions *fxsl_fdp_get* (section 4.6.8) and *fxsl_fdp_put* (section 4.6.9). An example of this call is in section 5.7. For more information about the operation of these signals refer to the *FibreXtreme Hardware Reference Manual for VME and CMC FPDP Cards*.

FXSL_FPDP_PIO1_IN Bitmask for the bit denoting FPDP input line PIO1.

FXSL_FPDP_PIO2_IN Bitmask for the bit denoting FPDP input line PIO2.

FXSL_FPDP_DIR_IN Bitmask for the bit denoting FPDP input line DIR.

FXSL_FPDP_NRDY_IN Bitmask for the bit denoting FPDP input line NRDY.

FXSL_FPDP_MASK_IN..... Bitmask that includes all FPDP input lines.

FXSL_FPDP_PIO1_OUT Bitmask for the bit denoting FPDP output line PIO1.

FXSL_FPDP_PIO2_OUT Bitmask for the bit denoting FPDP output line PIO2.

FXSL_FPDP_DIR_OUT Bitmask for the bit denoting FPDP output line DIR.

FXSL_FPDP_NRDY_OUT.... Bitmask for the bit denoting FPDP output line NRDY.

FXSL_FPDP_MASK_OUT ... Bitmask that includes all FPDP output lines.

4.3 Primitive Types

The SL240 API requires that several data types be used to perform some operations.

To avoid problems across platforms, the API defines a number of data types to hold numbers and characters. These data types are shown below and can be found in the files **fxsl.h** and **fxslapi.h** (both files are located in the *inc* directory).

fx_long..... The type *fx_long* is a signed 32-bit integer.

fx_ulong The type *fx_ulong* is an unsigned 32-bit integer. This type is used as the return value for all SL240 API function calls.

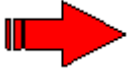
fx_short The type *fx_short* is a signed 16-bit integer.

fx_ushort The type *fx_ushort* is an unsigned 16-bit integer.

fx_byte..... The type *fx_byte* is a signed 8-bit integer.

fx_ubyte..... The type *fx_ubyte* is an unsigned 8-bit integer.

HANDLE The type *HANDLE* is an abstraction of the file descriptor in UNIX, the *HANDLE* in Windows, and other similar descriptors used by other operating systems. The SL240 API uses a *HANDLE* to determine which device the request should be executed on. Consult the example applications for more information on using the type *HANDLE*.



NOTE: VxWorks 6.x also defines a *HANDLE* data type. This definition conflicts with the SL240 *HANDLE* data type definition. To work around this conflict, the SL240 API uses a *HANDLE_SL240* data type instead of a *HANDLE* under VxWorks 6.x. Your VxWorks 6.x application should also use the *HANDLE_SL240* data type.

Pname **P** in front of a type name denotes a pointer to that type in function prototypes.

4.4 Structures

Structures are used to retrieve the configuration and status of the driver. These structures are described here.

4.4.1 fxsl_configstruct

The type *fxsl_configstruct* is used to configure the driver's parameters. See section 5.3 for an example. The user accessible structure members are:

allow_qing_on_lerror No meaning in SL100/SL240 revision B drivers.

convert_dsync..... Set to '1' to configure the card for SYNC with DVALID transfers.
Set to '0' to ignore SYNC with DVALID.

halt_on_link_errors Set to '1' to halt the receive DMA on a link error.
Set to '0' to ignore link errors on the receive channel.

loopConfig..... Set loop configuration mode.
0x0 Point-to-Point
0x1 Initiator
0x2 Receive Only Terminator
0x3 Terminator
0x4 EWRAP for verify
0xA Chain Target
0xB Chain No Tx No Rx

max_receive_timeout Reserved (see section 4.5 Time-out Operation for details).

max_send_timeout Maximum time for send or receive to wait on a semaphore (in 1/100ths of a second)

swapBytes Set to '1' to swap data bytes.
Set to '0' to not swap data bytes.

use_flow_control Set to '1' to use flow control on the send channel (recommended).
Set to '0' to ignore flow control imposed on a receiver.

useCRC Set to '1' to use CRC generation/checking
(recommended).
Set to '0' to not use CRC checking.

All of these parameters are unsigned 32-bit integers.

4.4.2 fxsl_statusstruct

The type *fxsl_statusstruct* is used to return the driver's status. See section 5.6 for an example. The user-accessible structure members are:

driver_revision_str..... 128-character string containing the driver revision.
nBoard Unit number of the board.
nBus..... PCI bus number where the card was found.
nSlot PCI slot number where the card was found.
bLinkUp..... Set to '1' if the link is up, Set to '0' if it is not.
num_link_errors Number of link errors since last time it was read. Resets to zero each time it is read.
lcr..... Value of link control register.
lsr..... Value of link status register.
ffr..... Value of FPDP flags register.
fttr..... Value of the FIFO threshold register.
revisionID Firmware revision ID.

All of the above fields are unsigned 32-bit integers unless otherwise specified.

4.5 Time-out Operation

Time-out operation of *fxsl_send()* and *fxsl_rcv()* as well as the meaning of **max_send_timeout** and **max_rcv_timeout** variables from **fxsl_configstruct** structure have changed from revision drivers. The new functionality is as follows:

1. **max_rcv_timeout** becomes a reserved field which may receive new meaning and functionality in future driver releases,
2. **max_send_timeout** field becomes effectively a **max_timeout** common to both send and receive directions.
3. **max_send_timeout** (virtual **max_timeout**) is initialized during the driver load to a value defined as:

```
#define FXSL_DEF_MAX_SEND_TIMEOUT 600 /* 6 seconds */
```

Corresponding to more or less to 6-second time-out.

4. **iTimeout** parameter passed as a value in *fxsl_send()* and *fxsl_rcv()* calls is subjected to the following check and corrections resulting in effective time-out applied to a particular call instance:

```
if(iTimeout <=0) or
if(iTimeout > max_send_timeout (virtual max_timeout))
    effective timeout = FXSL_DEF_MAX_SEND_TIMEOUT
    /* about 6 seconds */
otherwise
    effective timeout = iTimeout;
```

It follows that if time-outs larger than six seconds are required, a default value present in **max_send_timeout** in **fxsl_configstruct** structure has to be modified first. It can be done through *fxsl_get_config()*/*fxsl_set_config()* call sequence or through the **sl_mon** application

That is:

```
sl_mon unit# tmax new_timeout_value
```

There are certain additional limits on effective time-out value. These limits are imposed by a particular operating system and these limits are not the same for every operating system. Extending **max_send_timeout** (virtual max time-out) beyond these limits may result in unpredictable behavior. For very large time-outs testing prior to actual deployment is recommended.

4.6 API Functions

The SL240 API contains a variety of functions. They allow you to send and receive data, configure the driver, and get driver information in a variety of ways. The function prototypes are found in the file **fxslapi.h** (located in the *inc* directory) and detailed below. All functions return **FXSL_SUCCESS** upon successful completion. For a complete description of all return codes see section 4.7.

Table 4-1 API Functions

Function	Description	Page
<i>fxsl_open()</i>	Returns a handle (through the pHandle variable) to a specific SL240 adapter that is then passed to all other API calls that will operate on that device.	4-7
<i>fxsl_close()</i>	Closes a handle to a SL240 adapter.	4-8
<i>fxsl_recv()</i>	Retrieves data from the SL240 receive FIFO.	4-9
<i>fxsl_send()</i>	Sends data to the SL240 transmit FIFO.	4-11
<i>fxsl_status()</i>	Retrieves information about the current status of an SL240 adapter.	4-12
<i>fxsl_get_config()</i>	Retrieves the current configuration of a driver associated with an SL240 adapter.	4-13
<i>fxsl_set_config()</i>	Sets the current configuration of a driver associated with an SL240 adapter.	4-14
<i>fxsl_fdp_get()</i>	Retrieves a copy of the FDP status/control line bits.	4-15
<i>fxsl_fdp_put()</i>	Sets the FDP status/control line bits.	4-16
<i>fxsl_read_CR()</i>	Reads an SL240 run-time register.	4-17
<i>fxsl_write_CR()</i>	Writes to an SL240 run-time register.	4-18

4.6.1 fxsl_open

FUNCTION PROTOTYPE:

```
fx_ulong fxsl_open (fx_ulong nUnit, HANDLE *pHandle);
```

DESCRIPTION:

Returns a handle (through the **pHandle** variable) to a specific SL240 adapter that is then passed to all other API calls that will operate on that device.

INPUT:

nUnit Ordinal number of an SL240 adapter to return a handle to.

OUTPUT:

***pHandle** Contains an operating-system-specific handle to the driver.

ERROR CODES:

FXSL_SYSTEM_ERROR Specified unit could not be opened.

FXSL_BAD_PARAMETER **nUnit** is greater than 16.
pHandle is NULL.

4.6.2 fxsl_close

FUNCTION PROTOTYPE:

fx_ulong *fxsl_close* (*HANDLE* *hDevice*);

DESCRIPTION:

Closes a handle to a SL240 adapter.

INPUT:

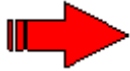
hDevice The handle to a driver instance to close.

OUTPUT:

None.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).



NOTE: The *HANDLE* becomes invalid once closed, and will cause system errors if any further use of the *HANDLE* is attempted.

4.6.3 fxsl_recv

FUNCTION PROTOTYPE:

```
fx_ulong fxsl_recv (HANDLE hDevice,
                   fx_ubyte *pBuf,
                   fx_ulong nBytes,
                   fx_ulong *pFlags,
                   fx_long iTimeOut,
                   fx_ulong *pTransferSize);
```

DESCRIPTION:

Retrieves data from the SL240 receive FIFO.

INPUT:

hDevice Handle to a device driver, returned by a call to *fxsl_open()*.

pBuf Pointer to a user buffer.

nBytes Size of the buffer, in bytes. Must be a multiple of four.

pFlags..... Pointer used to pass transaction flags into and out of the driver. Valid flags to pass into the API call are listed in section 4.2.2.

iTimeOut See section 4.5 Time-out Operation.

OUTPUT:

***pFlags**..... The **FXSL_USE_SYNC** or **FXSL_USE_SYNC_DVALID** bit will be set if the receive was completed because of a SYNC or a SYNC with DVALID, respectively.

***pTransferSize**..... Contains the number of bytes received.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
pBuf is NULL.
nBytes is not a multiple of 4.
pTransferSize is NULL.



NOTE: The SL240 API only checks that the parameters are not NULL.

FXSL_INSUFFICIENT_RESOURCES.... The driver did not have enough resources to complete the transaction.

FXSL_TIMEOUT The transaction timed out before completing.

FXSL_DRIVER_ERROR..... An internal driver error occurred during the transfer.

FXSL_LINK_ERROR..... A link error occurred during the transfer.

FXSL_CRC_ERROR..... A CRC error was detected in the received data during or just before this transaction.

FXSL_FIFO_OVERFLOW The receiver FIFO overflowed during or just before this transaction.

4.6.4 fxsl_send

FUNCTION PROTOTYPE:

```
fx_ulong fxsl_send (HANDLE hDevice,
                   fx_ubyte *pBuf,
                   fx_ulong nBytes,
                   fx_ulong *pFlags,
                   fx_long iTimeout,
                   fx_ulong *pTransferSize);
```

DESCRIPTION:

Sends data to the SL240 transmit FIFO

INPUT:

hDevice Handle to a device driver, returned by a call to *fxsl_open*().

pBuf Pointer to a user buffer.

nBytes Size of the buffer, in bytes. Must be a multiple of four.

pFlags..... Pointer used to pass transaction flags into and out of the driver. Valid flags to pass into the API call are listed in section 4.2.2.

iTimeout See section 4.5 Time-out Operation.

