DX Software Development Kit 2.2.1L

Getting Started

User Guide

Licensing and Government Use

Any Exar software (“Licensed Programs”) based on Hifn Technology described in this document is furnished under a license and may be used and copied only in accordance with the terms of such license and with the inclusion of this copyright notice. Distribution of this document or any copies thereof and the ability to transfer title or ownership of this document’s contents are subject to the terms of such license.

Such Licensed Programs and their documentation may contain public open-source software that would be licensed under open-source licenses. Refer to the applicable product release notes for open-source licenses and proprietary notices. Use, duplication, disclosure, and acquisition by the U.S. Government of such Licensed Programs is subject to the terms and definitions of their applicable license.

Disclaimer

Exar reserves the right to make changes to its products, including the contents of this document, or to discontinue any product or service without notice. Exar advises its customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied upon is current. Every effort has been made to keep the information in this document current and accurate as of the date of this document’s publication or revision.

Limited Warranty

Exar warrants Products based on the Hifn Technology, including cards, against defects in materials and workmanship for a period of twelve (12) months from the delivery date. Exar's sole liability shall be limited to either, replacing, repairing or issuing credit, at its option, for the Product if it has been paid for. Exar will not be liable under this provision unless: (a) Exar is promptly notified in writing upon discovery of claimed defects by Buyer; (b) The claimed defective Product is returned to Exar, insurance and transportation charges prepaid, by Buyer; (c) The claimed defective Product is received within twelve (12) months from the delivery date; and (d) Exar's examination of the Product discloses to its satisfaction that the alleged defect was not caused by misuse, neglect, improper installation, repair, alteration, accident or other hazard. THIS WARRANTY DOES NOT COVER PRODUCT DAMAGE WHICH RESULTS FROM ACCIDENT, MISUSE, ABUSE, IMPROPER LINE VOLTAGE, FIRE, FLOOD, LIGHTNING OR OTHER ACTS OF GOD OR DAMAGE RESULTING FROM ANY MODIFICATIONS, REPAIRS OR ALTERATIONS PERFORMED OTHER THAN BY EXAR OR EXAR'S AUTHORIZED AGENT OR RESULTING FROM FAILURE TO STRICTLY COMPLY WITH EXAR'S WRITTEN OPERATING AND MAINTENANCE INSTRUCTIONS. BUYER ACKNOWLEDGES THAT THE PRODUCT ARE HIGHLY SENSITIVE ELECTRONIC PRODUCT REQUIRING SPECIAL HANDLING AND THAT THIS WARRANTY DOES NOT APPLY TO IMPROPERLY HANDLED PRODUCT. PRODUCT MANUFACTURED TO MEET BUYER'S SPECIFIC PERFORMANCE SPECIFICATIONS ACCEPTED BY EXAR ARE WARRANTED ONLY TO PERFORM IN CONFORMITY WITH SUCH SPECIFICATIONS, AND ARE WARRANTED ONLY AGAINST DEFECTS NOT RELATED TO SUCH SPECIFICATIONS IN ACCORDANCE WITH THE TERMS AND CONDITIONS SET FORTH HEREIN ABOVE.

Life Support Policy

Exar's Product are not authorized for use as critical components in life support devices or systems. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury or death to human life. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Buyer agrees to indemnify, defend and hold Exar harmless for any cost, loss, liability, or expense (including without limitation attorneys' fees and other costs of litigation or threatened litigation) arising out of violation of the above prohibition by Buyer or any person or entity receiving Exar's Product through Buyer.

Patent Infringement - Indemnification

Exar agrees, at its own expense, to defend Buyer from and against any claim, suit or proceeding, and to pay all judgments and costs finally awarded against Buyer by reason of claim, suit or proceeding insofar as it is based upon an allegation that the Product as furnished by Exar infringes any United States letter patent, provided that Exar is notified promptly of such claim in writing and is given authority and full and proper information and assistance (at Exar's expense) for defense of same. In case such Product are finally constituted an infringement and the use of Product is enjoined, Exar shall at its sole discretion and at its own expense: (1) procure for Buyer the right to continue using the Product; (2) replace or modify the same so that it becomes non-infringing; or (3) remove such Product and grant Buyer a credit for the depreciated value of the same.

Buyer shall have the right to employ separate counsel in any claim, suit or proceeding and to participate in the defense thereof, but the fees and expenses of Buyer's counsel shall not be borne by Exar unless: (1) Exar specifically so agrees; or (2) Exar, after written request and without cause, does not assume such defense. Exar shall not be liable to indemnify Buyer for any settlement effected without Exar's written consent, unless Exar failed, after notice and without cause, to defend such claim, suit or proceeding.

The indemnification shall not apply and Buyer shall indemnify Exar and hold it harmless from all liability or expense (including costs of suit and attorney's fees) if the infringement arises from, or is based upon Exar's

Motorola

The use of this product in stateful compression protocols (for example, PPP or multi-history applications) with certain configurations may require a license from Motorola. In such cases, a license agreement for the right to use Motorola patents (US05,245,614, US05,130,993) may be obtained directly from Motorola.

Patents

May include one or more of the following United States patents: 4,930,142; 4,996,690; 4,701,745; 5,003,307; 5,016,009; 5,126,739; 5,146,221; 5,414,425; 5,414,850; 5,463,390; 5,506,580; 5,532,694; 6,320,846; 6,816,459; 6,651,099; 6,665,725; 6,771,646; 6,789,116; 6,954,789; 6,839,751; 7,299,282; 7,260,558. Other patents pending.

Trademarks

Exporting

This product may only be exported from the United States in accordance with applicable Export Administration Regulations. Diversion contrary to United States laws is prohibited.

Exar Confidential

If you have signed a Exar Confidential Disclosure Agreement that includes this document as part of its subject matter, please use this document in accordance with the terms of the agreement. If not, please destroy the document.

Table of Contents

Preface

About This Document

Welcome to Exar’s DX Software Development Kit (SDK) Getting Started User Guide for Exar’s DX2040 Compression and Security Acceleration card and XR9240 Compression and Security Coprocessor. This document gives instructions on how to install and use the DX SDK version 2.2.1L.

Note that the SDK software refers to the XR9240 device as “92xx” to accommodate future variations of the device.

Audience

This document is intended for integrators and application developers responsible for and familiar with software and hardware architecture of a target system.

Prerequisites

Before proceeding, you should generally understand:

- Compression algorithms such as eLZS, gzip, zlib, Deflate
- Encryption algorithms such as Advanced Encryption Standard (AES), Triple Data Encryption Standard (3DES) and their modes of operation
- Cryptographic hash functions
- Public Key concepts
- Software and hardware of the target system
- C and C++ Programming Language

Document Organization

This document is organized as follows:

Chapter 1, “Introduction” provides an overview of the DX SDK.

Chapter 2, “Installing the DX Card” describes how to install Exar’s DX card into the user’s system.

Chapter 3, “DX SDK Directory Structure” defines the directory structure of the DX SDK and describes their contents.

Chapter 4, “Building the DX SDK” provides step-by-step instructions on how to compile the DX SDK package.

Chapter 5, “Linking to the DX SDK APIs” lists the required files needed to link user mode or kernel mode applications to the DX SDK and driver.

Chapter 6, “Installing the Driver” provides step-by-step instructions on how to install the DX SDK driver.

Chapter 7, “Running the Applications” gives instructions on how to invoke Exar’s DX SDK applications.

Appendix A lists the driver configuration file parameters.

Appendix B defines the demo application configuration file, and a sample output file.

Appendix C describes the sdemo application, its encode and decode configuration files, key file, IVAAD file, and provides sample output files.

Related Documents

The following documents can be used to supplement this document.

DX SDK 2.2.1L Release Notes, RLN-0009

DX SDK User Guide, USR-0039

Raw Acceleration Application Programming Interface 2.2 Reference Guide, USR-0040

DX SDK 2.1.0L Raw Acceleration API Performance Application Note, APN-0006

XR9240 Compression and Security Coprocessor Data Sheet, DAT-0001

XR9240 Hardware Design User Guide, USR-0032

DX2040 Compression and Security Acceleration Card Data Sheet, DAT-0009

DX2040 Compression and Security Acceleration Card & XR9240 Compression and Security Coprocessor Errata, ERR-0005

Customer Support

For technical support about this product, please contact your local Exar sales office, representative, or distributor.

For general information about Exar and Exar products refer to: www.exar.com

Abbreviations

Term	Definition
3DES	Triple DES
AAD	Additional Authenticated Data
AES	Advanced Encryption Standard
API	Application Programming Interface
CBC	Cipher Block Chaining encryption mode
CTR	Counter encryption mode
DES	Data Encryption Standard
DIF	Data Integrity Field
ECB	Electronic Codebook encryption mode
ECPK	Elliptical Curve Public Key
eLZS	Enhanced Lempel-Ziv-Stac Compression
GCM	Galois Counter Mode
HIV	Hash Initialization Vector
HMAC	Hash Message Authentication Code
IV	Initial Vector
LZS	Lempel-Ziv-Stac Compression
SDK	Software Development Kit
SHA	Secure Hash Algorithm
XTS	XEX-based Tweaked CodeBook mode (TCB) with CipherText Stealing (CTS), or XEX-TCB-CTS

1 Introduction

Welcome to the DX Software Development Kit (SDK) Getting Started Guide for Release version 2.2.1L. This guide is intended to familiarize you with the components of the DX SDK.

- Application Programming Interface (API) enables customers to quickly begin product development with the Exar XR9240.
- Driver software provides the basic operation sets for the user application to access the XR9240.
- demo application demonstrates the Exar XR9240 functionality and measures the XR9240 performance for various modes, transforms, and block sizes.
- sdemo application is a simplified version of the demo application that can be used to easily demonstrate the functionality of the hardware.
- example gives users sample reference code demonstrating how to use the APIs.
- Diagnostic tools allow the user to monitor the status and performance of the hardware.

The DX SDK is designed to support the following Exar hardware products:

- XR9240 processor
- DX2040 card

This guide will walk you through the process of installing and bringing up the DX SDK and hardware platform. In this guide, you will:

- Learn about the structure and components that make up the software package
- Properly connect the DX card to a host
- Compile and install the driver
- Run the applications to verify the operations provided by the DX SDK

After reading this guide, you will be familiar with the basic features and operation of both the software and the hardware. You will be ready to begin developing your own custom DX application(s). Hardware designers will become familiar with the DX card layout, memory requirements, interfaces, etc. Software developers will become familiar with the API that is used to configure, communicate with, and control the DX card.