OUTPUT:

***pFlags**..... The **FXSL_USE_SYNC** or **FXSL_USE_SYNC_DVALID** bit will be set if SYNC was sent.

***pTransferSize**..... Contains number of bytes sent.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
pBuf is NULL
nBytes is not a multiple of 4
pTransferSize is NULL.



NOTE: The SL240 API only checks that the parameters are not NULL.

FXSL_INSUFFICIENT_RESOURCES.... The driver did not have enough resources to complete the transaction.

FXSL_TIMEOUT The transaction timed out before completing.

FXSL_DRIVER_ERROR..... An internal driver error occurred during the transfer.

FXSL_LINK_ERROR..... A link error occurred during the transfer.

4.6.5 fxsl_status

FUNCTION PROTOTYPE:

fx_ulong fxsl_status (*HANDLE* hDevice, *Pfxsl_statusstruct* pStatusStruct);

DESCRIPTION:

Retrieves information about the current status of an SL240 adapter.

INPUT:

hDevice Handle to a device driver instance.

pStatusStruct Address of an SL240 status structure.

OUTPUT:

pStatusStruct Allocated status structure updated.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
pStatusStruct is NULL.

4.6.6 fxsl_get_config

FUNCTION PROTOTYPE:

fx_ulong fxsl_get_config (*HANDLE* hDevice, *Pfxsl_configstruct* pConfigStruct);

DESCRIPTION:

Retrieves the current configuration of a driver associated with an SL240 adapter.

INPUT:

hDevice Handle to a device driver instance.
pConfigStruct Address of an SL240 configuration structure.

OUTPUT:

pConfigStruct Allocated configuration structure updated.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
pConfigStruct is NULL.

4.6.7 fxsl_set_config

FUNCTION PROTOTYPE:

fx_ulong fxsl_set_config (*HANDLE* hDevice, *Pfxsl_configstruct* pConfigStruct);

DESCRIPTION:

Sets the current configuration of a driver associated with an SL240 adapter.

INPUT:

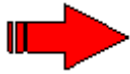
hDevice Handle to a device driver instance.
pConfigStruct Address of an SL240 configuration structure.

OUTPUT:

None.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
pConfigStruct is NULL.
pConfigStruct member **useCRC** is not 1 or 0.



NOTE: In order to change the current configuration of a driver, first call *fxsl_get_config()*, change the desired structure members, and then call *fxsl_set_config()* to enforce the changes.

4.6.8 fxsl_fdp_get

FUNCTION PROTOTYPE:

```
fx_ulong fxsl_fdp_get (HANDLE hDevice, fx_ulong *pVal);
```

DESCRIPTION:

Retrieves a copy of the FPDP status/control line bits.

INPUT:

hDevice Handle to a device driver instance.
pVal..... Pointer to a 32-bit integer to hold the values of the FPDP lines on the SL240 receive channel (bitmasked). The values may be extracted using the constants described in section 4.2.3.

OUTPUT:

***pVal**..... Contains bit values of FPDP lines.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
pVal is NULL.

4.6.9 fxsl_fdpd_put

FUNCTION PROTOTYPE:

fx_ulong fxsl_fdpd_put (*HANDLE* hDevice, *fx_ulong* fVal);

DESCRIPTION:

Sets the FPDP status/control line bits.

INPUT:

hDevice Handle to a device driver instance.
fVal..... A 32-bit integer containing the values of the FPDP lines to set on the SL240. The available bits are described in section 4.2.3.

OUTPUT:

None.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0, -1, etc.).
fVal specifies an invalid FPDP signal.

4.6.10 fxsl_read_CR

FUNCTION PROTOTYPE

*fxsl_read_CR(HANDLE hDevice, fx_ulong offset, fx_ulong *pval)*

DESCRIPTION

Reads an SL240 run-time register.

INPUT:

hDevice Handle to a device driver instance.
offset..... Offset of register to be read.
***pval**..... Pointer to a 32-bit integer variable to store the value read from the SL240 run-time register specified by **offset**.

OUTPUT:

***pval**..... Contains the value of a SL240 run-time register.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0,-1, etc)
offset is out of range.

4.6.11 fxsl_write_CR

FUNCTION PROTOTYPE

fxsl_write_CR(HANDLE hDevice, fx_ulong offset, fx_ulong val)

DESCRIPTION

Writes to an SL240 run-time register.

INPUT:

hDevice Handle to a device driver instance.
offset..... Offset of register to be written.
val..... 32-bit integer variable containing a value to be written to a run-time register specified by **offset**.

ERROR CODES:

FXSL_BAD_PARAMETER..... **hDevice** is invalid (0,-1, etc)
offset is out of range.



CAUTION: It is highly recommended that *fxsl_read_CR()* be called prior to calling *fxsl_write_CR()*. Improper use of *fxsl_write_CR()* may place the SL240 card in an unusable state and may require that power be cycled to the host to reset the card

4.7 Return Codes

Each function in the SL240 API returns either **FXSL_SUCCESS** or an error code. Each possible return code is listed below and followed by a description:

Table 4-2 Return Codes

Value	Name	Description
0	FXSL_SUCCESS	Returns when the call completes successfully.
1	FXSL_SYSTEM_ERROR	Returns when a system call returns an error.
2	FXSL_BAD_PARAMETER	Returns when an invalid parameter is received by an API call. An invalid parameter can be a value outside the specified range or a NULL pointer.
3	FXSL_LINK_ERROR	Returns when a call cannot be completed because of a link error.
4	FXSL_NO_DATA	Returns when a call cannot be complete because no data is available.
5	FXSL_BUSY	Reserved
6	FXSL_DRIVER_ERROR	Returns when the driver encounters an internal error.
7	FXSL_TIMEOUT	Returns when the time-out expires before the call completes.
8	FXSL_CALL_UNSUPPORTED	Returns when the requested call is unsupported. The driver priority may not be configurable on all platforms. If the priority cannot be changed, this error will also be returned.
9	FXSL_INSUFFICIENT_RESOURCES	Returns when resources are not available to complete a call.
10	FXSL_CRC_ERROR	Returns when a CRC error is detected during a receive.
11	FXSL_FIFO_OVERFLOW	Returns when Receive FIFO overflows.

5. API EXAMPLES

5.1 Installing the SL240 Device Driver

Installing the SL240 device driver is operating-system specific. Please consult the proper SL240 software installation manual for your operating system. Source code is provided for the following examples in the examples sub-directory of the SL240 software distribution.

5.2 Opening and Closing a Handle to the Driver

Use the function *fxsl_open()* to open a handle to the SL240 device driver. Use the function *fxsl_close()* to close the handle to the SL240 device driver. An example of this is shown below:

```
#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void fxsl_open_and_close_example (fx_ulong unit)
{
    HANDLE handle;

    if (fxsl_open (unit, &handle) != FXSL_SUCCESS)
    {
        printf("ERROR: could not open unit %d!\n",
            unit);
        return;
    }

    /* Perform operations using handle */

    if (fxsl_close (handle) != FXSL_SUCCESS)
    {
        printf("ERROR: could not close unit %d!\n",
            unit);
        return;
    }
}
```

5.3 Configuring the Driver

To configure the driver:

- Declare an *fxsl_configstruct*.
- Fill the configuration structure using *fxsl_get_config*.
- Modify the values in the configuration structure that are to be changed.
- Call *fxsl_set_config* to reconfigure the driver.

For example, to change the driver's mode of operation, use the following code:

```
#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void fxsl_configuration_example (fx_ulong unit)
{
    HANDLE          handle
    fxsl_configstruct  cfg;

    if (fxsl_open (unit, &handle) != FXSL_SUCCESS)
    {
        printf("ERROR:  could not open unit %d!\n",
            unit);
        return;
    }

    /* get the current configuration */
    if (fxsl_get_config (handle, &cfg) != FXSL_SUCCESS)
    {
        printf("ERROR:  could not get configuration!\n");
        return;
    }

    /* change the desired parameters */
    cfg.use_flow_control = 1;

    /* set the new configuration */
    if (fxsl_set_config (handle, &cfg) != FXSL_SUCCESS)
    {
        printf("ERROR:  could not set configuration!\n");
        return;
    }

    if (fxsl_close(handle) != FXSL_SUCCESS)
    {
        printf("ERROR:  could not close unit %d!\n",
            unit);
        return;
    }
}
```

5.4 Sending Data

The API function `fxsl_send()` is used to send a buffer of data. A basic send transaction is shown below:

```
#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void FXSL_send_example (HANDLE handle, fx_ulong buf_size)
{
    char    *buf;
    int     i;
    fx_ulong bytes_sent,
           flags;

    /* allocate a buffer */
    buf = malloc(buf_size);
    if (buf == NULL)
    {
        printf("ERROR: could not allocate memory!\n");
        return;
    }
    for (i=0; i<buf_size; i++)
    {
        buf[i] = i;
    }

    /* set up variables used in the send */
    flags = 0;

    /* perform the send */
    if (fxsl_send (handle, buf, buf_size,
                  &flags, 0, &bytes_sent) != FXSL_SUCCESS)
    {
        printf("ERROR: send command failed!\n");
    }
    else
    {
        printf("Transferred %d bytes.\n", bytes_sent);
    }
    free(buf);
}
```



NOTE: Some platforms require buffers to be aligned. It is recommended that more memory be allocated than actually needed so the user application can perform proper buffer alignment.

The previous example sends the buffer without any flags or time-out. In the SL240 API time-outs are specified in 1/100ths of a second. To use a time-out of one second and end the transaction with a SYNC, use `fxsl_send()` like this:

```
flags = FXSL_USE_SYNC;
fxsl_send(handle, buf, buf_size, &flags, 100, &bytes_sent)
```

5.5 Receiving Data

The API function `fxsl_rcv()` is used to receive data. A basic call to `fxsl_rcv()` is shown below:

```
#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void fxsl_rcv_example (HANDLE handle, fx_ulong buf_size)
{
    char      *buf;
    fx_ulong  bytes_rcv,
             flags;

    /* allocate a buffer */
    buf = malloc(buf_size + 4);
    if (buf == NULL)
    {
        printf("ERROR: could not allocate memory!\n");
        return;
    }

    /* set up variables used in the receive */
    flags = 0;

    /* perform the receive */
    if (fxsl_rcv (handle, buf, buf_size,
                 &flags, 0, &bytes_rcv) != FXSL_SUCCESS)
    {
        printf("ERROR: receive command failed!\n");
    }
    else
    {
        printf("Received %d bytes.\n", bytes_rcv);
    }
    free(buf);
}
```

The previous example does not use any flags or time-outs. To use a SYNC to signify the end of a data buffer, perform the following read:

```

/* set up variables used in the receive */
flags = FXSL_USE_SYNC;

/* perform the receive */
if (fxsl_rcv (handle, buf, buf_size + 4), &flags, 0
    &bytes_rcv) != FXSL_SUCCESS)
{
    printf("ERROR: receive command failed!\n");
    return;
}
else
{
    if (flags & FXSL_USE_SYNC)
    {
        printf("Received %d bytes and sync.\n",
            bytes_rcv);
    }
    else
    {
        printf("Received %d bytes and no sync.\n",
            bytes_rcv);
    }
}
}

```

When using a SYNC flag on a receive, the flags are cleared when the transaction is started. If a SYNC is received, the **FXSL_USE_SYNC** bit is set. If no SYNC is received, the **FXSL_USE_SYNC** bit will be cleared. This allows you to know if a receive was completed because a SYNC was received.



NOTE: When receiving data with SYNC, it is recommended that at least four bytes more than the largest expected frame (buffer size) be allocated. This is to avoid a possible race condition between “buffer full” and “SYNC received” transaction completion status.

5.6 Using Status Information

To get the status of the card and driver:

- Declare an *fxsl_statusstruct*.
- Update the status structure using *fxsl_status*.