Before proceeding, make sure you have the following items:

DX Card, DX SDK, and Accessories

- Access to DX SDK 2.2.1L
- Exar DX card
- An available 12V PCIe slot to power the DX card
- Operating system - one of:

Red Hat Enterprise Linux 6.0 (Kernel 2.6.32-71.el6 for x86 64-bit)
Red Hat Enterprise Linux 6.2 (Kernel 2.6.32-220.el6 for x86 64-bit)
Red Hat Enterprise Linux 6.3 (Kernel 2.6.32-279.el6 for x86 64bit)
Red Hat Enterprise Linux 6.4 (Kernel 2.6.32-358.el6 for x86 64-bit)
CentOs release 6.1 (Kernel 2.6.32-131.0.15.el6 for x86 64bit)
CentOs release 6.2 (Kernel 2.6.32-220.el6 for x86 64bit)
CentOs release 6.3 (Kernel 2.6.32-279.el6 for x86 64bit)
CentOS release 6.4 (Kernel 2.6.32-358.el6 for x86_64)
CentOS release 7.0 (Kernel 3.10 for x86_64)
SUSE Linux Enterprise Server 10 SP3 (Kernel 2.6.16.60-0.54.5-smp for x86 64-bit)
SUSE Linux Enterprise Server 11 SP1 (Kernel 2.6.32.36-0.5-default for x86 64-bit)
SUSE Linux Enterprise Server 11 SP2 (Kernel version 3.0.10 for x86 64-bit)
Fedora 19 (Kernel version 3.9.4 for x86 64-bit)
Ubuntu 14.04 (Kernel version 3.13 for x86 64-bit)

- GNU make, GNU gcc, GNU libc

Documentation

Refer to the section below for a complete list of the DX SDK documentation.

1.1 Documentation Overview

This section provides an index of the available documentation, a description of individual document contents and how to use the document set.

Exar documentation identifiers such as DAT-0001-A01 include the following information:

"DAT": document type in the first two letters of the identifier,

DAT	=	Data sheet
APN	=	Application Note
USR	=	User Guide
RLN	=	Release Note

"0001": four numbers that indicate the document number

"-A01": the document release number. The first alpha character indicates the major document release version; the second two integers indicate the minor document release number. Initial revisions start at A01 and increment.

1.1.1 Where to find the documentation

All released Exar documentation is available on Exar's Extranet. An account can be requested from the main page of the Exar web site http://www.exar.com. Click on the Extranet Login button to bring up the login page.

1.1.2 How to use the documentation

Hardware designers should reference the *XR9240 Data Sheet*, DAT-0001, the *DX2040 Compression and Security Acceleration Card Data Sheet*, DAT-0009. The *DX2040 Compression and Security Acceleration Card & XR9240 Compression and Security Coprocessor Errata,* ERR-0005, contains the most current information about defects in the silicon that impact the use of specific features. The *XR9240 Hardware Design User Guide,* USR-0032, contains detailed recommendations for schematic design and PCB layout for card designers.

Software designers should reference the *DX Software Development Kit User Guide,* USR-0039, for an overview of the SDK architecture, data structures, and system considerations. The SDK User Guide also gives an example of a typical data operation using the Raw Acceleration API, and an overview of the SDK application programs. Software designers will also need to be familiar with the Application Programming Guide USR-0040. Software designers should examine the hardware documents for information about the system capabilities and theory of operation.

System and software designers should reference the *DX Software Development Kit Getting Started User Guide,* USR-0038, for directions on how to compile and install the SDK and driver, and for directions on how to run the application programs. The *DX Software Development Kit Release Notes,* RLN-0009, should be read for deviations in usage and features to the SDK. The DX SDK Performance Application Notes give performance benchmark data for the DX SDK APIs running on all DX cards.

1.1.3 Hardware Documents

DAT-0001, *XR9240 Compression and Security Coprocessor Data Sheet*

The *XR9240 Compression and Security Coprocessor Data Sheet* should be used by hardware designers, system engineers and software engineers to understand the XR9240 coprocessor capabilities and theory of operation. The Data Sheet includes the complete PCIe and internal register sets as well as the physical and electrical specifications required to implement a hardware design with the XR9240 coprocessor. All designers will benefit from being familiar with the section covering the chip architecture and data structures. This document helps set the foundations that will be expanded upon in documents such as the *XR9240 Hardware Design User Guide,* USR-0032.

DAT-0009, *DX2040 Compression and Security Acceleration Card Data Sheet*

The *DX2040 Compression and Security Acceleration Card Data Sheet* should be used by hardware designers and system engineers to understand how the XR9240 coprocessor has been designed into a card. In addition to a product description, the Data Sheet includes a complete signal list for all external interfaces, the physical, thermal and electrical specifications, as well as the safety and EMI/EMC certifications. The document *DX20xx Compression and Security Acceleration Card & XR9240 Compression and Security Coprocessor Errata,* ERR-0005, should be referenced for any hardware deviations for the DX 2040 card.

USR-0032, *XR9240 Hardware Design User Guide*

The *XR9240 Hardware Design User Guide* contains detailed recommendations for schematic design and PCB layout. Contents include power and ground considerations, thermal requirements, PCIe interface considerations and termination requirements.

ERR-0005, *DX2040 Compression and Security Acceleration Card & XR9240 Compression and Security Coprocessor Errata*

The *DX2040 Compression and Security Acceleration Card & XR9240 Acceleration Processor Errata* lists the hardware errata for both the XR9240 processor and DX cards. Revisions are listed for each errata so that hardware designers may identify which errata are applicable.

1.1.4 Software Documents

USR-0039, *DX Software Development Kit User Guide*

The *DX Software Development Kit User Guide* provides designers with the overall SDK architecture and operation. This document describes the fundamental session structures for the Raw Acceleration API. Important system considerations such as memory requirements, application dependent modes of operation and performance considerations are also described in this document and should be understood before installing the hardware and software.

The *DX Software Development Kit User Guide* also describes the driver architecture and the tradeoffs for selecting the appropriate driver configuration file parameter settings.

The User Guide gives a step-by-step guide to a typical Raw Acceleration data operation. The SDK application programs, called *demo, sdemo* and *example*, may also be viewed by designers to gain insight into how to use the SDK APIs to perform data operations. The User Guide describes the general procedures of these application programs and the diagnostic tools.

USR-0038, *DX Software Development Kit Getting Started User Guide*

This document should be used as a reference when ready to install the hardware and software. The *Getting Started Guide* introduces the product documentation, describes how to install a DX card, and gives a brief overview of the SDK directory structure. Detailed instructions are given for compiling the SDK, linking to the SDK APIs, installing the driver, and running the application programs. The application programs can be used to verify the SDK installation and hardware performance.

Designers should have already read the *DX Software Development Kit User Guide* for general system guidelines before installing the hardware and SDK.

USR-0040, *Raw Acceleration Application Programming Interface Reference Guide*

The *Raw Acceleration Application Programming Interface Reference Guide* provides the software designer with the exact syntax and usage of the Raw Acceleration API. This document assumes that the designer is already familiar with the SDK and has read the *DX Software Development Kit User Guide*.

1.1.5 System Documents

RLN-0009, *DX Software Development Kit Release Notes*

The *DX Software Development Kit Release Notes* document contains release specific information about the SDK. Software engineers should always carefully read the release notes. Project managers should also inspect the release notes in order to assess the impact of any defects or limitations. The document covers late breaking information about the release not covered in other documents, such as new features, changes since the last release, and limitations of the current release.

APN-0006, *DX SDK Raw Acceleration API Performance Application Note*

The *DX SDK Performance Application Note* documents contains release specific performance data for the DX SDK APIs for all DX cards. This document also describes the factors that affect performance and the performance measurement procedure, and lists the exact platforms on which the performance tests were run.

2 Installing the DX Card

2.1 Hardware Overview

The DX SDK operates with the following Exar DX cards and their card specific drivers:

- DX2040

Your DX card should look similar to Figure 2-1 below. There may be slight differences between the card pictured and the card you receive (card color, socketed versus soldered components, etc.).

The DX2040 card may be powered from any available 12V PCIe slot. The DX 2040 card ssupports 8 lanes of full duplex transceivers which plug into a x8 or larger PCIe 3.0 complaint slot. All communication to the card is via the PCIe interface. Refer to the *DX2040 Compression and Security Acceleration Card Data Sheet*, DAT-0009, for more information about the DX2040 cards.

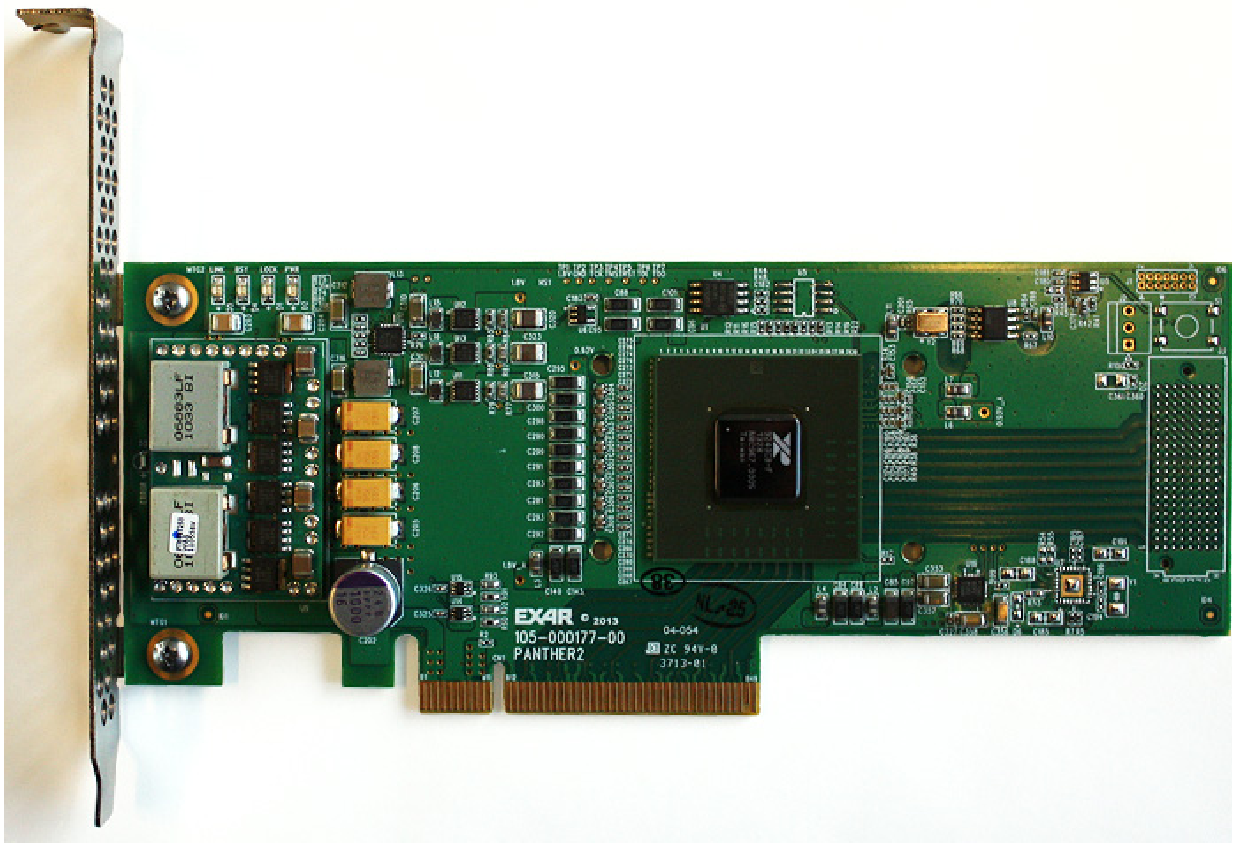

Figure 2-1. DX2040 Card

2.2 System Requirements

In order to install and test the DX card, the following system requirements must be met.