The API provides the call, *fxsl_status()*, to retrieve the driver's status. A sample call to this function is shown below:

```
#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void fxsl_status_example (HANDLE handle)
{
    fxsl_statusstruct status;

    if (fxsl_status (handle, &status) != FXSL_SUCCESS)
    {
        printf("ERROR: call to fxsl_status failed!\n");
        return;
    }
    else
    {
        printf("board %d found in PCI bus %d slot %d\n",
            status.nBoard, status.nBus, status.nSlot);
        printf("  Driver: %s\n", status.driver_revision_str );
        if (status.bLinkUp)
        {
            printf("    Link is up.\n");
        }
        else
        {
            printf("    Link is down!!!\n");
        }
        printf("    LCR: 0x%8.8x LSR: 0x%8.8x\n",
            status.lcr, status.lsr);
        printf("    FFR: 0x%8.8x FTR: 0x%8.8x\n",
            status.ffr, status.ftr);
        return;
    }
}
```

5.7 Using FPDP Lines

The front panel data port (FPDP) lines in the SL240 hardware are intended to allow an SL240 adapter to communicate, through Curtiss-Wright Controls Inc. FPDP to SL240 converter, with a DSP using FPDP. The available FPDP signals that can be sent and received by the PCI/PMC cards are:

- PIO1
- PIO2
- DIR
- NRDY

To read the current FPDP values call *fxsl_fpdp_get()* and to set the FPDP values use *fxsl_fpdp_put()*. The values read are the FPDP values being received by the card. The values set are the FPDP values being sent from the card. The following example shows how to read and write FPDP values (see also Section 4.2.3).

```
#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void FXSL_fpdp_example (HANDLE handle)
{
    fx_ulong val;

    /* read the FPDP values */
    if (fxsl_fpdp_get (handle, &val) != FXSL_SUCCESS)
    {
        printf("ERROR: can not read FPDP values!\n");
        return;
    }
    else
    {
        if (!(val & FXSL_FPDP_MASK_IN))
        {
            printf("FPDP: no lines set.\n");
        }
        else
        {
            printf("FPDP: ");
            if (val & FXSL_FPDP_PIO1_IN)
            {
                printf("PIO1 ");
            }
            if (val & FXSL_FPDP_PIO2_IN)
            {
                printf("PIO2 ");
            }
            if (val & FXSL_FPDP_DIR_IN)
            {
                printf("DIR ");
            }
            if (val & FXSL_FPDP_NRDY_IN)
            {
                printf("NRDY ");
            }
            printf("\n");
        }
    }
}
```

```

/* set the values to send */
val = FXSL_FPDP_PIO1_OUT;
/* send the FPDP values */
if (fxsl_fdpdp_put (handle, val) != FXSL_SUCCESS)
{
    printf("ERROR: can not set FPDP values!\n");
    return;
}
else
{
    printf("FPDP values have been set\n");
    return;
}
}

```

5.8 Using SL240 Run-Time Registers

To read or write an SL240 register, call either *fxsl_read_CR()* or *fxsl_write_CR()*. Sample calls to these functions are shown below.

```

#include "inc/fxsl.h"
#include "inc/fxslapi.h"

fx_ulong fxsl_read_CR_example (HANDLE handle, fx_ulong offset)
{
    fx_long regval;
    if (fxsl_read_CR(handle, offset, &regval) !=
FXSL_BAD_PARAMETER);
    {
        printf("Read %lx from SL240 Runtime Register %x
\n",regval,offset);
    }
    else
    {
        printf("fxsl_read_CR() failed\n");
        return;
    }
    return regval;
}

```

```

#include "inc/fxsl.h"
#include "inc/fxslapi.h"

void fxsl_write_CR_example (HANDLE handle, fx_ulong offset,
fx_ulong val)
{
    if(fxsl_write_CR (handle, offset, val)) != FXSL_BAD_PARAMETER
    {
        printf("Wrote %lx to register %x \n", val ,offset);
    }
    else
    {
        printf("fxsl_write_CR() failed \n");
        return;
    }
    return;
}

```


5.9 API Procedure Flow

The following is an example of an API procedure call sequence from a point-to-point configuration.

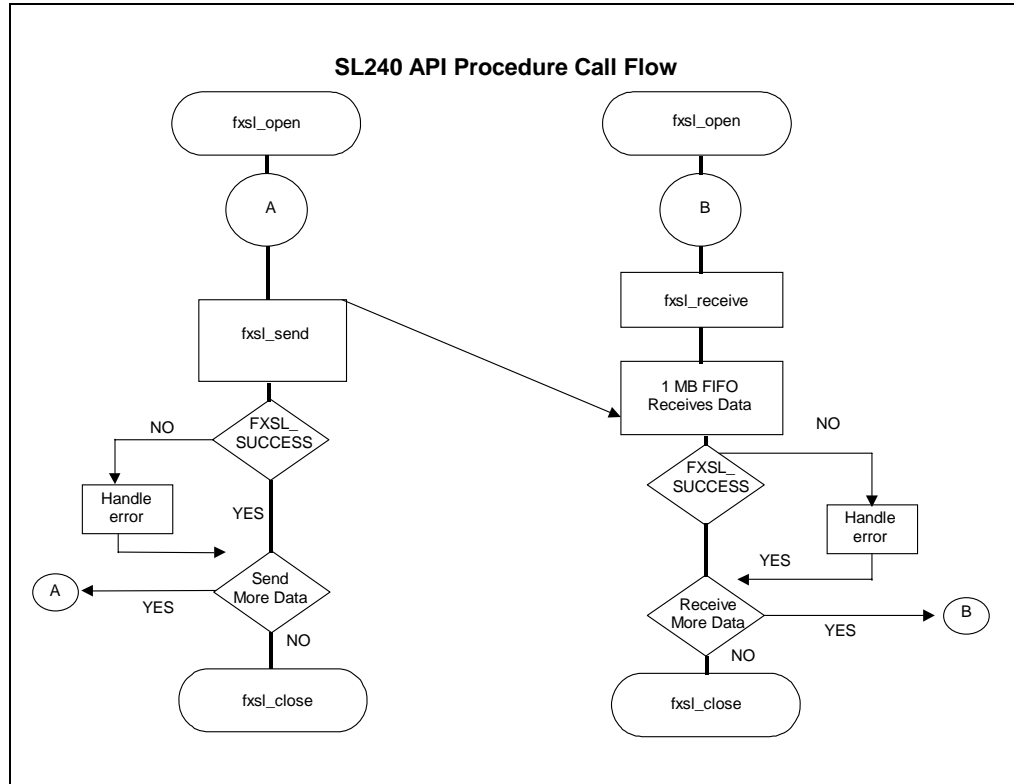


Figure 5-1 API Procedure Call Sequence

5.10 Additional Topics

This section discusses using multiple threads to increase throughput and using the FPDP signals for additional control capabilities. These topics may not be relevant to many applications and may be skipped.

5.10.1 Using Multiple Threads

A thread is defined here as a basic object that executes instructions on a processor. Thread implementation is operating-system specific. That is, Windows NT thread implementation is different from VxWorks or UNIX thread implementation.

The SL240 software architecture is designed to handle multiple send and/or receive requests simultaneously. The *fxsl_send()* and *fxsl_recv()* API calls are synchronous or blocking which means that they do not return to the caller until all the data has been sent/received or some error occurs. The synchronous nature of the API calls requires that either multiple threads of multiple applications must be used in order to actually make multiple requests simultaneously. An application may increase overall throughput by using multiple threads that will make API calls practically simultaneous. It will be the responsibility of the application to do any necessary thread synchronization in order to insure proper ordering of these requests.

Sustained throughput gains from using multiple threads can be anywhere from 0%-100%. In addition, multiple processor systems may experience even more significant gains from using multiple threads. For an example of how to use multiple threads in an application, see the source code provided for the throughput application (**sltp.c**).

5.10.2 Synchronization using FPDP Lines

The incoming FPDP bits on the SL240 PCI and PMC cards may be used for general-purpose synchronization. These bits do not have any inherent meaning for the PCI and PMC cards and thus may be used for any purpose.

6. UTILITY APPLICATIONS

6.1 Application Overview

A variety of SL240 utility applications are delivered with the SL240 device driver. These samples are provided to assist in verifying the card's functionality, to assist in application development, and to provide examples using the SL240 API.

6.1.1 Running SL240 Applications Under Tornado 2.2 and VxWorks

To load SL240 Applications using the Tornado 2.2, do the following:

1. Logon to the Windows host.
2. Load the Tornado 2.2 IDE.
3. From the Tornado 2.2 IDE select file->Open workspace, and then select Browse to locate the SL240 Tornado 2.2 Applications workspace (sl240apps.wsp) and then click OK to open the workspace.
4. With the SL240 Applications workspace loaded, right click on the individual projects and select download to load the applications onto the target. Figure 6-1 shows the Tornado 2.2 window and menu.

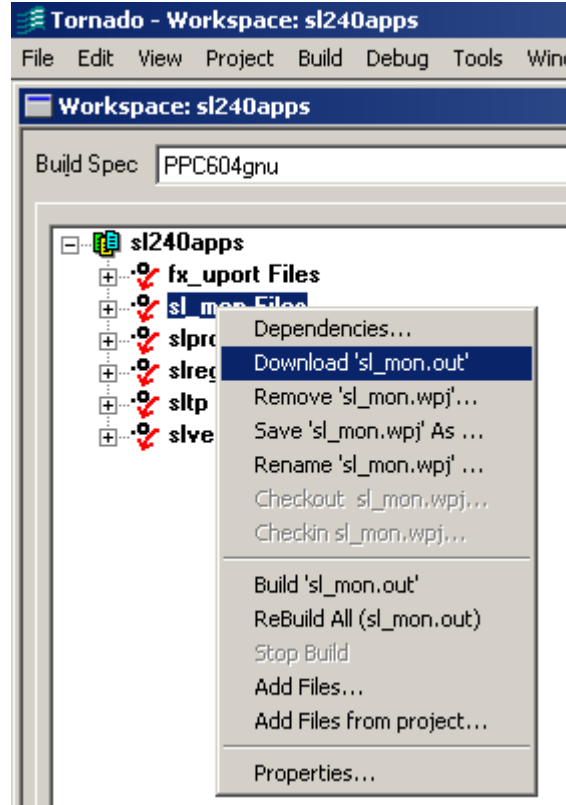


Figure 6-1 Tornado 2.2 Window

Prior to executing SL240 applications under VxWorks, the SL240 API binary must be loaded:

```
ld<fxslapi.o
```

Next load the application binary. For example:

```
ld<sl_mon.o
```

Execute the application enclosing any parameters inside quotes. For example:

```
sl_mon "0"
```

6.2 Monitor Application

The monitor application **sl_mon** allows you to display information about the status of the SL240 driver, to change the driver's configuration, and to use the front panel data port lines.

To execute the monitor run the **sl_mon** application. The application has the following parameters:

unit Unit number to monitor.
command Command to execute.
val Value to be used by the commands (not required for all commands).

The available **command** options are:

a Displays number of installed SL240 devices for this host.
creg Display PCI configuration registers.
rreg Display PCI controller's run-time registers.
clrf Clear all data from the receive FIFO.
fpdp Set the FPDP values, '0', '2', '4', '8' or '0x10' in any combination.
flow Configure flow control;
 '0' is off,
 '1' is on.
sync Configure the reception of SYNC-with-DVALID frames.
 '1' is receive SYNC with DVALID frames,
 '0' is ignore SYNC with DVALID frames.
lerr Configure the handling of link errors, 1, 2, 3 or 4.
 See Table 6-1 for details.
crc Configure CRC generation/checking;
 '0' is no,
 '1' is yes.
swap Byte Swap Data;
 '0' is off,
 '1' is on.
lcfg Set loop configuration mode.
 '0x0' Point-to-Point

'0x1' Initiator
 '0x2' Receive Only Terminator
 '0x3' Terminator
 '0x4' EWRAP for verify
 '0xA' Chain Target
 '0xB' Chain No Tx No Rx

tmaxSet the maximum time-out.