- An available half height or full height x8 or larger slot in a PCIe backplane For optimal performance, the DX2040 cards should be used with a PCIe 3.0 (10 GT/s) compatible interface.
- Host computer with Linux kernel version as specified in Chapter 1 installed

There are no other hardware requirements.

2.3 Installation Guidelines

2.3.1 ESD Protection

Warning

When handling the card, wear an ESD wrist-strap or foot-strap to dissipate any build-up of static charge.

The DX card is not designed to withstand ESD discharges. If the DX card is being moved from anywhere other than an ESD-protected area, cover the card with an anti-static material.

2.3.2 Installation Procedure

Note

It is highly recommended to read this document completely before installing the DX card.

To install the DX card, perform the following steps:

1. Power off the system and unplug the power cord.

Caution

Failure to unplug the power cord could damage the adapter or cause physical danger. Follow any specific instructions that came with your system about installing adapter cards.

2. Firmly insert the DX card into the selected PCIe slot.
3. If using a chassis, ensure that the card is fully seated, replace the slot cover, and secure the bracket to the system chassis.

4. Plug in the power cord and turn on the system power.

3 DX SDK Directory Structure

3.1 Unpacking the Software

The DX SDK package consists of the following:

- DX SDK system software
- DX SDK documentation

If this is your first shipment of the DX SDK, you will also receive the following items:

- Appropriate documentation for DX card
- Appropriate DX card (if purchased)

In the next chapter you will learn how to install the DX SDK on a Linux host. It is assumed throughout this guide that you are installing the DX SDK on an i386 or x86_64 platform.

There are a few things to note before we continue. The DX SDK can be run in either user space or kernel space. Additionally, this guide is designed for use specifically with the DX SDK 2.2.1L release. Other releases may possess different components and functionality.

3.2 Software Packages

There are two software packages used to compile the executable modules and applications for the Software Development Kit: (1) Public Package, and (2) Exar Package. Each package is described in more detail in the following sections. The user will compile a combination of these packages based on their license agreement with Exar.

3.2.1 Valid Package Combinations

The user may only utilize these packages in the following combinations:

1. Public Package only
2. Public Package + Exar Package

Note: The packages should be un-zipped in the order listed above.

3.2.2 Public Package

This package contains source files that are under a BSD or GPL agreement.

This package provides the fundamental Kernel mode routines and modules, including the driver module that supports the kernel mode Raw Acceleration APIs, support for the hardware algorithms, the userspace API library, as well as the eLZS compression and decompression modules.

The Public Package name includes "_PUBLIC" for easy identification. For example, the Public Package name is:

```
DX_SDK_v2.2.1L_PUBLIC_20150116.tar.gz
```

3.2.3 Exar Package

This package contains kernel mode and user mode demo and example application source files and the user mode diagnostic tool created by Exar.

This package is dependent on the Public package and it must be used with the Public package to generate the loadable Kernel mode application modules and the executable User mode binary files.

The Exar Package name includes "_EXAR_" for easy identification. For example, the Exar Package name is:

```
DX_SDK_v2.2.1L_EXAR_20150116.tar.gz
```

3.3 Package Directory Structures

This section describes the package directories for the Public and Exar packages.

3.3.1 Public Package Directory Structure

The Public Package directory contains the following sub-directories and files:

api
esf
h
dsd820x
dsd92xx
sai
swlib
userspace
driver.cfg.xml
generate_report
Load
Makefile
README.public
UnLoad

The DX SDK components are organized in separate directories, each with its own subdirectory structure. The following paragraphs will provide a high-level description of each component. The detailed software component descriptions and usage are beyond the scope of this document; refer to the User Guides and related software development collateral.

3.3.1.1 api Directory

This directory contains the source files that handle the calls to the Raw Acceleration API and transfer the API calls to the other internal processing modules.

3.3.1.2 esf Directory

This directory contains the source code for the Exar Service Framework (ESF). ESF is a chipset-independent core module that operates with the Device Specific Driver (DSD) modules to provide the algorithm acceleration. All chipset-independent code is partitioned in the esf directory, while all chipset-dependent code is partitioned in the DSD directories. Making the hardware dependent code as small and as simple as possible minimizes the design complexity to add support for a new chipset.

The ESF module’s basic functions are to manage the sessions and keys to facilitate the hardware acceleration and software simulation operations.

3.3.1.3 h Directory

This directory contains the header files needed to use the Raw Acceleration API and other internal routines.

3.3.1.4 dsd820x Directory

This directory contains the source code for the 820x Device Specific Driver (DSD). A single instance of the 820x DSD will manage all 820x devices installed in the system.

The 820x DSD provides the driver for all cards based on the 820x device, and provides a unified hardware interface to the Exar Service Framework (ESF).

3.3.1.5 dsd92xx Directory

This directory contains the source code for the XR9240 Device Specific Driver (DSD). A single instance of the XR9240 DSD will manage all XR9240 devices installed in the system.

The XR9240 DSD provides the driver for all cards based on the XR9240 device, and provides a unified hardware interface to the Exar Service Framework (ESF).

3.3.1.6 sai Directory

This directory contains the source code for the xml file parser, log and OSAL modules. There is a corresponding sub-directory for each of these modules.

3.3.1.7 swlib Directory

This directory contains the software library for the cipher, hash, and compression functions. There is a corresponding sub-directory for each of these three functions.

The swlib is implemented as a DSD, allowing it to have the same unified interface to the Exar Service Framework (ESF).

3.3.1.8 userspace Directory

This directory contains the user space C-modules for the Raw Acceleration API.

3.3.1.9 driver.cfg.xml file

This is a sample driver configuration file in xml format. The user should edit this file before installing the driver.

3.3.1.10 generate_report file

This script gathers the system information, device status, and SDK logs. The generated file may be sent to Exar technical support. The gathered information helps Exar to better understand a customer's issue. To create the report file, issue the command "make report".

3.3.1.11 Load file

This file is a sample script to load the driver. This script assumes there is a configuration file named *driver.cfg.xml* in the current directory. If another file name is used, the user should specify that driver configuration file name in the script file.

3.3.1.12 Makefile

This file is the master makefile to build the API source code, driver, and applications.

3.3.1.13 README.public file

This file is the README file for the PUBLIC package.

3.3.1.14 UnLoad file

This file is a sample script to unload the driver.

3.3.2 Exar Package Directory Structure

The Exar package directory contains the following sub-directories and files:

app

diag

demo.cfg.xml

Makefile

README.exar

sdemo.decode.cfg.xml

sdemo.drbg.xml

sdemo.encode.cfg.xml

sdemo.ivaad.xml

sdemo.key.xml

The DX SDK components are organized in separate directories, each with its own subdirectory structure. The following paragraphs will provide a high-level description of each component. The detailed software component descriptions and usage are beyond the scope of this document; refer to the User Guides and related software development collateral.

3.3.2.1 app Directory

The following applications can be found in the app directory: demo, example and sdemo.

/app/demo Subdirectory

This directory contains Linux source code for the demo application. The demo application includes a performance test.

/app/example Subdirectory

The example application demonstrates the usage of the APIs for operations such as compression/decompression, encryption/decryption and hash operations.

/app/sdemo Subdirectory

This directory contains Linux source code for the sdemo application. The sdemo application is a simple demo that reads the input data from a specified source file and stores the result in a specified destination file.

3.3.2.2 diag Directory

This directory contains the diagnostic tool implementation.

/man1 Subdirectory

This directory contains the man pages of dx_monitor, dx_diag and dx_status.

3.3.2.3 demo.cfg.xml

This is a sample demo application configuration file in xml format. The user should edit this file to generate the application specific configuration file for demo application.

3.3.2.4 Makefile

This file is the master makefile to build the API source code, driver, and applications.

3.3.2.5 README.exar file

This file is the README file for the EXAR package.

3.3.2.6 sdemo.decode.cfg.xml

This is a sample sdemo application decode configuration file in xml format. The user should edit this file to generate the application specific decode configuration file for the sdemo application.

3.3.2.7 sdemo.drbg.xml

This is a sample sdemo application drbg configuration file in xml format. The user should edit this file to generate the application specific drbg configuration file for the sdemo application.

3.3.2.8 sdemo.encode.cfg.xml

This is a sample sdemo application encode configuration file in xml format. The user should edit this file to generate the application specific encode configuration file for the sdemo application.

3.3.2.9 sdemo.ivaad.xml

This is a sample sdemo application AAD and IV file in xml format. The user should edit this file to generate the application specific AAD and IV data for the sdemo application.

3.3.2.10 sdemo.key.xml

This is a sample sdemo application key file in xml format. The user should edit this file to generate the application specific key file for the sdemo application.

4 Building the DX SDK

The DX SDK 2.2.1L can operate with the XR9240 devices or the DX cards. This section will describe how to build the DX SDK.

Note that this document specifically covers the DX SDK 2.2.1L release code. Other releases may deviate from these instructions. Existing user applications may require modification to operate with other DX SDK revisions.

Before compiling the DX SDK, it is assumed that the Exar DX card is properly connected to the Linux host system via the PCIe interface. Refer to Chapter 2, "Installing the DX Card" for instructions.

The basic compilation steps include:

1. Understand the DX SDK directory structure. Refer to Chapter 3, "DX SDK Directory Structure".
2. Copy the files in the packages you require from the Exar Extranet site or FTP distribution to your working directory.
3. Build the SDK package.

4.1 Copy DX SDK Files to Working Directory

Step 1 Install the Public Package

The first step to installing the DX SDK on your system is to copy all the files from the Public Package to a working directory, such as /home/dre_sw/dev.

Note: This step is required for all package combinations.

Untar the main Public package source code file with the Linux tar command:

```
tar xzvf DX_SDK_v2.2.1L_PUBLIC_20150116.tar.gz
```

This step will extract all the files from the Public package into the current working directory.

Step 2 Install the Exar Package

Note: This step is only required for package combinations that include the EXAR package.

Copy all the files from the Exar Package to the same working directory used in the first step, such as /home/dre_sw/dev.

Untar the main source code file with the Linux tar command:

```
tar xzvf DX_SDK_v2.2.1L_EXAR_20150116.tar.gz
```

After executing the tar command, there may be prompt to replace existing files. If so, respond "yes".

This step will extract supplemental directories and files from the Exar package into the current working directory.

4.2 Makefile Options

Issue "make help" to view the available targets and Makefile options.

For the Public package, the help response is similar to:

```
Supported Makefile Targets:
    ko:        build the kernel module dre_drv.ko
    sdk:       build the user space libraries libpan.a and libpan.so
    elzs:      build the user space libraries libelzs.a and libelzs.so
    all:       build all above targets, this is the "default" target
    install:   install driver files to enable udev, install diag tool
    uninstall: remove driver files and disable the udev, uninstall diag tool
    report:    generate a tar file that includes important debug information
    clean:     clean the files generated above
Supported Flags:
    1. BUILD_ELZS_LIB: generate library "libelzs", default is enabled
       "make BUILD_ELZS_LIB=0" builds all targets except libelzs
    2. DRE_820X_SUPPORT: manually specify whether the built driver supports 820x
       device
      "make DRE_820X_SUPPORT=0" forces the built driver not to support the 820x device
       "make DRE_820X_SUPPORT=1" forces the built driver to support the 820x device
    3. DRE_92XX_SUPPORT: manually specify whether the built driver supports 92xx
       device
      "make DRE_92XX_SUPPORT=0" forces the built driver not to support the 92xx device
      "make DRE_92XX_SUPPORT=1" forces the built driver to support the 92xx device
```

For the Exar package, the help response is similar to:

```
Supported Makefile Targets:
    ko:         build the kernel modules like dre_drv.ko, demo.ko, etc
    sdk:        build the user space libraries libpan.a and libpan.so
    elzs:       build the user space libraries libelzs.a and libelzs.so
    example:    build the user space examples pp_example, pk_example
    demo:       build the user space tools demo and sdemo
    diag:       build the user space tools diag
    all:        build all above targets, this is the "default" target
    install:    install driver files to enable udev, install diag tool
    uninstall: remove driver files and disable the udev, uninstall diag tool
    report:     generate a tar file that includes important debug information
    clean:      clean the files generated above
Supported Flags:
    1. BUILD_ELZS_LIB: generate library "libelzs", default is enabled
       "make BUILD_ELZS_LIB=0" builds all targets except libelzs
    2. BUILD_PP_NEG_TEST: generate negative tests in pp_example, default is enabled
       "make BUILD_PP_NEG_TEST=0" will not generate negative test cases in pp_example
    3. BUILD_HW_FPGA_TEST: generate HW FPGA test in pp_example, default is disabled
       "make BUILD_HW_FPGA_TEST=1" will generate HW FPGA test case in pp_example
        Note you must not enable it if the card has no FPGA, otherwise it will hang
    4. BUILD_SW_FPGA_TEST: generate SW FPGA test in pp_example, default is disabled
       "make BUILD_SW_FPGA_TEST=1" will generate SW FPGA test case in pp_example
    5. DRE_820X_SUPPORT: manually specify whether the built driver supports 820x
       device
      "make DRE_820X_SUPPORT=0" forces the built driver not to support the 820x device
       "make DRE_820X_SUPPORT=1" forces the built driver to support the 820x device
    6. DRE_92XX_SUPPORT: manually specify whether the built driver supports 92xx
       device
      "make DRE_92XX_SUPPORT=0" forces the built driver not to support the 92xx device
      "make DRE_92XX_SUPPORT=1" forces the built driver to support the 92xx device
```

4.3 Support for Exar Hardware

The Makefile will probe for installed Exar hardware and build the appropriate device specific drivers (DSDs).

Note

The DX SDK does not support a mixture of 820x and XR9240 hardware on the same node. Nodes must be either populated with all 820x based hardware or all XR9240 based hardware.

The DX SDK must be re-built if the installed Exar hardware changes from 820x devices or DX 18xx cards to XR9240 device or DX2040 cards and was built with only one of `DRE_820X_SUPPORT=1` or `DRE_92XX_SUPPORT=1` set.

To build the SDK to support both the legacy 820x hardware and the XR9240 hardware, use the make command "`make DRE_820X_SUPPORT=1 DRE_92XX_SUPPORT=1`". Doing so will cause the SDK to install the device specific driver for both the 820x and XR9240 devices. If a node contains 820x based hardware that is later upgraded to XR9240 based hardware, the SDK will then automatically recognize the XR9240 hardware after a reboot.

4.4 Build the SDK

Once the files have been copied locally and untarred the DX SDK can be built.

This procedure will build all the directories for the entire DX SDK package. It will fill the created directories with the executable files for the APIs, driver, and applications.

When compiled, the DX SDK will generate both static and dynamic libraries, however all applications provided by Exar (demo, sdemo, example, etc) will be linked statically. Customer applications may link to either the static or dynamic library based on their requirements.

Note: for DX SDK 2.0.0L and later, the default Linux kernel path in the Makefile is:

```
/lib/modules/$(shell uname -r)/build
```

and it is assumed that there is a file named *.config* in the kernel path. For example, if your kernel version shown by the command "uname -r" is 2.6.32-220.el6.x86_64, it is assumed the kernel path is /lib/modules/2.6.32-220.el6.x86_64/build. If /lib/modules/2.6.32-220.el6.x86_64/build does not exist, the user should create a soft link to this directory. For example, if the kernel path is /usr/src/kernels/2.6.32-220.el6.x86_64, the command to create a soft link would be:

```
#ln -s /usr/src/kernels/2.6.32-220.el6.x86_64 /lib/modules/2.6.32-220.el6.x86_64/
build
```

The user may also set the variable "KERNELDIR" in Makefile to point to the source directory of the kernel on which the built SDK/driver/application would be run.

If the *.config* file does not exist in kernel path, the user should build one. For example (in /lib/modules/2.6.32-220.el6.x86_64/build/):

```
#ls -a .config
#make oldconfig
#ls -a .config
.config
```

To build the SDK, follow the steps outlined below:

1. Make sure the proper version of `make` is on your search path.
2. Change into the top-level directory where you extracted the files, for example /home/dre_sw/dev.
3. Run `make`

   This will generate the standard make output. There should be no errors or warnings. After this step, all the project object and executable files will be generated under the current directory, /home/dre_sw/dev.

4.4.1 Public Package Build Results

The following files will be present after a successful build of the Public package:

- dre_drv.ko - Linux driver
- libpan.a - Raw API Linux user mode static library
- libpan.so: Linux symbolic link to libpan.so.2.2.1
- libpan.so.2: Linux symbolic link to libpan.so.2.2.1
- libpan.so.2.0.0: Linux SDK shared library
- libelzs.a - Linux eLZS user mode static library
- libelzs.so - Linux eLZS symbolic link to libelzs.so.1.20.0
- libelzs.so.1 - Linux eLZS symbolic link to libelzs.so.1.20.0
- libelzs.so.1.20.0 - Linux eLZS shared library

4.4.2 Public + Exar Package Build Results

The following files will be present after a successful build of the Public + Exar package:

- dre_drv.ko - Linux driver
- libpan.a - Raw API Linux user mode static library.
- libpan.so: Linux symbolic link to libpan.so.2.2.1
- libpan.so.2: Linux symbolic link to libpan.so.2.2.1
- libpan.so.2.0.0: Linux SDK shared library
- demo.ko - Linux kernel mode demo application
- demo - Linux user mode demo application
- pk_example.ko, pp_example.ko - Linux kernel mode examples
- pk_example, pp_example - Linux user mode examples
- libelzs.a - Linux eLZS user mode static library
- libelzs.so - Linux eLZS symbolic link to libelzs.so.1.20.0
- libelzs.so.1 - Linux eLZS symbolic link to libelzs.so.1.20.0
- libelzs.so.1.20.0 - Linux eLZS shared library
- sdemo - Linux user mode sdemo application
- sdemo.ko: Linux kernel mode sdemo application
- dx - Linux user mode diagnostic application
- dx_monitor: a copy of dx
- dx_status - Script that reports the PCIe capabilities of the installed DX cards

- dxperf - Wrapper for the demo application that runs the performance test

5 Linking to the DX SDK APIs

This chapter describes the files needed to link user mode or other kernel mode applications to the DX SDK and driver.

5.1 Exar Application Programming Interfaces

To use Exar's Raw Acceleration APIs, include the file *dre_api.h* in your application source code and add the /h directory to the list of directories to be searched for header files by using the gcc "-I" options.

5.1.1 User mode applications

To use the Raw Acceleration APIs in user mode applications, the following files are required:

- All header files in the /h directory
- Raw Acceleration user mode library file *libpan.a* or *libpan.so.*
- User application source code
- *pthread* library

The user application code should be linked to both the *libpan.a* and the *pthread* libraries. All user mode applications must define the macros _REENTRANT and LINUX in the makefile.

5.1.2 Other Linux kernel modules

To use the Raw Acceleration APIs on other Linux kernel modules, the following files are required:

- All header files in the /h directory
- Kernel driver file *dre_drv.ko*
- User application source code

The user should first install the kernel driver file *dre_drv.ko* and then build their own kernel modules. The application must define the macro LINUX in its makefile before building the modules.

6 Installing the Driver

This chapter describes how to install the driver module after the SDK has been built.

Note that the driver can be loaded without Exar XR9240 hardware in the system. In this mode, the user should modify the driver configuration file to enable the internal software library for data processing.

6.1 Editing the Driver Configuration File (Optional)

Before installing the driver, the user should review the parameters in the driver configuration file, *driver.cfg.xml*, to determine if the default settings can be applied to their system.

Note that the configuration settings are global and will be applied to all the probed Exar DX cards or XR9240 devices. The user should ensure the configuration parameters are applicable for all cards/devices.

A sample of the driver configuration file is provided in Appendix A.

6.2 Load Script Options

For many servers, all interrupts are handled by CPU0 which causes CPU0 to experience a higher load than the other CPUs and results in poor system level performance. To address this issue, an “irq_balance” option was added to the Load script. If specified using the command "sh Load irq_balance", interrupts for each of the XR9240 device’s 16 rings will be binded to different CPU cores, thereby evenly distributing the SDK interrupts across all available CPUs.

6.3 Installing the Driver

Please note that the scripts in this section are case sensitive.