A detailed list of the **command** options and acceptable **val** values for each option are available using the **sl_mon** online help. To display the online help issue the following command:

```
sl_mon h
```

The following output is displayed:

```
Usage:    sl_mon unit [command [val]]
COMMANDS:
[]        Display whole status and configuration
creg      Display PCI configuration registers
rreg      Display PCI runtime (CSR) registers
clrf      Clear (read) receive FIFO until it is cleared
tmax <val> Set Max Timeout to val tics
fpdp <val> Set front panel data port values
           Valid values are any combination of the following:
           0x0 Clear all    0x2 Set PIO1    0x4 Set PIO2
           0x8 Set DIR     0x10 Set NRDY

flow <0>|<1> Set flow of control usage (0=OFF, 1=ON)
crc <0>|<1> Set CRC generation/checking (0=OFF, 1=ON)
swap <0>|<1> Set data byte swaping to (0=OFF, 1=ON)
sync <0>|<1> Receive SYNC with DVALID (0=OFF, 1=ON)
lerr <val> Set link error handling mode to 1, 2, 3 or 4.
           1 No halt, No(?) queuing    2 Halt, No(?) queuing
           3 No halt, Allow queuing    4 Halt, Allow queuing
lcfg <val> Set link configuration value to 0, 1, 2, 3, 4, 0xA or 0xB.
           0x0 Point-to-Point  0x1 Initiator    0x2 RX only Terminator
           0x3 Terminator      0x4 EWRAP        0xA Chain Target
           0xB Chain no TX no RX

Note:    VxWorks use: enclose argument line in " "
```

To display the number of installed SL100/SL240 devices, issue the following command:

```
sl_mon a
```

The following output is displayed:

```
Hardware: unit 0 bus 0 slot 2 - SL100 Firmware 1C.13 (1C.13) for 5.0V PCI (D64)
```

6.2.1 Link Error Handling Mode

The following link error modes are defined for the monitor application.

Table 6-1 Link Error Modes

Mode Number	Parameter Settings
1	halt_on_link_errors = NO
	allow_qing_on_lerror = NO
2	halt_on_link_errors = YES
	allow_qing_on_lerror = NO
3	halt_on_link_errors = NO
	allow_qing_on_lerror = YES
4	halt_on_link_errors = YES
	allow_qing_on_lerror = YES



NOTE: Revision B and higher SL240 drivers ignore **allow_qing_on_lerror**. Consequently, it is recommended that either parameters 1 and 2 or 3 and 4 be used with **lerr**. All options are listed in Table 6-1 for completeness.

To display the status of the driver, execute **sl_mon** with the unit specified and the command unspecified. To execute, type:

```
sl_mon 0
```

The following output is displayed:

```
FibreXtreme (SL) Monitor (sl_mon) rev. 3.02 (2003/10/06)
Driver:   rev. b2-433333:776764 for NT5.0 with API rev. 2.1
Hardware: unit/bus/slot 0/0/3 - SL240 (D64) Firm. 1C.1b (9C.1b) for 5.0V PCI
Link Control Register (CSR 0x08) = 0x33
Link Status Register (CSR 0x0c) = 0x200      Link is UP
FPDP Flags Register (CSR 0x10) = 0x30000      NR.D.P2.P1.S: i=00000 o=00000
FIFO Threshold Register (CSR 0x14) = 0x0      Int.thr. = 0x0
Data count = 0x0 (0) bytes
Link (and other) Errors = 0
Configurable parameters:
Loop Configuration: 0 (Point-to-Point)
Max Timeout: 600 (6000 ms)
Flow Control: 1 (YES) Halt on link error: 1 (YES)
CRC generate/check: 1 (YES) Allow Queuing on link error: 1 (YES)
Data Byte Swapping: 0 (NO) Receive SYNC with DVALID: 0 (NO)
```

6.3 Throughput Application

The throughput application provides a method for testing the actual SL240 performance in a system. The maximum throughput is 105.2 MB/s for the SL100 and 246 MB/s for the SL240. This, however, can be limited by other factors such as PCI bus throughput, system memory bandwidth, processing power, and other components in the system.

6.3.1 SLTP

The **sltp** application performs send and receive throughput tests. The parameters are:

- d **data process** Set to '1' for data processing (CPU will touch all data)
Set to '0' otherwise
- g Run the test for graphing throughput performance.
- h Display the **sltp** online help
- i Show additional help
- m **mode** mode of operation
2=send
3=receive
- n **numTimes** Number of times to transfer the packet per task.
- p **packet size** Size of packet to transfer (in bytes).
- r **relentless** Continually send/receive fixed transaction size.
- s **SYNC** Set to '0' to disable the use of sync after each transfer.
Set to '1' to use **SYNC** after each transfer.
Set to '2' to use **SYNC_DATA_VALID** after each transfer.
- t **numTasks** Number of tasks/threads to use for performing the transfer.
- u **unit** Unit number to test
- w **data print** Set to '1' for writing (printing) receive data to screen
Set to '0' otherwise

To display the **sltp** online help information, issue the following command:

```
sltp -h
```

Figure 6-2 displays the online help output.

```
FibreXtreme (SL) Throughput Test (sltp)
Rev. 3.01 (2003/04/24) for SL240 on NT4.0
Usage: sltp [-u unit] [-m mode] [-p sizePkt] [-n nTimes] [-t numTasks] [-r]
        [-s SYNC] [-d dataProcess] [-v valueToWrite] [-w dataPrint]
        sltp [-u unit] [-m mode] [-g] [-s SYNC] [-d dataProcess]
        sltp [-h | -i]
Defaults:      -u 0 -m 2 -p 131072 -n 1000 -t 2 -s 0 -d 0 -w 0
Parameters:
-u - unit: SL100/240 board/unit number
-m - mode: 2-send (s), 3-receive (r)                DIR
-d - 1 for data processing (CPU will touch all data), 0 otherwise  D
-v - value to write to memory (-d 1 overrides)
-s - 0 = no SYNC. 1 = SYNC. 2 = SYNC w/DVALID      S
-t - number of threads/tasks to use                TSK
-n - number of times to send/receive the packet (per task)      ITER
-p - packet size in bytes                            SIZE
-g - run the test for graphing throughput performance
-r - relentless (repeat test forever - CTRL-C to stop)
-w - 1 for writing (printing) receive data to screen, 0 otherwise

Other reported values:
```

```

- total number of bytes transferred          BYTES
- test time in seconds                      TIME
- number of IO operations per second        IO/s
- number of microseconds per IO            us/IO
- throughput in mega (million) bytes per second  MB/s
Example: sltp -u 0                          - use defaults on unit 0
        sltp -m 3 -p 8 -n 100 -t 3         - receive test
        sltp -m 2 -p 0x100 -n 10000 -t 2  - send test
        sltp -u 0 -g -m 2                  - send throughput graphing on unit
0
Note: VxWorks use: enclose argument line in " "
Note: type sltp -i for more information.

```

Figure 6-2 sltp Online Help Output

For testing SL240 send performance, it is recommended that flow control be ignored. To do so, using the **sl_mon** application, issue the following command:

```
sl_mon 0 flow 0
```

Once testing is completed, re-enable flow control on the SL240 card, by issuing the following command:

```
sl_mon 0 flow 1
```



NOTE: Flow control should be enabled for normal use, including testing SL240 receive performance.

For example, to perform a throughput test with a 128 KB packet to be transferred 1000 times, with two threads, ignoring syncs, issue the following command for transmitter:

```
sltp -u 0 -m 2 -p 131072 -n 1000 -t 2 -s 0
```

```
Testing SEND (keep flow control OFF or have an activate data receiver)
```

DIR	D	S	TSK	ITER	BYTES	TIME	IO/s	us/IO	SIZE	MB/s
=====	=====	=====	=====	=====	=====	=====	=====	=====	=====	=====
._s_	0	0	2	1000	262144000	4.31	462	2155	131072	60.82

The **sltp** application features two default commands that can be issued to quickly check or graph SL240 throughput performance.

To perform a quick throughput transmit test issue the following command:

```
sltp -u 0
```

To produce a graph of SL240 throughput performance, issue the following command:

```
sltp -u 0 -g
```

Figure 6-3 displays the throughput performance data.

```
Testing SEND (keep flow control OFF or have an active data receiver)
```

```
FibreXtreme (SL) Throughput Test (sltp) 3.01 (2003/04/24)
```

```
Driver: rev. b2-322232:776762 Motorola MVME2300 - MPC 604r for VxWorks5.4 (API rev. 2.1)
```

```
Hardware: unit/bus/slot 0/0/16 SL240 (D64) Firm. 1C.13 (9C.13) for 5.0V PCI
```

DIR	D	S	TSK	ITER	BYTES	TIME	IO/s	us/IO	SIZE	MB/s
=====	=====	=====	=====	=====	=====	=====	=====	=====	=====	=====
._s_	0	0	1	80000	320000	1.19	66666	14	4	0.26
._s_	0	0	1	80000	640000	1.24	64000	15	8	0.51
._s_	0	0	1	80000	1280000	1.31	60606	16	16	0.97

s	0	0	1	80000	2560000	1.31	60606	16	32	1.95
s	0	0	1	80000	5120000	1.34	59259	16	64	3.82
s	0	0	1	80000	10240000	1.36	58394	17	128	7.52
s	0	0	1	80000	20480000	1.43	55555	17	256	14.32
s	0	0	1	80000	40960000	1.49	53333	18	512	27.48
s	0	0	1	80000	81920000	1.68	47337	21	1024	48.76
s	0	0	1	80000	163840000	2.11	37735	26	2048	77.64
s	0	0	1	50000	204800000	1.83	27173	36	4096	111.91
s	0	0	1	25000	204800000	1.41	17605	56	8192	145.24
s	0	0	1	12500	204800000	1.19	10416	95	16384	172.10
s	0	0	1	6250	204800000	1.09	5681	174	32768	187.88
s	0	0	1	3125	204800000	1.06	2920	339	65536	193.20
s	0	0	1	1562	204734464	1.03	1501	659	131072	198.77
s	0	0	1	781	204734464	1.01	765	1293	262144	202.70
s	0	0	1	390	204472320	1.01	382	2589	524288	202.44
s	0	0	2	40000	320000	1.38	57553	17	4	0.23
s	0	0	2	40000	640000	1.40	56737	17	8	0.45
s	0	0	2	40000	1280000	1.33	59701	16	16	0.96
s	0	0	2	40000	2560000	1.36	58394	17	32	1.88
s	0	0	2	40000	5120000	1.31	60606	16	64	3.90
s	0	0	2	40000	10240000	1.33	59701	16	128	7.69
s	0	0	2	40000	20480000	1.36	58394	17	256	15.05
s	0	0	2	40000	40960000	1.50	52980	18	512	27.30
s	0	0	2	40000	81920000	1.83	43478	22	1024	44.76
s	0	0	2	40000	163840000	2.53	31496	31	2048	64.75
s	0	0	2	25000	204800000	2.19	22727	43	4096	93.51
s	0	0	2	12500	204800000	1.68	14792	67	8192	121.90
s	0	0	2	6250	204800000	1.36	9124	108	16384	150.58
s	0	0	2	3125	204800000	1.20	5165	192	32768	170.66
s	0	0	2	1562	204734464	1.09	2840	348	65536	187.82
s	0	0	2	781	204734464	1.04	1487	665	131072	196.86
s	0	0	2	390	204472320	1.03	750	1320	262144	198.51
s	0	0	2	195	204472320	1.01	382	2589	524288	202.44
s	0	0	3	26666	319992	1.38	57552	17	4	0.23
s	0	0	3	26666	639984	1.40	56736	17	8	0.45
s	0	0	3	26666	1279968	1.34	59257	16	16	0.95
s	0	0	3	26666	2559936	1.36	58392	17	32	1.88
s	0	0	3	26666	5119872	1.33	59700	16	64	3.84
s	0	0	3	26666	10239744	1.33	59700	16	128	7.69
s	0	0	3	26666	20479488	1.36	58392	17	256	15.05
s	0	0	3	26666	40958976	1.50	52978	18	512	27.30
s	0	0	3	26666	81917952	1.85	43009	23	1024	44.27
s	0	0	3	26666	163835904	2.51	31745	31	2048	65.27
s	0	0	3	16666	204791808	2.19	22726	43	4096	93.51
s	0	0	3	8333	204791808	1.70	14619	68	8192	120.46
s	0	0	3	4166	204767232	1.36	9122	108	16384	150.56
s	0	0	3	2083	204767232	1.18	5251	188	32768	173.53
s	0	0	3	1041	204668928	1.09	2839	349	65536	187.76
s	0	0	3	520	204472320	1.04	1485	666	131072	196.60
s	0	0	3	260	204472320	1.03	750	1320	262144	198.51
s	0	0	3	130	204472320	1.01	382	2589	524288	202.44