To install the Linux Driver

Step 1 Configure the Platform

The Linux driver requires a large amount of memory to load (see the *DX SDK User Guide*, USR-0039, for more information). For servers that do not satisfy the memory requirement, the configuration settings for cmds_per_ring and data_desc_per_cmd in the *driver.cfg* file should be reduced.

For the preliminary version of the DX2040 card or the first revision of the XR9240 device, the no_snoop feature must be disabled on most PCIe Gen3 platforms before loading the driver. For Intel Sandy Bridge E5 16xx/26xx/46xx platforms, the DX SDK will automatically disable no_snoop on the preliminary products.

No special procedure is required for the production release of the XR9240 device and DX2040 card.

If the driver load fails, please contact Exar Technical Support.

Step 2 **Run the Load script**

Note: Running the Load script requires root privilege.

```
sh Load
```

Step 3 **Check that the driver was successfully loaded**

```
lsmod
```

After the driver has been successfully loaded, running lsmod should reveal the module *dre_drv*.

Step 4 **Special instruction for SUSE Linux Enterprise Server 11**

For SLES 11, add 'allow_unsupported_modules 1' to /etc/modprobe.d/unsupported-modules to automate loading the driver after a reboot.

To uninstall the Linux Driver

Step 1 **Run the UnLoad script**

Note: Running the UnLoad script requires root privilege.

```
sh UnLoad
```

Step 2 **Check that the driver was successfully unloaded**

```
lsmod
```

After the driver has been successfully unloaded, running lsmod should reveal that the module *dre_drv* has been removed.

7 Running the Applications

The DX SDK contains applications that can be used to verify the software installation, hardware functionality and system performance. The code for these applications can also be used to demonstrate how to write the user application software.

The DX SDK applications are:

- sdemo application
- demo application
- example application
- dx diagnostic tools

As a Getting Started Guide, this document only provides instructions on how install and run these applications. Please refer to the *DX SDK User Guide*, USR-0039, for additional information on these applications. Note that there are additional configuration steps required to use the system beyond simply running the test applications.

Please note, before running the applications, the driver must already be installed.

Also note that the output messages from the Kernel mode example application are dumped to /var/log/messages as well as to a "text-mode" terminal such as system console. However, if you execute the command in a "terminal" window such as those created when the user is running in graphical mode (i.e. running X) or a remote terminal session via programs like PuTTY, you will NOT see the messages. In order to see the output messages from the Kernel mode example application, you will need to switch to a text mode console first (i.e, entering Ctrl-Alt-F1), which will switch the display to the first text mode terminal or system console.

7.1 sdemo Application

The sdemo application provides a simple functional demonstration of the Exar XR9240 and DX cards, including compression, encryption, and Hash/MAC for both encode and decode. The sdemo application has the following configuration files:

- a configuration file to specify operations and input/output files (required)
- a key file for algorithms that require a key, e.g. encryption or MAC
- an IVAAD file for algorithms that require IV or AAD data
- a drbg file for the DRBG known answer test

7.1.1 Editing the sdemo Configuration File

Exar provides two sample sdemo configuration files, *sdemo.encode.cfg.xml* and *sdemo.decode.cfg.xml*, for customer reference. These configuration files should be reviewed and edited before running the sdemo application. For the detailed syntax of the sdemo configuration files, please refer to the *DX SDK User Guide*, USR-0039. Appendix C of this document provides samples of the sdemo configuration files.

The default parameters in the sdemo encode configuration file will run an DEFLATE compression operation using the README.public file as the source input data. The default parameters in the sdemo decode configuration file will decompress the compressed data file README.public.encode, created by running:

```
./sdemo sdemo.encode.cfg.xml
```

Amend the sdemo configuration files to run a different algorithm or to use a different source data file.Editing the sdemo Key File (Optional)

Exar provides a sample sdemo key file, *sdemo.key.xml*, for customer reference. This key file should be reviewed and edited before running the sdemo application if the selected algorithms require a key. All encryption and MAC algorithms require a key. For the detailed syntax of the sdemo key file, please refer to the *DX SDK User Guide*, USR-0039.

7.1.2 Editing the sdemo IVAAD File (Optional)

Exar provides a sample IVAAD file, *sdemo.ivaad.xml,* for customer reference. This file should be reviewed and edited before running the sdemo application if the selected algorithms require an IV or AAD. All encryption algorithms except ARC4 require an IV; AES GCM requires an AAD.

7.1.3 Editing the drbg File (Optional)

Exar provides a sample drbg file, *sdemo.drbg.xml,* for testing the DRBG implementation. This file should be reviewed and edited before running the sdemo application if running this optional test.

For the standard, run-time DRBG implementation, the DX SDK retrieves random bytes from the XR9240 hardware RNG to use as the entropy source. This results in a DRBG output that is un-predictable and cannot be subject to a known answer test.

In order to verify the DRBG implementation, the sdemo application implements an interface to “specify” the entropy source as well as other input parameters required to create the DRBG.

The drbg file, *sdemo.drbg.xml,* is used to specify the DRBG input parameters and expected DRBG output. The sdemo application uses this data to perform a known answer test. The NIST DRBG specification includes several test vectors that may be used to verify a DRBG implementation.

7.1.4 Running the sdemo Application

Before running the sdemo application, the driver must have been successfully loaded and the configuration files appropriately configured. The data to be processed must have been loaded into the specified source file.

To run the sdemo application in user mode

Step 1 Display the command usage

```
./sdemo --help
```

The sdemo usage help page will be displayed.

```
./sdemo { file.cfg.xml [file.key.xml] [file.ivaad.xml] } | file.drbg.xml
file.cfg.xml:    The configuration file which specifies the src and dst data file
                 names and the algorithm. The configuration file name must end
                 with the file extension ".cfg.xml". Refer to sdemo.cfg.xml as an
                 example.
file.key.xml:    Only required if the algorithm specified in the configuraion
                 file requires a key. The key file name must end with the file
                 extension ".key.xml". Refer to sdemo.key.xml as an example.
file.ivaad.xml:  Only required if the algorithm specified in the configuraion
                 file requires an IV or AAD or both. The ivaad file name must end
                 with the file extension ".ivaad.xml". Refer to sdemo.ivaad.xml
                 as an example.
file.drbg.xml:   Only required for a drbg test. The drbg file name must end with
                 the file extension ".drbg.xml". Refer to sdemo.drbg.xml as an
                 example.
```

Step 2 Run the default DEFLATE compression transform

```
./sdemo sdemo.encode.cfg.xml
```

The sample *sdemo.encode.cfg* file included in the SDK package will perform DEFLATE compression on the file *README.public* and save the results in the destination file *README.public.encode*.

Step 3 Run the default DEFLATE decompression transform

```
./sdemo sdemo.decode.cfg.xml
```

The sample *sdemo.decode.cfg* file included in the SDK package will decompress the file *README.public.encode* (from Step 2 above) and save the result in the destination file *README.public.check*.

Step 4 Compare and verify the pair-wise result

```
diff README.public README.public.check
```

An error will be returned if the configuration is not correct.

To run the sdemo in Kernel mode

Step 1 Type:

```
insmod sdemo.ko { conf_name=file.cfg.xml [ key_name=file.key.xml]
[ivAad_name=file.ivaad.xml] | drbg_name=file.drbg.xml }
```

For example:

```
insmod sdemo.ko conf_name=sdemo.encode.cfg.xml
insmod sdemo.ko conf_name=sdemo.decode.cfg.xml
```

To exit the demo in Kernel mode

Step 1 **Type:**

```
rmmod sdemo
```

7.2 demo Application

The demo application measures the XR9240 hardware or internal software algorithm library performance using the Raw Acceleration APIs to process data.

There are two binary Linux demo applications: user mode and kernel mode. A log file called *demo.log* records the result and any errors that may occurred during the test. In user mode the results are also printed to standard output. A kernel mode log is saved in kernel mode.

The demo application is run from the command line using parameters defined in the configuration file. Configurable parameters include the number of threads to spawn, the operation to execute, and the test mode to be used. A log records the result and any errors that may have occurred during the test.

7.2.1 Editing the demo Configuration file

Exar provides a default demo configuration file, *demo.cfg.xml*, for first time users. This configuration file must be edited before running the demo application. For the detailed syntax of the demo configuration file, please refer to the *DX SDK User Guide*, USR-0039. Appendix B of this document provides a sample of the demo configuration file and its output.

The following parameters must be set for all test cases:

- test_raw_pp

  This option specifies whether to test a Raw Acceleration session.
- test_raw_pk

  This option specifies whether to test the Raw Acceleration Public Key API operations.
- test_raw_rng

  This option specifies whether to test the Raw Acceleration random number generator API operations.
- test_raw_drbg

  This option specifies whether to test the DRBG API operations.
- cmd

  This option specifies the number of commands to run for each thread.

- stop_sec

  This option specifies the run time in seconds.
- max_per_run

  This option specifies the maximum number of commands per thread.
- thread

  This option specifies the number of threads per session.

To test a Raw Acceleration session, the following parameters should also be set:

- async

  This option specifies whether the session is run in synchronous or asynchronous mode. Refer to the *DX SDK User Guide*, USR-0039, for a description of asynchronous and synchronous modes of operation.
- pkt_size

  This option specifies the packet size for each command.
- en_alg_conf

  This option specifies whether the algorithm for the session is configurable. If en_alg_conf is disabled, the demo application will iterate through all the combinations of algorithms for supported Raw Acceleration sessions, which may take the application a very long time to run.

The remaining configuration options are test mode dependent.

7.2.2 Running the Kernel Mode demo Application

Before running the kernel mode demo application, make sure that the driver has been successfully loaded and installed (see Section 6.3).

To run the demo in Kernel mode

Step 1 **Type:**

```
insmod demo.ko conf_name=<conf file name>
```

where:

conf_name is the name of the demo configuration file

To exit the demo in Kernel mode

Step 1 **Type:**

```
rmmod demo
```

Kernel mode demo Example 1: Compression operation

Step 1 Edit the demo configuration file *demo.cfg.xml*.

```
<demo_configuration>
    <test_raw_pp>1</test_raw_pp>
    <test_raw_pk>0</test_raw_pk>
    <test_raw_rng>0</test_raw_rng>
    <test_raw_drbg>0</test_raw_drbg>
    <cmd>100</cmd>
    <stop_sec>0</stop_sec>
    <max_per_run>100</max_per_run>
    <thread>1</thread>
    <async>0</async>
    <pkt_size>2048</pkt_size>
    <src_data_file></src_data_file>
    <en_alg_conf>1</en_alg_conf>
    <direction>1</direction>
    <enc_algo>NONE</enc_algo>
    <comp_algo>ELZS</comp_algo>
    <seg_hash_algo>NONE</seg_hash_algo>
</demo_configuration>
```

Step 2 To run the demo compression test, type:

```
insmod demo.ko conf_name=demo.cfg.xml
```

Kernel mode demo Example 2: Hash operation

Step 1 Edit the demo configuration file *demo.cfg.xml*.

```
<demo_configuration>
    <test_raw_pp>1</test_raw_pp>
    <test_raw_pk>0</test_raw_pk>
    <test_raw_rng>0</test_raw_rng>
    <test_raw_drbg>0</test_raw_drbg>
    <cmd>100</cmd>
    <stop_sec>0</stop_sec>
    <max_per_run>100</max_per_run>
    <thread>1</thread>
    <async>0</async>
    <pkt_size>2048</pkt_size>
    <src_data_file></src_data_file>
    <en_alg_conf>1</en_alg_conf>
    <direction>1</direction>
    <enc_algo>NONE</enc_algo>
    <comp_algo>NONE</comp_algo>
    <seg_hash_algo>HASH_SHA1</seg_hash_algo>
</demo_configuration>
```

Step 2 Run the demo hash test:

```
insmod demo.ko conf_name=demo.cfg.xml
```

Kernel mode demo Example 3: Encryption operation

Step 1 Edit the demo configuration file *demo.cfg.xml*.

```
<demo_configuration>
    <test_raw_pp>1</test_raw_pp>
    <test_raw_pk>0</test_raw_pk>
    <test_raw_rng>0</test_raw_rng>
    <test_raw_drbg>0</test_raw_drbg>
    <cmd>100</cmd>
    <stop_sec>0</stop_sec>
    <max_per_run>100</max_per_run>
    <thread>1</thread>
    <async>0</async>
    <pkt_size>2048</pkt_size>
    <src_data_file></src_data_file>
    <en_alg_conf>1</en_alg_conf>
    <direction>1</direction>
    <enc_algo>AES_CTR</enc_algo>
    <comp_algo>NONE</comp_algo>
    <seg_hash_algo>NONE</seg_hash_algo>
    <drbg_request_bits>1024</drbg_request_bits>
</demo_configuration>
```

Step 2 **Run the demo encryption test:**

```
insmod demo.ko conf_name=demo.cfg.xml
```

Kernel mode demo Example 4: Performance AES_CTR test

Step 1 **Edit the demo configuration file *demo.cfg.xml*.**

```
<demo_configuration>
    <test_raw_pp>1</test_raw_pp>
    <test_raw_pk>0</test_raw_pk>
    <test_raw_rng>0</test_raw_rng>
    <test_raw_drbg>0</test_raw_drbg>
    <cmd>100000</cmd>
    <stop_sec>0</stop_sec>
    <max_per_run>100</max_per_run>
    <thread>8</thread>
    <async>0</async>
    <pkt_size>2048</pkt_size>
    <src_data_file></src_data_file>
    <en_alg_conf>1</en_alg_conf>
    <direction>1</direction>
    <enc_algo>AES_CTR</enc_algo>
    <comp_algo>NONE</comp_algo>
    <seg_hash_algo>NONE</seg_hash_algo>
</demo_configuration>
```

Step 2 **Run the perf mode test:**

```
insmod demo.ko conf_name=demo.cfg.xml
```

7.2.3 Running the User Mode demo Application

To run the demo in User mode

Step 1 **Type:**

```
./demo <conf_name>
```

where:

conf_name is the name of the demo configuration file.

7.3 example Application

The example application provides a sample reference to demonstrate the usage of the APIs for specific operations.

There are four binary example Linux applications:

- User Mode Packet Processing example Application
- Kernel Mode Packet Processing example Application
- User Mode Public Key example Application
- Kernel Mode Public Key example Application

7.3.1 Packet Processing example Application

A packet processing example, *pp_example*, demonstrates how to use the Raw Acceleration session API in both user mode and kernel mode applications.

The Raw Acceleration API example sessions demonstrate:

1. HMAC-SHA1 operation. All commands are submitted in synchronous mode to a stateless Raw session.

   First, an encode command with input data to generate a MAC is compared against a known value. Second, a decode command uses the input data and MAC generated in the first step to verify the MAC. If negative testing is enabled (“make BUILD_P-P_NEG_TEST=0” to disable it), a third decode command with input data and a corrupt MAC is generated to verify that an error is returned.

2. eLZS Compression and Decompression. Commands are submitted in asynchronous mode to a stateless Raw session.

   First, a compression command compresses the input data. Second, a decompress command operates on the compressed data to generate decompressed data. The input data is compared against the decompressed data to verify that they are the same. If negative testing is enabled (“make BUILD_PP_NEG_TEST=0” to disable it), a third decompress command with corrupt compressed data is performed to verify that an error is returned.

3. AES-GCM Encode and Decode. Commands are submitted in asynchronous mode to a stateless Raw session.

   First, an encode command uses plaintext input to generate the cipher and authentication tags, which are then compared against known values. Second, a decode command uses the cipher and authentication tags to verify the tag and generate the decrypted plaintext, which is then compared against the original plaintext.

4. HMAC-SHA1 + eLZS operation. All commands are submitted in synchronous mode to a stateless Raw session.

   First, an encode command compresses the input data and generates a MAC. Second, a decode command uses the compressed data and MAC to decompress and verify the MAC.

5. Gzip + AES-CBC + HMAC-SHA1 Encode and Decode. Commands are submitted in asynchronous mode to a stateless Raw session.

   First, an encode command compresses, encrypts and authenticates the input data to generate cipher data and an authentication tag. Second, a decode command verifies the authentication, decryption and decompression and generates the decoded data, which is then compared against the original input data.

6. ESP Transport Encode. All commands are submitted in synchronous mode to a stateless Raw session.

   First, an encode command encrypts and authenticates the input data. Second, a decode command authenticates, decrypts and generates the decoded data, which is then compared against the original input data.

7. TLS Record Layer Encode and Decode. All commands are submitted in synchronous mode to a stateless Raw session.

   First, an encode command authenticates and encrypts the input data. Second, a decode command decrypts authenticates and generates the decoded data, which is then compared against the original input data.

8. Stateful Deflate Dynamic Compression and Decompression. All commands are submitted in synchronous mode to a stateful Raw session.

   First, the input data is split into three fragments and each fragment is compressed using a stateful command. The compressed data results are concatenated and decompressed using a single command. The decompressed data is compared against the original input data.

9. If the SDK was built with the Make option "BUILD_HW_FPGA_TEST=1", the pp_example will also include one FPGA hardware pairwise example. Both commands are submitted in synchronous mode to a stateless Raw session.

   a. Encode command: ILK_PRE Passthrough + XR9240 eLZS Compression + ILK_POST Disabled

      First, an encode command containing input data is passed to the FPGA via the ILK_PRE interface, and the FPGA passes the data back to the XR9240 unchanged. Second, the XR9240 compresses the data using eLZS and returns the compressed data back to the pp_example application. The ILK_POST function is skipped since it is disabled.

      Decode command: ILK_PRE Passthrough + XR9240 eLZS Decompression + ILK_POST Disabled

      First, a decode command containing the compressed data from the encode command is passed to FPGA via the ILK_PRE interface, and the FPGA passes the data back to the XR9240 unchanged. Second, the XR9240 decompresses the data using eLZS and returns the decompressed data to the pp_example application. The ILK_POST interface is skipped since it is disabled.

      The original input data is compared against the decompressed data to verify that they are the same.

10. If the SDK was built with the Make option “BUILD_SW_FPGA_TEST=1” the pp_example will also include three FPGA Software Model pairwise examples. The commands are submitted in synchronous mode to a stateless Raw session.

    a. Encode command: ILK_PRE Passthrough + swlib eLZS Compression + ILK_POST Disabled

        First, an encode command containing input data is passed to the FPGA Software Model, and the ILK_PRE function passes the data to swlib unchanged. Second, the swlib compresses the data using eLZS and returns the compressed data to the pp_example application. The ILK_POST function is skipped since it is disabled.

       Decode command: ILK_PRE passthrough + swlib eLZS Decompression + ILK_POST disabled

        First, a decode command containing the compressed data from the encode command is passed to the FPGA Software Model, and the FPGA ILK_PRE function passes the data to the swlib unchanged. Second, the swlib decompresses the data using eLZS and returns the decompressed data to the pp_example application. The ILK_POST function is skipped since it is disabled.

       The original input data is compared against the decompressed data to verify that they are the same.

    b. Encode command: ILK_PRE Changes swlib Compression + swlib eLZS->gzip Compression + ILK_POST Disabled

        An encode command containing input data is passed to the FPGA Software Model. This command configures the swlib to perform eLZS compression, but also configures ILK_PRE to force the swlib to perform gzip compression instead. First, the FPGA ILK_PRE function changes the TE command header so that the swlib will perform gzip compression. Second, the swlib compresses the data using gzip and returns the compressed data to the pp_example application. The ILK_POST function is skipped since it is disabled.

       Decode command: ILK_PRE Changes swlib Decompression + swlib eLZS->gzip Decompression + ILK_POST Disabled

        A decode command containing the compressed data from the encode command is passed to the FPGA Software Model. This command configures the swlib to perform eLZS decompression, but also configures ILK_PRE to force the swlib to perform gzip decompression. First, the FPGA ILK_PRE function changes the TE header so that the swlib will perform gzip decompression. Second, the swlib decompresses the data using gzip and returns the decompressed data to the pp_example application. The ILK_POST function is skipped since it is disabled.

       The input data is compared against the decompressed data to verify that they are the same.

    c. Encode command: ILK_PRE Disabled + swlib eLZS Compression + ILK_POST Pad & AES Encryption

        An encode command containing input data is passed to the FPGA Software Model. The ILK_PRE function is skipped since it is disabled. The swlib compresses the data using eLZS and passes the compressed data to ILK_POST. The ILK_POST function pads and then encrypts the data using AES128-CBC.

Decode command: ILK_PRE AES Decryption & Depad + swlib eLZS Decompression + ILK_POST Disabled

A decode command containing the encrypted data from the encode command is passed to the FPGA Software Model. The FPGA ILK_PRE function decrypts the data using AES128-CBC, depads the data, and then passes the data to the swlib. The swlib decompresses the data using eLZS and returns the decompressed data to the pp_example application. The ILK_POST function is skipped since it is disabled.

The input data is compared against the decompressed data to verify that they are the same.

7.3.1.1 Packet Processing User Mode example Application

The Linux user mode packet processing example demonstrates how to use the Raw Acceleration API session in a Linux user module application.