Figure 6-3 SL240 Throughput Data

6.3.2 SL240 Versatile Exerciser (slvex)

slvex is a data verification application that allows random packet sizes to be transferred to SL240 cards and verified. **slvex** accepts the following parameters:

- c **CRC check**.....CRC mode:
 0=no CRC check
 1=CRC check
- d **data verify**.....Verify data transferred to SL240 valid verify options are:
 0=no data verification
 1=data verification
- e **timeout base**.....Internal time-out multiplier.
 Recommend values from 10- 20.
- f **flow control**.....Selects whether flow control is enabled or disabled:
 0 disables flow control,
 1 enables flow control
- hDisplays the online help.
- l **loop configuration**.....SL240 loop configuration (topology)
- m **mode**.....Mode of operation:
 0=duplex
 1=receive slow
 2=transmit only
 3=receive fast
- p **min packet size**Minimum packet size in bytes.
- s **data pattern scheme**....Selects whether data pattern is sequential or random,
 0 enables random transfer and
 1 enables sequential transfer.
- u **unit**SL240 unit to test.
- v **packet variation**.....Package size variation to use in bytes
- w **SYNC**Set to '0' to send/receive a SYNC after each transfer.
 Set to '1' to send/receive a SYNC with DVALID after
 each transfer.

To display the **slvex** online help, issue the following command:

```
slvex -h
```

```
Usage: slvex [-u unit] [-m mode] [-p minPkt] [-v varPkt] [-d dVerify] [-w
syncDValid]
           [-c useCRC] [-e tBase] [-l loopConf] [-s sequential] [-f useFlow]
           slvex [-h]
Defaults:   -u 0 -m 0 -p 4096 -v 0 -d 1 -w 0 -c 1 -e 10 -l 0 -s 0 -f 1
Basic parameters:
  -u - SL100/240 board/unit [number]
  -m - mode: 0-DUPLEX 1-RECEIVE_SLOW 2-SEND_FAST 3-RECEIVE_FAST 4-SEND_SLOW
  -p - minimum packet size in bytes (will be rounded to multiple of 4)
  -v - positive packet size variation in bytes (as the above)
  -d - dataVerify: 1=yes, 0-no (for faster operation or multiple of apps)
Extended parameters:
  -w - SYNC: 1-use SYNC w/DVALID, 0-use SYNC
  -c - CRC: 1-use CRC, 0-do not use CRC
  -e - timeouts Base (10 or 20 will do nice, be patient if larger)
  -l - loopConfig: 0-p-p 1-Initiator 2-Rx Only 4-ewrap 10-chained 11-noTx
noRx
  -s - data pattern: 1-sequential, 0-random
  -f - flow control: 1=yes, 0-no
  -t - (unsupported) application execution timeout [seconds]
```

```

Example: slvex -u 0          - duplex transfers using slow receiver
         slvex -m 3 -p 16 -v 128000 - fast receiver
         slvex -m 2 -p 16 -v 128000 - fast transmitter
Note: Running more than one sender or receiver per unit may result in greater
      performance though out of order semi-error messages (if on) may appear.
Note: VxWorks use: enclose argument line in " "

```

Figure 6-4 slvex Online Help output

To execute **slvex** on unit 0 with default parameters, issue the following command:

```
slvex -u 0
```

The runtime output of **slvex** is shown in Figure 6-5

```

sl240 - slvex -u 0
C:\program files\sl240\bin>slvex -u 0
Running: slvex -u 0 -m 0 -p 4096 -u 0 -d 1 -w 0 -c 1 -e 10 -l 0 -s 0 -f 1
Min/Max Packet Size = 4096/4096 [0x1000/0x1000] bytes
Press 'r' to restart or 'q' to quit.
Press 'h' for help on other runtime functions.
dT=44 pTx=3999 pRx=3999 pE=0 pT=0 -- pTx/s=88.9 pRx/s=88.9

```

Figure 6-5 slvex Runtime Output

The runtime help information can be display by typing 'h' during program execution. Each option may be toggled by typing the first character of the option and pressing **E** (Example: to toggle 'showGood frames' press the **G** key and press **E**).

Figure 6-6 depicts the output from **slvex** when the 'print' option is selected during program execution:

```

C:\sl240 - slvex -u 0 -v 256000 -m 0
D:\Program Files\sl240\bin>slvex -u 0 -v 256000 -m 0
Running: slvex -u 0 -m 0 -p 4096 -v 256000 -d 1 -w 0 -c 1 -e 10 -l 0 -s 0 -f 1
Min/Max Packet Size = 4096/260096 [0x1000/0x3f800] bytes
Press 'n' to restart or 'q' to quit.
Press 'h' for help on other runtime functions.
pT=4 pTx=295 pRx=295 pE=0 pI=0 -- pIx/s=59.0 pRx/s=59.0
Driver: rev. b2-433333:776764 for NI5.0 (API rev. 2.1)
Hardware: unit/bus/slot 0/0/3 SL240 (D64) Firm. 1C.1b (9C.1b) for 5.0U PCI
FibreXtreme (SL) Versatile Exerciser (slvex) rev. 3.02 (2003/10/06)
ERROR and OTHER STATISTICS - reported on Tue Oct 07 11:33:54 2003
ERR 00 Tx= 380 Rx= 380 ERR 01 Tx= 0 Rx= 0
ERR 02 Tx= 0 Rx= 0 ERR 03 Tx= 0 Rx= 0
ERR 04 Tx= 0 Rx= 0 ERR 05 Tx= 0 Rx= 0
ERR 06 Tx= 0 Rx= 0 ERR 07 Tx= 0 Rx= 0
ERR 08 Tx= 0 Rx= 0 ERR 09 Tx= 0 Rx= 0
ERR 10 Tx= 0 Rx= 0 ERR 11 Tx= 0 Rx= 0
ERR 12 Tx= 0 Rx= 0 ERR 13 Tx= 0 Rx= 0
ERR 14 Tx= 0 Rx= 0 ERR 15 Tx= 0 Rx= 0
ERR 16 Tx= 0 Rx= 0 ERR 17 Tx= 0 Rx= 0
ERR 18 Tx= 0 Rx= 0 ERR 19 Tx= 0 Rx= 0
Codes: 12-DMQ_FAILURE 13-UNDETECTED_DATA_ERROR 14-FRAGMENT
Codes: 15-SYNC_MISSED 16-OVERSIZED_RECEIVE 17-OUT_OF_ORDER
Codes: 18-TR_OTHER_THEN_TIMEOUT 19-SYSTEM_UNKNOWN
dT=4 pTx=380 pRx=380 pE=0 pI=0 -- pIx/s=76.0 pRx/s=76.0
Tx[byte/s]=1.0e+007 Rx[byte/s]=1.0e+007
Min/Max Packet Size = 4096/260096 [0x1000/0x3f800] bytes
Running: -u 0 -m 0 -p 4096 -v 256000 -d 1 -w 0 -c 1 -e 10 -l 0 -s 0 -f 1
dT=14 pTx=1293 pRx=1293 pE=0 pI=0 -- pIx/s=86.2 pRx/s=86.2

```

Figure 6-6 slvex 'print' Option Output



NOTE: All other, undocumented 'error' conditions are for Curtiss-Wright Controls internal use only, and should be ignored unless a Curtiss-Wright Controls Customer Support Engineer requests the information.

EVALUATING SL240 RECEIVE PERFORMANCE

As with SL240 transmit performance, receive throughput, can be limited by other factors such as PCI bus throughput, system memory bandwidth, processing power, and other components in the system. To accurately measure receiver throughput, the maximum transmit performance must first be determined.

To determine SL240 transmit performance, do the following:

1. Run the **slvex** application on the host that will be used to transmit the data with the following parameters:

```
slvex -m 2 -p 256000 -d 0 -f 0
```

The pTx/s runtime output displays the number of packets transmitted per/second. Once, this number levels off (there is no significant increase in the number of packets transmitted per second). Refer to Figure 6-5 for more details. Display the **slvex** runtime statistics by typing 'p' and record the data displayed in the Tx[byte/s] field. This number represents the maximum transmit speed of the host. Exit the **slvex** application by typing 'q' and then enter.

2. Using the **sl_mon** application, issue the following command to clear CRC errors from the receiver's FIFO:

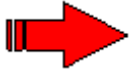
```
sl_mon 0 clrf
```

Issue the command until **sl_mon** reports:

```
sl_mon: 0 Bytes cleared
```

3. On the transmitter start the **slvex** application with the following parameters:

```
slvex -m 2 -d 0 -p 256000
```



NOTE: Do not be concerned when **slvex** reports time-outs at this point. It will not have a significant impact on testing receiver performance.

Execute the **tprx** script/batch file on the receiver:

```
./tprx.sh or tprx
```

6.3.3 SL240 Register (slreg)

The **slreg** application allows the user to display and modify the SL240 control registers. The SL240 register application accepts the following parameters:

- u **unit**..... Unit number
- o **offset** The byte offset of the control register to display or modify.
- c **count** The number of 32 bit registers to display when reading. The default is 1.
- w **data** The 32 bit data pattern to be modified according to the mask and written to the register specified by the offset.
- m **mask** The 32 bit mask used to select the bits from data that are subjected to write.
- h Display the **slreg** online help.



NOTE: The **slreg** application accepts both decimal and hexadecimal input formats. When using hexadecimal the leading 0x is required.



CAUTION: improper use of the **slreg** application could place the SL240 card in an invalid state.

To display the online help, issue the following command:

```
slreg -h
```

```
NAME
  slreg - FibreXtreme SL240 Register Access ver. 3.00 (2002/09/23)

SYNOPSIS
  slreg -o offset [OPTION]...

DESCRIPTION
  Slreg is a front-end application that provides access to SLxxx HW registers.
  Basic register read, write, and read-modify-bit-write operations are
  supported across all platforms. Some options that relate to PCI register
  space as well as transmit/receive FIFO PCI space might be limited to select
  platforms. Slreg operates on 32-bit registers.

  -u unit_no    SLxxx unit number (0-default for the first board in the system).
  -o offset     Byte offset of the register to read/write. If byte offset is not
                long word aligned slreg will perform alignment by truncating two
                least significant bits in the offset. (e.g. 7 will become 4).
  -c count      Number of 32-bit registers to display when reading (default 1).
  -w data       32-bit data pattern to be modified according to -m option and
                written to register specified by -o option. When this option is
                applied, -c option is ignored.
  -m mask       32-bit mask for selecting specific bits to be affected by write
                operation. Option is ignored for read operation. When option is
                not applied, all 32 bits in register at offset specified by -o
                will be modified accordingly. In this case slreg does not
                perform preceding register read.
  -h           This help page is printed (all other options are ignored). With
                no options at all slreg prints short version of this help page.
                For either case neither the presence of the SLxxx hardware nor
                the presence of appropriate driver is required. For both cases
                the slreg revision number (date) is printed.

Defaults:
  slreg -o 0x00 -u 0 -c 1 -m 0xffffffff

Note:
  Slreg accepts numbers in both decimal and hexadecimal (with leading 0x)
  formats. Slreg will print appropriate error messages if requested unit driver
  is not present on the system.

Example:
  To display 16 registers of SLxxx unit 0 starting at offset 0x00:
    slreg -c 16 -o 0x00 (or slreg -c 16 -o 0x00 -u 0 )
  To write to CSR 0x1c of SLxxx unit 1 a value of 0x55:
    slreg -u 1 -o 0x1c -w 0x55
  To write 1 in bit 31 and 0 in bit 8 of CSR 0x4:
    slreg -o 0x04 -w 0x80000000 -m 0x80000100
```

Figure 6-7 slreg Online Help Display

To display all sixteen SL240 control registers using **slreg** issue the following command:

```
slreg -c 16 -o 0x00 -u 0
```

To write the value 0x55 to register 0x1C of unit 0 issue the following command:

```
slreg -u 0 -o 0x1c -w 0x55
```

To write '1' in bit 31 and '0' in bit 8 of CR 0x4 on unit 0 issue the following command:

```
slreg -u 0 -o 0x04 -w 0x80000000 -m 0x80000100
```

The SL240 software also includes scripts/batch files that can be used to perform various operations using **slreg**:

*auto	Automatically run an SL240 application on timed basis
driver	Loads/unloads SL240 driver
*mess	Displays log messages
laser	Toggles SL240 laser off/on
reset	Resets SL240 hardware
setD32	Configure SL240 for 32 bit transfers.
setD64	Configure SL240 for 64-bit transfers.
tprx	Used for testing SL240 receive throughput
tptx	Used for testing SL240 send throughput

* Not available on all SL240 platforms.

These scripts can be executed on UNIX platforms, using the script name, example:

```
./reset.sh
```

These scripts can be executed on Windows platforms, simply by typing the script name, example:

```
setD32
```

These scripts can be executed on VxWorks by using the < symbol followed by the script name, example:

```
<setD32.cmd
```

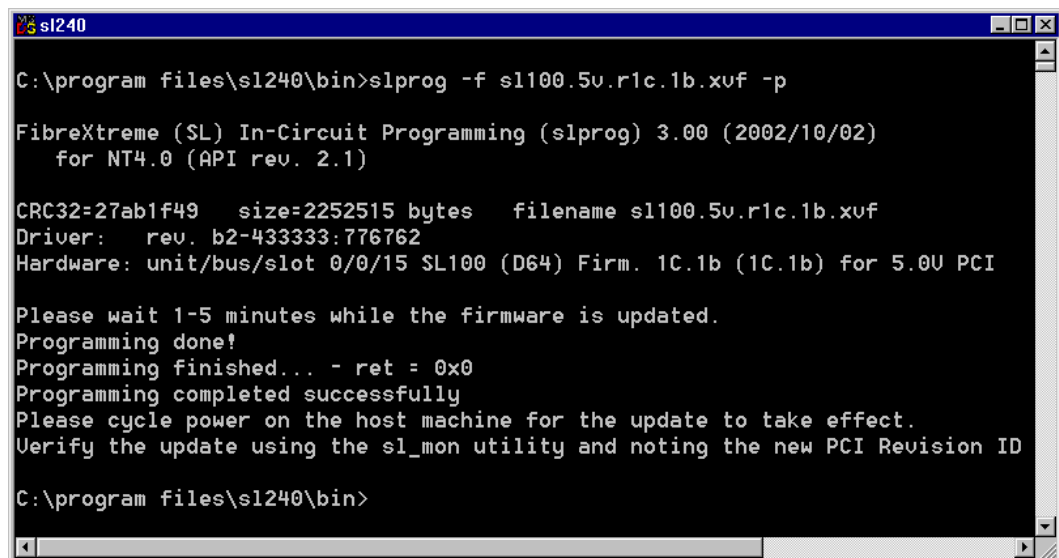
6.4 Firmware Programming Application

The **slprog** program performs a firmware update to the SL240 card.

The parameters for **slprog** are:

- f **file**..... The name of the file containing the firmware update data. The file name may include a directory path as well. This parameter also calculates the checksum for the firmware update file.
- h **help**..... Display **slprog** online help.
- p **proceed** Instructs **slprog** to proceed with firmware update.
- u **unit** Unit number to program.

In the event that the FX SL100/SL240 firmware needs to be updated, you can do so without having to return the cards to the factory. The **slprog** program provides a simple means of updating the card without removing it from the host environment. The firmware update file will be available from the Curtiss-Wright Controls Technical Support. The firmware update can be performed for each card in the system by specifying the appropriate unit number. The file size is reported by the open operation and again by the read operation to verify the complete file has been processed. The programming time varies depending on the host platform. When the programming has completed, the host machine power must be cycled in order for the update to take effect. Figure 6-8 specifies the unit as 0 and the filename as **sl100.5v.r1c.1b.xvf**. Actual units and filenames may vary.



```

C:\program files\sl240\bin>slprog -f sl100.5v.r1c.1b.xvf -p

FibreXtreme (SL) In-Circuit Programming (slprog) 3.00 (2002/10/02)
for NT4.0 (API rev. 2.1)

CRC32=27ab1f49 size=2252515 bytes filename sl100.5v.r1c.1b.xvf
Driver: rev. b2-433333:776762
Hardware: unit/bus/slot 0/0/15 SL100 (D64) Firm. 1C.1b (1C.1b) for 5.0U PCI

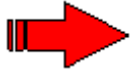
Please wait 1-5 minutes while the firmware is updated.
Programming done!
Programming finished... - ret = 0x0
Programming completed successfully
Please cycle power on the host machine for the update to take effect.
Verify the update using the sl_mon utility and noting the new PCI Revision ID

C:\program files\sl240\bin>

```

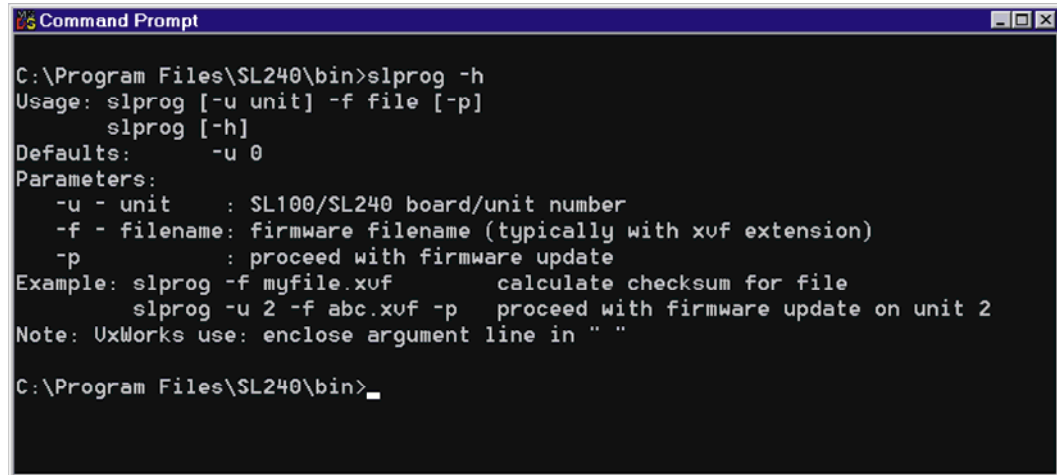
Figure 6-8 Firmware Programming Example

If any errors are encountered, verify the filename, file size, file permissions, and directory path. If the TCP/IP **ftp** utility is used to transfer the file to the host machine, make sure that the binary mode is specified.



NOTE: Before upgrading the firmware, run the **sl_mon** utility and note the current PCI Revision ID. When programming has been completed, power to the host machine must be cycled off and on before the firmware update will take effect. When the machine has been restarted, run the **sl_mon** utility again and verify that the PCI Revision ID has changed.

Figure 6-8 is an example of using the **slprog -h** parameter to display online help.



```
Command Prompt
C:\Program Files\SL240\bin>slprog -h
Usage: slprog [-u unit] -f file [-p]
       slprog [-h]
Defaults:      -u 0
Parameters:
  -u - unit    : SL100/SL240 board/unit number
  -f - filename: firmware filename (typically with xvf extension)
  -p          : proceed with firmware update
Example: slprog -f myfile.xvf          calculate checksum for file
         slprog -u 2 -f abc.xvf -p    proceed with firmware update on unit 2
Note: UxWorks use: enclose argument line in " "
C:\Program Files\SL240\bin>_
```

Figure 6-9 Online help Display

This page intentionally left blank

GLOSSARY

1x3 -----	A 3-pin connector for use with copper media.
8B/10B -----	A data-encoding scheme developed by IBM for translating byte-wide data to an encoded 10-bit format.
AAL5 -----	ATM Adaptation Layer for computer data.
active -----	A term used to denote a port that is receiving a signal.
AL -----	See Arbitrated Loop.
ALPA -----	Arbitrated Loop Physical Address.
ANSI -----	American National Standards Institute.
AP -----	Access Point.
API -----	Applications Program Interface.
APID -----	Access Point Identification Number. A number ranging between 0 and 65535 that is assigned by the user to identify a process. All APID's attached to a single FX board must be unique.
arbitrated loop -----	The simplest form of a Fabric topology. Has shared bandwidth, distributed topology. Interconnects NL_ports/FL_ports at the nodes/Fabric using unidirectional links. It has only one active L_port-L_port connection, so blocking is possible. A fairness algorithm ensures that no L_port is blocked from accessing the loop. Should any link in the loop fail, communication between all L_ports is terminated (see cross-point, point-to-point).
ASIC -----	Application Specific Integrated Circuit. An integrated circuit designed to perform a specific function. ASICs are typically made up of several interconnected building blocks and can be quite large and complex.
ATM -----	Asynchronous Transfer Mode. A network technology that transfers data in small 53-byte packets, and permits transmission over long distances. Proposed speeds range from 25 Mbps to 622 Mbps.
Auto-Speed Negotiation -----	This feature enables FibreXpress FX200 cards to interoperate with existing FC devices at 1.0625 Gbps, and provides seamless transition to higher performance 2.125 Gbps devices.
bandwidth -----	The amount of data that can be transmitted over a channel.
baud -----	A unit of speed in data transmission, usually equal to one bit per second.
BIOS -----	Basic Input/Output System.
bps -----	bits per second.
broadcast -----	Sending a transmission to all nodes on a network.
BSP -----	Board Support Package. A set of software routines written by the OS vendor or SBC vendor that provides support for a particular SBC.
burst transfers -----	Messages are transmitted in a format that includes the initial address followed by all the data. Burst transfers eliminate the need for repeated addresses for each data block, permitting higher throughput.
channel -----	A point-to-point link that transports data from one point to another at the highest speed with the least delay, performing simple error correction in

- hardware. Channels are hardware intensive and have lower overhead than networks. Channels do not have the burden of station management.
- channel network** ----- Combines the best attributes of both channel and network, giving high bandwidth, low latency I/O for client server. Performance is measured in transactions per second instead of packets per second.
- circuit** ----- Bi-directional path allowing communications between two L_Ports.
- circuit-switched mode** ----- Data transfer through a dedicated connection (Class 1).
- CMC** ----- Common Mezzanine Card.
- communications protocol** ----- A special sequence of control characters that are exchanged between a computer and a remote terminal in order to establish synchronous communication.
- CRC** ----- Cyclic Redundancy Check. A code used to check for errors in Fibre Channel.
- cross-point** ----- Provides a bi-directional connection between a node (N_port) and the Fabric (F_port). Can be configured to be non-blocking by providing multiple paths between any two F_ports. Adding stations to a Fabric does not reduce the point-to-point channel bandwidth (see point-to-point).
- datagram** ----- Type of data transfer for Class 3 service. Transfer has no confirmation of receipt and rapid data transmission.
- dBm** ----- decibels relative to one milliwatt.
- direct connect links**----- An actual physical, dedicated connection between two devices with the entire bandwidth available to serve each direct link. Direct links provide a fast and reliable medium for sending large volumes of data.
- DMA** ----- Direct Memory Access.
- DMA write**----- The DMA engine on the bus controller writes the data from the host computer to the SRAM buffer, freeing the host CPU for other tasks. (FibreXpress board becomes a master for the bus.)
- E_Port** ----- Element Port. Used to connect fabric elements together.
- ECL** ----- Emitter Coupled Logic.
- ethernet**----- A widely used shared networking technology.
- exchange**----- One or more sequences for a single operation that are not concurrent, but are grouped together.
- F_Port**----- Fabric Port. The access point of the fabric for physically connecting the user's N_Port.
- fabric**----- A self-managed, active, intelligent switching mechanism that handles routing in Fibre Channel Networks.
- fabric elements**----- Another name for ports.
- FC**----- Fibre Channel.
- FC-AL** ----- Fibre Channel Arbitrated Loop. Provides a low-cost way to attach multiple ports in a loop without hubs and switches.