To run the User mode Packet Processing example Application

Step 1 **Type:**

```
./pp_example
```

A detailed report of the results for each example may be found in the standard output. If all are successful, “PP_Example: [ALL OK]!” will be printed in the output and the application will return zero.

```
User Mode Example:
Session Type: Stateless Raw Session
Cmd Submission Type: Synchronous
Operation: HMAC_SHA1 KAT
Result [OK]

User Mode Example
Session Type: Stateless Raw Session
Cmd Submission Type: Asynchronous
Operation: eLZS PairWise
Result [OK]

User Mode Example
Session Type: Stateless Raw Session
Cmd Submission Type: Asynchronous
Operation:  AES_GCM KAT
Result [OK]

User Mode Example
Session Type: Stateless Raw Session
Cmd Submission Type: Synchronous
Operation:  HMAC-SHA1 + eLZS PairWise
```

```
*|-- head(16)--|-------- data -------|
*|----------------------  MAC -------|
*              |-------- eLZS -------|
Result [OK]

User Mode Example
Session Type: Stateless Raw Session
Cmd Submission Type: Asynchronous
Operation:  Gzip + AES-CBC + HMAC-SHA1 PairWise
*|-- head(16)--|------(raw)comp -|--pad--|
*              |--------ENC--------------|
*|---------------------MAC---------------|
Result [OK]

User Mode Example
Session Type: Stateless Raw Session
Cmd Submission Type: Synchronous
Operation: ESP Transport Encode KAT
ESP Enc Algorithm: 3DES_CBC
ESP Auth Algorithm: HMAC_MD5
Result: [OK]

User Mode Example
Session Type: Stateless Raw Session
Cmd Submission Type: Synchronous
Operation: TLS1.0 Record Layer Encode && Decode KAT
TLS Enc Algorithm: 3DES_CBC
TLS Auth Algorithm: HMAC_SHA1
Result: [OK]

User Mode Example
Session Type: Stateful Raw Session
Cmd Submission Type: Synchronous
Operation: Stateful Deflate Dynamic PairWise
Result [OK]
```

If the SDK was built with the Make option “BUILD_HW_FPGA_TEST=1”, the output below will also be displayed.

```
DX20xx FPGA example:

*************  Test 1: eLZS Compression/Decompression  *************

ILK PRE: pass-through ----> Swlib: eLZS compression ----> ILK POST:
disabled
Input(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Output(len:8):
0x00 0x60 0x7f 0xff 0xeb 0x00 0x00 0x00

ILK PRE: pass-through ----> Swlib: eLZS decompression ----> ILK POST:
disabled
Input(len:8):
0x00 0x60 0x7f 0xff 0xeb 0x00 0x00 0x00

Output(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

eLZS pairwise test passed
```

If the SDK was built with the Make option “BUILD_SW_FPGA_TEST=1”, the output below will also be displayed.

```
FPGA SW Model examples:

*************  Test 1: eLZS Compression/Decompression  *************

ILK PRE: Pass-through --> Swlib: eLZS compression --> ILK POST: Disabled
Input(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Output(len:8):
0x00 0x60 0x7f 0xff 0xeb 0x00 0x00 0x00

ILK PRE: Pass-through --> Swlib: eLZS decompression --> ILK POST: Disabled
Input(len:8):
0x00 0x60 0x7f 0xff 0xeb 0x00 0x00 0x00

Output(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
```

```
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

eLZS pairwise test passed

*************  Test 2: gzip Compression/Decompression  *************
ILK PRE: Force 92xx gzip compression --> Swlib: eLZS compression -->
ILK POST: Disabled
Input(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Output(len:24):
0x1f 0x8b 0x08 0x00 0x00 0x00 0x00 0x00
0x02 0x03 0x63 0x60 0xa0 0x0c 0x00 0x00
0x36 0x63 0x8d 0x75 0x40 0x00 0x00 0x00

ILK PRE: Force 92xx gzip decompression --> Swlib: eLZS decompression -->
ILK POST: Disabled
Input(len:24):
0x1f 0x8b 0x08 0x00 0x00 0x00 0x00 0x00
0x02 0x03 0x63 0x60 0xa0 0x0c 0x00 0x00
0x36 0x63 0x8d 0x75 0x40 0x00 0x00 0x00

Output(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Gzip pairwise test passed

**  Test 3: eLZS-AES Compression-Encryption/Decompression-Decryption  **
ILK PRE: Disabled --> Swlib: eLZS compression --> ILK POST:
pad(DRE_PAD_ALG_FIXED) + AES128-CBC encryption
AES Key (len: 16):
0x8e 0x73 0xb0 0xf7 0xda 0x0e 0x64 0x52
0xc8 0x10 0xf3 0x2b 0x80 0x90 0x79 0xe5

AES IV (len: 16):
```

```
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f

Input(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

Output(len:16):
0xdf 0x23 0xe4 0xdc 0x17 0x20 0x50 0x2c
0xad 0x78 0x3e 0x08 0x06 0x58 0x42 0x08

ILK PRE: AES128-CBC decryption + depad(DRE_PAD_ALG_FIXED) --> Swlib: eLZS
decompression --> ILK POST: Disabled
AES Key (len: 16):
0x8e 0x73 0xb0 0xf7 0xda 0x0e 0x64 0x52
0xc8 0x10 0xf3 0x2b 0x80 0x90 0x79 0xe5

AES IV (len: 16):
0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07
0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f

Input(len:16):
0xdf 0x23 0xe4 0xdc 0x17 0x20 0x50 0x2c
0xad 0x78 0x3e 0x08 0x06 0x58 0x42 0x08

Output(len:64):
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

eLZS-AES pairwise test passed

ALL OK!
```

If an error occurs during any of the example tests, “PP_Example: [FAIL]” will be printed in the output and the application will return a non-zero value. Please refer to the *DX SDK User Guide*, USR-0039, for a complete list of the error codes.

7.3.1.2 Kernel Mode Packet Processing example Application

The Linux kernel mode packet processing example demonstrates how to use the Raw Acceleration API session in a Linux kernel module application.

To run the Kernel mode Packet Processing example Application

Step 1 **Type:**

```
insmod pp_example.ko
```

If successful, the module will be loaded and run. The results may be viewed using the Linux command `dmesg`. The output will be similar to the user mode example output.

Step 2 **Unload the application module**

```
rmmod pp_example
```

7.3.2 User Mode Public Key example Application

The user mode public key example demonstrates how to use the public key related functions of the Raw Acceleration API in a user mode application. This example tests all PK operations, including DH, RSA, DSA, ECDH, ECDSA, and also includes a Miller-Rabin primality test and DRBG example.

To run the User mode Public Key example Application

Step 1 **Type:**

```
./pk_example
```

If successful, the application will return a zero value. Otherwise, the application will return a non-zero value that indicates the error condition.

The results will be similar to:

```
--------------begin PK example-------------------
PK submit DHGETPUB command ok
PK submit DHGETSHARE command ok
PK submit RSAENCRYPT command ok
PK submit RSADECRYPT command  ok
PK submit RSASIGN command ok
PK submit RSAVERIFY command ok
PK submit DSAVERIFY command  ok
PK submit DSASIGN command ok
PK submit MODEXP command  ok
PK submit MODMUL command ok
PK submit ECDHGETPUB command ok
PK submit ECDHGETSHARE command ok
PK submit ECDSASIGN command ok
PK submit ECDSAVERIFY command ok
PK submit ECPOINTVERIFY command ok
PK submit ECPOINTMUL command ok
PK submit DRE_PKCMD_MRTEST command ok
PK submit DRE_PKCMD_MRTEST command ok
-->>>Total cmds = 18, Submit cmds = 18
-->>>pk cmd: DHGETSHARE ok
-->>>pk cmd: DHGETPUB ok
-->>>pk cmd: RSADECRYPT ok
-->>>pk cmd: RSAENCRYPT ok
-->>>pk cmd: RSAVERIFY ok
-->>>pk cmd: RSASIGN ok
-->>>pk cmd: DSASIGN ok
-->>>pk cmd: DSAVERIFY ok
-->>>pk cmd: ECDHGETSHARE ok
-->>>pk cmd: ECDHGETPUB ok
-->>>pk cmd: ECDSAVERIFY ok
-->>>pk cmd: ECDSASIGN ok
-->>>pk cmd: MODEXP ok
-->>>pk cmd: MODMUL ok
-->>>pk cmd: ECPOINTVERIFY ok
-->>>pk cmd: ECPOINTMUL ok
----------------end PK example-------------------
-->>>>run pk example success!

-----------------begin DRBG Example---------------
AES 128 CTR DRBG Without Prediction Resistance

result:
1cc50658f0fb5b9c53d0c0517eb793b5
----------------end DRBG Example -----------------
-->>>>run drbg example success!

ALL OK!
```

Please refer to the *DX SDK User Guide*, USR-0039, for a complete list of the error codes.

7.3.3 Kernel Mode Public Key example Application

The Linux mode Public Key example demonstrates how to use the public key related functions of the Raw Acceleration API in a Linux kernel mode application.This example tests all PK operations, including DH, RSA, and DSA.

To run the Kernel mode Public Key example Application

Step 1 **Type:**

```
insmod pk_example.ko
```

If successful, the module will be loaded and run. The results can be displayed using dmesg and will be similar to those for the Public Key user mode example in Section 7.3.2.

Step 2 **Unload the application module**

```
rmmod pk_example
```

7.4 Diagnostic Tools

For instructions about using the diagnostic tool, please refer the *DX SDK User Guide*, USR-0039, and the application man pages: man dx_monitor, man dx_diag, and man dx_status.

Appendix A: Driver Configuration File

The driver configuration file is written in xml format so as to be easy to read and understand. Rows starting with "#" indicate a comment. The general format is "Variable=value", with no space between "Variable", "=" and "value".

A.1 Driver Configuration File Parameters

Please refer to the *DX SDK User Guide*, USR-0039, for guidelines in selecting parameter values that are appropriate to your system application.