FCP -----	Fibre Channel Protocol. The mapping of the SCSI communication protocol over Fibre Channel.
FC-PH -----	Fibre Channel Physical interface. Fibre Channel Physical standard, consisting of the three lower levels, FC-0, FC-1, and FC-2.
FCSI -----	Fibre Channel Systems Initiative is made up of IBM, Hewlett-Packard and Sun Microsystems. This group strives to advance Fibre Channel as an affordable, high-speed interconnection standard.
FC-SW -----	Fibre Channel Switch Fabric standard. Formerly known as FC-XS: Fibre Channel Xpoint Switch. The crosspoint-switched fabric topology is the highest-performance Fibre Channel fabric, providing a choice of multiple path routings between pairs of F_ports.
Fibre Channel -----	Fibre Channel (FC) is a serial data transfer interface technology operating at speeds up to 4 Gbps. It is defined as an open standard by ANSI. It operates over copper and fiber optic cabling at distances of up to 10 kilometers. Supported topologies include point-to-point, arbitrated-loop, and fabric switches.
FibreXpress -----	A Curtiss-Wright Controls trademark name for a family of networking products that maximize the superior communication and interconnect capabilities of ANSI standard Fibre Channel. The Dual Channel FX400 series of 64-bit adapters support up to 400 MB per second (800 MB per second duplex) 1600 MBps in combined throughput.
FibreXtreme -----	A Curtiss-Wright Controls trademark name for a family of networking products based on the original Simplex Link technology, Curtiss-Wright's FibreXtreme Serial FPDP Data Link moves data at a sustained 247 MB per second with microsecond latency. Supports up to 2.5 Gbps serial data link using a highly specialized communications protocol optimized for maximum data throughput. Higher throughputs are achieved with the SL240 PCIe and SL240 PCIe XMC.
FIFO -----	first in first out
Firmware -----	Microprocessor executable code, typically for embedded type processors.
Flash -----	A type of Electrical Erasable Programmable Read Only Memory (EEPROM). Erased and written to in blocks vs. bytes.
FL_Port -----	Fabric Loop Port. Joins an arbitrated loop to the fabric.
FPDP -----	Front Panel Data Port.
frame -----	A linear set of transmitted bits that define a basic transport element. A frame is the smallest indivisible packet of data that is sent on the FC.
frame-switched mode -----	Data transfer is connectionless (Classes 2 and 3) and data transmission is in frames. The bandwidth is allocated on a link-by-link basis. Frames from same port are independently switched and may take different paths.
FTP application -----	A test application for transferring files from one computer to another.
FX -----	FibreXpress.
G_Port -----	A port which can function as either an F_Port or an E_Port. Its function is defined at login.

Gbps -----	Gigabits per second.
gigabit -----	One billion bits, or one thousand megabits.
GLM -----	Gigabit per second Link Module. A Link Module that can be used for optical or copper media.
HANDLE -----	Abstraction for the <i>Handle</i> in Windows and <i>file descriptor</i> in Unix.
HBA -----	Host Bus Adapter.
heartbeat -----	A visual indicator that flashes periodically to indicate the embedded controller is functioning properly.
HIPPI -----	High Performance Parallel Interface. An 800 Mbps interface to supercomputer networks (previously called high-speed channel) developed by ANSI.
HSSDC -----	High Speed Serial Data Connectors and Cable Assemblies. A type of high-speed interconnect system which allows for transmission of data rates greater than 2 Gbps and up to 30 meters.
hunt group -----	A group of lines that are linked so that one call to the group will find the line that is free. This provides the ability for more than one port to respond to the same alias address.
I/O -----	Input/Output.
IOCB -----	I/O Control Block. A block of information stored in system memory, usually of fixed length, which contains control codes and data. The IOCB is created by a host computer and sent to some other computer. The IOCB contains command/instructions, data, and memory pointers intended to direct the other computer to perform some function.
inactive -----	A term used to denote a port that is not receiving a signal.
intermix -----	A Fibre-Channel-defined mode of service that reserves the full Fibre Channel bandwidth for a dedicated (Class 1) connection, but also allows connectionless (Class 2) traffic to share the link if the bandwidth is available.
IP -----	Internet Protocol is a data communications protocol.
IPI -----	Intelligent Peripheral Interface.
insertion delay -----	The amount of time the data is delayed for the insertion of FXSL framing protocol. It is measured from when the data becomes available at the FIFO to when the data is actually transmitted on the link. The actual values are either 188 ns in Mode-0 or Mode-1 (with no CRC), or 226 ns in Mode-2 or Mode-3 (with CRC).
KB -----	Kilobytes.
L_Port -----	Loop Port. Either an FL_Port or an NL_Port that supports the arbitrated loop topology.
LAN -----	Local Area Network, typically less than 5 kilometers. Transmissions within a LAN are mostly digital, carrying data at rates above 1 Mbps.
latency -----	The delay between the initiation of data transmission and the receipt of data at its destination.

LCF -----	Link_Control Facility. Provides logical interface between nodes and the rest of Fibre Channel.
Link Module -----	A mezzanine board mounted on the board to interface between the board and the network.
longword -----	32-bit or 4-byte word.
LP -----	Lightweight Protocol.
LX1500 -----	LinkXchange LX1500 Crossbar Switch.
LX2500 -----	LinkXchange LX2500 Crossbar Switch.
Mbps -----	Megabits per second.
MBps -----	Megabytes per second.
MB -----	Megabytes.
media -----	Means of connecting nodes; either fibre optics, coaxial cable or unshielded twisted pair.
ms -----	Milliseconds
mW -----	Milliwatt.
µs -----	Microseconds
monitor -----	An application program used to display the status and change the configuration of the driver.
multicast -----	A single transmission is sent to multiple destination N_ports, a one-to-many transmission. Multicasting provides a way for one host to send packets to a selective group of hosts.
N_Port -----	Node Port. A Fibre-Channel-defined entity at the node end of a link that connects to the fabric via an F-Port.
network -----	Connects a group of nodes, providing the protocol that supports interaction among these nodes. Networks are software intensive, and have high overhead. Networks also operate in an environment of unanticipated connections. Networks have a limited ability to provide the I/O bandwidth required by today's applications and client/server architectures.
NL_Port -----	Node Loop Port. Joins nodes on an arbitrated loop.
node -----	A host computer and interface board. Each processor, disk array, workstation or any computing device is called a node. Connects to FC through a node port (N_Port).
normal write -----	A host CPU writes data to the SRAM buffer through the bus and bus controller (FibreXpress board operates as a slave of the bus).
ns -----	nanoseconds.
NVRAM -----	Non-Volatile Random Access Memory. Generic term for memory that retains its contents when power is turned off.
OFC -----	Open Fibre Control. A safety interlock system used on some FC shortwave links.
one-to-many -----	One node transmits to multiple nodes. See broadcast, multicast.

- operation** ----- One of Fibre Channel's building blocks composed of one or more exchanges.
- out-of-band control** ----- On the LinkXchange products, a method of issuing switch commands that does not use any bandwidth of the 32 switch ports.
- PCB**----- Printed Circuit Board.
- PCI**----- Peripheral Component Interface. A PC bus that allows some expansion boards to communicate directly with the CPU in either 32 bits or 64 bits at a time, this bus also permits multiplexing (more than one electrical signal to be present on the bus at one time).
- PCI-e** ----- PCI-Express. PCI Express is an implementation of the PCI connection standard that uses existing PCI programming concepts, but bases it on a different and faster serial physical-layer communications protocol. PCI Express is a two point serial link where devices do not have to share bandwidth. The standard specifies up to 32 two-point links ("Lanes") per card.
- PECL**----- Positive Emitter Coupled Logic.
- Physical Layer Switch** ----- Multipurpose, non-blocking 288-port cross-point switch (Curtiss-Wright's GLX4000) for digital speeds up to 10 Gbps (See cross-point).
- PIO** ----- Programmed Input/Output.
- PMC**----- PCI Mezzanine Card. Everything that is true for PCI cards is true for PMC except there is a footprint or card format change.
- point-to-point**----- Bi-directional links that interconnect the N_ports of a pair of nodes. Non-blocking.
- port** ----- A physical element through which information passes. It is an electrical or optical interface with a pair of wires or fibers—one each for incoming and outgoing data.
- profiles** ----- Subsets of Fibre Channel standards that improve interoperability and simplify implementation. It is like a cross-section of FC, providing guidelines for implementing a particular application.
- protocols**----- Data transmission conventions encompassing timing, control, formatting, and data representation. This set of hardware and software interfaces in a terminal or computer allow it to transmit over a communication network, and these conventions collectively form a communications language.
- retimed**----- "Retimed" port cards use a phase-locked loop to recover the clock from a serial data stream. They then use the recovered clock to strobe the data through a one-bit latch to minimize the accumulation of edge jitter. This process is sometimes called "reclocked." (Retimed port cards do *not* synchronize the data to a local crystal-controlled reference clock.) Non-retimed port cards do not clock the serial data stream at all. From a timing standpoint, they function as gate delays as the data passes asynchronously through them.
- RISC** ----- Reduced Instruction Set Computer. A type of microprocessor that executes a limited number of instructions that typically allows it to run faster than a Complex Instruction Set Computer (CISC).

RJ-45 -----	Short for Registered Jack-45. An eight-wire connector commonly used to connect computers onto a local-area network (LAN), especially Ethernet. RJ-45 connectors look similar to the RJ-11 connectors used for connecting telephone equipment, but they are somewhat wider.
SAP -----	Service Access Point.
SBC -----	Single Board Computer.
SCSI -----	Small Computer System Interface.
sequence -----	The unit of transfer, made up of one or more related frames for a single operation.
SFF -----	Small Form Factor. Based on SFF MSA.
SFF MSA -----	Small Form Factor Transceiver Multisource Agreement (SFF MSA), July 5, 2000.
shared connect links -----	The ability to send and receive data without establishing a dedicated physical connection so that other devices can also use the medium. This shared link is more efficient for smaller data transmissions because the overhead of direct connect link is avoided.
SRAM -----	Static Random Access Memory.
SRAM Transfer -----	Process in which the data is transferred from the host computer to the SRAM buffer by normal or by DMA write.
SFP -----	Small Form-factor Pluggable based on MultiSource Agreement (MSA), September 14, 2000, FO Transceiver Industry.
STP -----	Shielded Twisted Pair. A type of cable media.
striping -----	To multiply bandwidth by using multiple ports in parallel.
switched fabric -----	(see the definition for “fabric”).
SYNC -----	FibreXtreme Simplex Link primitive used to synchronize the source and destination cards.
SYNC with DVALID -----	A special case of the SYNC primitive occurring in the middle of a buffer of data.
TCP -----	Transmission Control Protocol.
terminal application -----	A test application that sends characters received from the keyboard and displays received characters.
throughput application -----	An application that tests the throughput for the given system.
time-out -----	The time allotted for a native message to travel the network ring and return. If this time is exceeded, an automatic retransmission of the native message occurs.
topology -----	Refers to the order of information flow due to logical and physical arrangement of stations on a network.
ULP -----	Upper Level Protocol.
VHDL -----	Very high-speed integrated circuit Hardware Description Language.
VME -----	Acronym for VERSA-module Europe: bus architecture used in some computers.

XMC ----- A connector(s) on PMC form-factor cards to support high-speed switched interconnect protocols such as PCIe with or without traditional PMC connections.

This page intentionally left blank

INDEX

A

API..... 1-1, 4-5
 function calls..... 4-2
 installation..... 3-1
 library..... 2-5
 procedure call sequence..... 5-9
API calls..... 2-1, 4-6, 4-8, 4-10
 blocking..... 5-10
 synchronous..... 5-10
application
 binary..... 6-2
 development..... 3-3, 6-1
 software..... 2-1
 throughput..... 5-10
application program..... 1-1
applications..... 1-1, 1-3, 2-5, 3-1, 5-10
 binaries..... 3-3
 example..... 4-3
 installation..... 3-1
 load..... 3-4
 rebuild..... 3-4
 SL240..... 3-1
 source..... 3-1
 utility..... 3-3, 6-1

B

bandwidth..... 6-5
binary mode..... 6-14
bit mask..... 4-1, 4-14, 6-11
byte count..... 4-2
byte-swapping..... 4-1

C

chain target..... 4-3, 6-3
channel..... 4-3
 receive..... 4-14
characters..... 4-2
CMC..... 4-2
code
 examples..... 4-1
command.. 5-4, 6-3, 6-4, 6-5, 6-6, 6-8, 6-9, 6-10,
 6-11, 6-12
 failed..... 5-3, 5-5
command prompt..... 3-4
configuration..... 2-1, 2-2, 2-3, 5-2
 driver..... 4-12, 4-13
 loop mode..... 6-2
 point-to-point..... 5-9, 6-2
 registers..... 6-2
 ring..... 2-3

 structure..... 4-12, 4-13, 5-2
 transmitter-receiver..... 2-2
constants..... 1-1, 4-1, 4-2, 4-14
CRC
 checking..... 4-3
 generation/checking..... 4-3, 6-2, 6-3

D

data . 1-1, 2-1, 2-3, 2-4, 4-1, 4-2, 4-3, 4-5, 4-8, 4-
 9, 4-10, 4-18, 5-3, 5-4, 5-5, 5-10, 6-2, 6-3, 6-
 5, 6-6, 6-8, 6-10, 6-11, 6-14
data stream..... 4-1
data types..... 4-1, 4-2
device driver 2-5, 4-1, 4-8, 4-10, 4-11, 4-12, 5-1,
 6-1
 instance..... 4-13, 4-14, 4-15, 4-16
device driverinstance..... 4-17
DIR
 input line..... 4-2
 output line..... 4-2
DMA
 receive..... 4-3
 transactions..... 2-1
driver
 configuration..... 4-3, 4-5, 6-2
 installation..... 3-1
 priority..... 4-18
 revision..... 4-4
 status..... 4-3, 6-2
DSP..... 5-7

E

entry points..... 4-1
error
 CRC..... 4-9, 4-18
 driver..... 4-9, 4-10
 internal..... 4-18
 link..... 4-3, 4-4, 4-9, 4-10, 4-18, 6-2, 6-4
error code..... 4-18
error values..... 4-1

F

features..... 2-1
FibreXtreme..... 1-1
FibreXtreme SL100..... 1-1
FibreXtreme SL240..... 1-1
FIFO
 receive..... 4-5, 4-8, 4-18, 6-2
 receiver..... 4-9
 transmit..... 4-5, 4-10
file size..... 6-14

firmware
 revision..... 4-1
 revision ID 4-4
firmware update 6-14, 6-15
flag parameter 4-1
flow control..... 4-3, 6-2, 6-6
FPDP
 invalid signal..... 4-15
 lines..... 4-14, 5-7
 output lines..... 4-2
 signals 4-2
 status/control line bits 4-5, 4-14, 4-15
 values 5-7, 5-8, 6-2
frame 5-5
ftp utility 6-14
function
 fxsl_close()..... 5-1
function
 fxsl_get_config()..... 4-12
 fxsl_close()..... 4-7
 fxsl_configstruct()..... 5-2
 fxsl_fdp_get()..... 4-14
 fxsl_fdp_put() 4-15
 fxsl_get_config()..... 5-2
 fxsl_open() 4-5, 4-6, 4-8, 5-1
 fxsl_read_CR()..... 4-16
 fxsl_recv..... 4-8
 fxsl_recv()..... 5-4
 fxsl_Recv()..... 4-4
 fxsl_send()..... 4-10, 5-3
 fxsl_Send()..... 4-4
 fxsl_set_config() 4-13, 5-2
 fxsl_status()..... 4-11, 5-6
 fxsl_write_CR() 4-17
 prototype 4-3, 4-5
functions..... 1-1, 2-1, 4-1, 4-5
 calling..... 4-2
 defined 2-5
fxsl_fdp_get 4-2
fxsl_fdp_get()..... 5-7
fxsl_fdp_put 4-2
fxsl_fdp_put() 5-7
fxsl_recv() 5-10
fxsl_send() 5-10
fxsl_status 5-6

H

handle. 4-3, 4-6, 4-7, 4-8, 4-10, 4-11, 4-12, 4-13,
 4-14, 4-15, 4-16, 4-17, 5-1, 5-2, 5-3, 5-4
 operating-system-specific 4-6

I

initiator..... 4-3, 6-3, 6-8
installation
 API..... 3-1
 applications 3-1
 SL240 driver 3-1

L

link control register..... 4-4
Linux 1-2, 4-1

M

makefile 3-4
mask..... 6-11
max_timeout 4-4
memory 4-1, 5-4, 6-5
 allocation 5-3
 memory bandwidth 6-10
mode 6-8
 loop configuration..... 4-3
 mode of operation..... 6-5
monitor 6-2, 6-4
multiple threads 5-10

N

node
 transmitting 2-3
node..... 2-2, 2-3
 receiver 2-4
 receiving 2-4
 re-transmitter 2-4
 re-transmitting 2-4
 transmitter 2-4
 transmitting 2-4

NRDY

 input line 4-2
 output line 4-2
 numbers 4-2

O

offset 4-16, 4-17, 5-8, 6-11
operating system 2-5, 4-5, 5-1
operation 1-1
 mode 5-2
 requests 2-5

P

packet size
 random 6-8
packet size..... 6-8
parameter

- invalid 4-18
- PIO1
 - input line 4-2
 - output line 4-2
- PIO2
 - input line 4-2
 - output line 4-2
- pointer
 - NULL 4-18
- processing power 6-5, 6-10
- R**
- receive operation 4-2
- receiver 4-3, 6-6, 6-9, 6-10
- register 6-11, 6-12
 - FIFO threshold 4-4
 - FPDP flags 4-4
 - link status 4-4
 - offset 4-16, 4-17
 - run time 4-16, 4-17
 - run-time 4-5, 4-16, 4-17
- resources
 - not available 4-18
- re-transmitter 2-3
- return code 4-5, 4-18
- return values 4-1
- runtime output 6-10
- runtime statistics 6-10
- S**
- scripts/batch files 6-12
- semaphore 4-3
- signal
 - operation 4-2
- signals
 - DIR 5-7
 - NRDY 5-7
 - PIO1 5-7
 - PIO2 5-7
 - SYNC 4-8, 4-10, 5-3, 5-5, 6-2
 - SYNC with DVALID ... 4-1, 4-3, 4-8, 6-2, 6-3
- sl_mon
 - a 6-2
 - application. 4-1, 4-4, 4-5, 6-2, 6-3, 6-4, 6-6, 6-10, 6-11, 6-15
 - clrf 6-2
 - command 6-2
 - crc 6-2
 - creg 6-2
 - flow 6-2
 - fpdp 6-2
 - lcfg 6-2
 - lerr 6-2
 - online help 6-3
 - rreg 6-2
 - swap 6-2
 - sync 6-2
 - tmax 6-3
 - unit 6-2
 - val 6-2, 6-3
- slprog
 - application 6-14
 - file 6-14
 - help 6-14
 - proceed 6-14
 - unit 6-14
- slreg
 - Offset 6-12
 - application 6-11
 - count 6-11, 6-12
 - data 6-11, 6-12
 - display online help 6-11
 - mask 6-11, 6-12
 - numbers 6-12
 - offset 6-11, 6-12
 - online help 6-11, 6-12
 - unit 6-11
- sltp
 - data 6-5
 - data print 6-5
 - display online help 6-5
 - graphing 6-5
 - mode 6-5
 - numTasks 6-5
 - numTimes 6-5
 - packet size 6-5
 - relentless 6-5
 - show additional help 6-5
 - SYNC 6-5
 - unit 6-5
- sltp application 6-5, 6-6
- sltp online help 6-5, 6-6
- slvex 6-10, 6-11
 - application 6-10, 6-11
 - CRC check 6-8
 - data pattern scheme 6-8
 - data verify 6-8
 - display online help 6-8
 - flow control 6-8
 - loop configuration 6-8
 - min packet size 6-8
 - mode 6-8
 - online help 6-8

- packet variation 6-8
 - print 6-10
 - runtime help 6-9
 - runtime output 6-9
 - runtime statistics 6-10
 - SYNC 6-8
 - timeout base 6-8
 - time-outs 6-11
 - unit 6-8
 - slvex application 6-8
 - software layers 2-5
 - status 4-4, 4-11, 5-6
 - adapter 4-5
 - card and driver 5-6
 - driver 6-4
 - structure 4-11
 - structure
 - fxsl_configstruct 4-3, 4-4, 5-2
 - fxsl_statusstruct 4-4, 5-6
 - structures 1-1, 4-3
 - synchronization 5-10
 - system memory 6-5
- T**
- terminator 4-3, 6-3
 - receive only 4-3, 6-3
 - thread 5-10, 6-6
 - definition 5-10
 - thread implementation 5-10
 - throughput 5-10, 6-5
 - application 6-5
 - increase 5-10
 - maximum 6-5
 - performance graph 6-5, 6-6
 - receive 6-10
 - sustained 5-10
 - test 6-6
 - transmit test 6-6
 - time-out 4-4, 4-5, 5-3, 5-5
 - maximum 6-3
 - operation 4-4, 4-8, 4-10
 - premature expiration 4-18
 - transaction 4-8, 4-10
 - value 4-5
 - topology 2-1
 - multiple-node ring 2-4
 - point-to-point 2-1, 2-2, 4-3
 - ring 2-1
 - transaction flags 4-8, 4-10
 - transmit operation 4-1
 - transmitter 2-3, 6-6, 6-9, 6-11
- U**
- unit 4-5, 4-6, 5-1, 5-2, 6-4, 6-9, 6-12, 6-14
 - unit number 4-4, 6-2, 6-5, 6-11, 6-14
 - UNIX 3-3, 4-3, 5-10, 6-13
 - unpredictable behavior 4-5
 - user buffer 4-8, 4-10
- V**
- variable 4-5, 4-17
 - pHandle 4-6
 - pointer 4-16
 - VME 4-2
 - VxWorks... 1-1, 1-2, 4-1, 5-10, 6-2, 6-3, 6-6, 6-9, 6-13
- W**
- Windows 1-1, 1-2, 3-4, 4-3, 6-13
 - Windows Explorer 3-4
 - Windows NT 3-4, 5-10