A.2 Driver Configuration File Format

An example driver configuration file is shown below.

```
<driver_configuration>
    <HWItems>
        <!--The number of commands in a command ring
            Valid values: 32, 64, 128, 256, 512, 1024, 2048, 4096,
                                8192, 16384, 32768, 65536
            Default: 4096 -->
        <cmds_per_ring>4096</cmds_per_ring>

        <!--Maximum number of descriptors per command that may use the command cache.
            For large commands that require additional descriptors, the SDK will
            dynamically allocate the memory with negligible overhead to process them.
            The minimum value is 32, the value must be even.
            Default: 64 -->
        <data_desc_per_cmd>64</data_desc_per_cmd>

        <!--HW real time verification for COMP, ENC, HASH/AUTH engines
            0: Disable HW real time verification
            1: Enable HW real time verification
            Default: 1 -->
        <real_time_verification>1</real_time_verification>

        <!--Load_balance algorithm
            0: Round Robin, suggested setting when all devices have similar
               throughput.
               This setting will provide the best performance if the packet size is
               greater than 32 kbytes.
            1: Queue Depth, suggested setting when devices have different throughputs.
            2: Ring-CPU Binding, this setting is used for debugging.
            3: Ring-CPU Automatic Binding, this setting will minimize the lock between
               the CPUs. This setting will provide the best performance if the packet
               size is less than 512 bytes.
            Default: 0 -->
        <load_balance_algorithm>0</load_balance_algorithm>
    </HWItems>

    <SWLogItems>
        <!--Log file path and name
            The log_file_name value is a string.
            Both absolute path and relative paths are supported. For example:
            Absolute path: "/a/b/c/d.log". Make sure "/a/b/c" exists and is valid.
            Relative path: "d.log", which is relative to the current working
            directory.
            The file d.log will be created if it doesn't already exist in the path.
            To ignore and discard the log, use the empty string "log_file_name=".
            Default: dresys.log-->
        <log_file_name>dresys.log</log_file_name>

        <!--Log information printing level
            0: (DRE_LOG_EMG, least verbose)
            1: (DRE_LOG_ERR)
            2: (DRE_LOG_WARNING)
            3: (DRE_LOG_INFO)
            4: (DRE_LOG_TRACE most verbose)
            Default: 1 -->
        <log_print_level>1</log_print_level>

        <!--Print log to console
            0: Log message will be saved to a log file
            1: Log message will be saved to a log file and printed to the console
            Default: 1 -->
        <log_redirection>1</log_redirection>

        <!--log file size
            Log file size limitation in MB.
            A value of zero indicates no size limitation.
            Default: 0 -->
        <log_file_size>0</log_file_size>

    </SWLogItems>
    <SWCapabilityItems>
        <!--Failover: Failover to software lib if there are no operational devices
            0: Disable
            1: Enable
            Default: 1 -->
        <failover>1</failover>

        <!--Notification_mode, determines how the results are retrieved
            0: Interrupt mode with kernel thread
            1: (Obsoleted, effectively works as 0)
            2: (Obsoleted, effectively works as 0)
            3: Interrupt mode with tasklet(DPC)
            Default: 3 -->
        <notification_mode>3</notification_mode>

        <!--PP statistics log enable
            0: Disable logging PP statistics
            1: Enable logging PP statistics (has a performance tradeoff,
               especially for small packet sizes)
            Default: 0 -->
        <pp_statistics_enable>0</pp_statistics_enable>

        <!--pp_malloc_mem_threshold, not currently implemented.
            packet <= pp_malloc_mem_threshold: driver will use memory copy
            packet > pp_malloc_mem_threshold: driver will use memory lock
            Default: 1800 -->
        <pp_malloc_mem_threshold>1800</pp_malloc_mem_threshold>

        <!--Max session num
            The maximum number of simultaneous open sessions supported.
            Minimum value is 2, maximum value is 32M.
            Default: 4096 -->
        <max_session_num>4096</max_session_num>

        <!--Max key num
            The maximum number of open key handles supported, including PP
            and PK keys.
            Minimum value is 7, maximum value is 32M.
            Default: 4096-->
        <max_key_num>4096</max_key_num>

        <temperature_check>
            <!--Temperature_Verification
                0: Disable temperature verification
                1: Enable temperature verification
                Default: 1 -->
            <temp_over_enable>1</temp_over_enable>

            <!--Normal_temp: The upper limit of the temperature at which the device
                can still function normally. Only valid when temp_over_enable=1.
                Valid values are 0 to 125 C.
                Default: 105 -->
            <normal_temp>105</normal_temp>

            <!--Over_heat_temp: The temperature at which the device will stop working.
                 Only valid when temp_over_enable=1.
                 Valid values are from 0 to 125 C.
                 Default: 115 -->
            <over_heat_temp>115</over_heat_temp>
        </temperature_check>

        <!--xts dif format
            0: T10/03-310r0 DIF standard
            1: T10/08-044r1 SBC-3 DIF standard
            Default: 0-->
        <xts_dif_format>0</xts_dif_format>

        <!--pk engine enable
            0: Disable
            1: Enable
            Default: 1 -->
        <pk_enable>1</pk_enable>

        <!--pcie_error_detection and recovery
            0: Disable
            1: Enable
            Default: 1 -->
        <pcie_error_recovery_enable>1</pcie_error_recovery_enable>

        <!--pcie_error_interval_threshold
            Only valid when pcie_error_recovery_enable=1.
            SDK will attempt to recover single PCIe error within the time window
```

```
            defined by this parameter. A setting of "0" will disable the threshold
            feature, causing the
            SDK to always attmept to detect and recover from PCIe errors.
            Default: 8 -->
        <pcie_error_interval_threshold>8</pcie_error_interval_threshold>

        <!--rng_bit_rate
            The number of clock cycles between bit samples in the serial-to-parallel
            conversion whitening LFSR inside the RNG engine.
            0: 125 Mbps (quickest, but least random)
            1: 62.5 Mbps
            2: 41.7 Mbps
            3: 31.25 Mbps
            4: 25 Mbps
            5: 20.8 Mbps
            6: 17.9 Mbps
            7: 15.63 Mbps (slowest, but most random)
            Default: 7 -->
        <rng_bit_rate>7</rng_bit_rate>

        <!--rng_sample_interval
            Number of clocks to skip between capturing the 32 bit output of the Seed
            Generator into the RNG buffer.
            0: 32*(rng_bit_rate+1)
            1: 64*(rng_bit_rate+1)
            2: 128*(rng_bit_rate+1)
            3: 256*(rng_bit_rate+1)
            Default: 2 -->
        <rng_sample_interval>2</rng_sample_interval>

        <!--cpu_dma_zero_latency
            Controls the amount of allowed CPU DMA latency.
            Enabling this parameter will increase performance for systems where the
            CPU enters an idle C-state, but will also significantly increase the power
            consumption. Applications that do not experience reduced performance
            should disable this parameter to conserve power.
            0: Disable
            1: Enable.
            Default: 0 -->
        <cpu_dma_zero_latency>0</cpu_dma_zero_latency>
    </SWCapabilityItems>
</driver_configuration>
```

Appendix B: demo Application

This section describes the demo application configuration file, log file format, and the sample files.

B.1 demo Configuration File

The default demo configuration file is written in XML format so as to be easy to read and understand. Refer to the *DX SDK User Guide*, USR-0039, for a detailed explanation of the demo configuration file parameters. All of the demo configuration file parameters must be reviewed and set before running the application.

B.2 demo Configuration File Format

An example demo configuration file is shown below.

```
<demo_configuration>

     <!--

          Perform PP command performance test?

          0: No

          1: Yes

          Default: 1

     -->

     <test_raw_pp>1</test_raw_pp>


     <!--

          Perform PK command performance test?

          0: No

          1: Yes

          Default: 0

     -->

     <test_raw_pk>0</test_raw_pk>


     <!--

          Perform RNG command performance test?

          0: No
          1: Yes

          Default: 0

     -->

     <test_raw_rng>0</test_raw_rng>


     <!--

          Perform DRBG command performance test?

          0: No

          1: Yes

          Default: 0

     -->

     <test_raw_drbg>0</test_raw_drbg>


     <!--

          Run time,  in seconds.

          If stop_sec > 0, demo will run test for the specified time.

          If stop_sec = 0, demo will run the number of commands specified by cmd.

          Default: 10

     -->

     <stop_sec>10</stop_sec>


     <!--

          Total number of commands to test for every session of every thread.

          Only valid when stop_sec=0.

          Default: 20000

     -->

     <cmd>20000</cmd>


     <!--

          The number of threads to run during the test.

          Selected value should be a mutiple of the number of CPUs available in the
          system. Value 0 means auto compute the thread according to CPU number.
```

```
          For async mode, thread = CPU * 1.5; for sync mode, thread = CPU * 3.

          Default: 0

     -->

     <thread>0</thread>


     <!--

       The number of sessions managed by each thread.

          Default:1

     -->

     <sess_per_thread>1</sess_per_thread>


     <!--

          Max number of commands for a single run of each thread.

          The demo application allocates up to "max_per_run" commands for each thread.

          If (cmd < max_per_run) then each thread's allocated command num=cmd, # times
cmd operations submitted=1.

         If (cmd >= max_per_run) then each thread's allocated command num=max_per_run,
         # times cmd operations submitted=(cmd/max_per_run).

         Each thread will then loop on the allocated commands, and each allocated
         command will be submitted to all sessions, until stop_sec expires or the
         number of commands set by "cmd" have been tested.

          Default: 200

     -->

     <max_per_run>200</max_per_run>


     <!--

          Whether to run in asynchronous mode or synchronous mode.

          Only valid if test_raw_pp = 1.

          0: Synchronous mode

          1: Asynchronous mode

          Default: 1

     -->

     <async>1</async>
```

```
<!--
     Packet size. Max value supported is 524288 (512KB).
     Default: 32768
-->
<pkt_size>32768</pkt_size>


<!--
     Source data file
    Specifies the source data file name and location (relative or absolute path).
     Let file_size be the size in bytes of the specified source data file:
       a. If file_size >= pkt_size * max_per_run, demo will read (pkt_size *
          max_per_run) bytes from the source file and construct max_per_run
          number of commands, each pkt_size bytes in length.

          For example, if file_size=800KB, pkt_size=4KB, max_per_run=100. Demo
          reads the first 400KB from the specified file to form 100 4KB
          commands. The remaining 400bytes are discarded.
     b. If file_size < (pkt_size * max_per_run), demo will read the entire file
          to construct "file_size / pkt_size" number of commands, where each
          command is pkt_size bytes long.

          If there are not enough bytes remaining to form a pkt_size command,
          the remaining bytes will be prepended to the starting bytes of the
          file to form a valid pkt_size command.

          For example, if file_size=800KB, pkt_size=64KB, max_per_run=100. Demo
          reads 768KB bytes to form 12 64KB commands, and appends the first 32KB
          of the file to the last 32KB of the source data file to form another
          64KB command, for a total of 13 64KB commands. These 13 64KB commands
          are looped to form the 100 (max_per_run) 64KB commands. If no file is
          specified, demo will use a 256 byte file which has content 0x00 to
          0x255.
     Default: demo
-->
<src_data_file>demo</src_data_file>


<!--
     PP algorithm configuration. This setting does NOT apply to PK commands.
     0: Disable. The demo will loop all supported PP algorithms
     1: Enable.  The demo will run the specified PP algorithm once
```

```
          Default: 1
     -->
     <en_alg_conf>1</en_alg_conf>

     <!--
          Encode or Decode direction, only valid when en_alg_conf=1.
          This configures PP command only, and demo always run both
          private key and public key commands for PK.
          1: Encode
          2: Decode
          3: Both Encode and Decode
          Default: 3
     -->
     <direction>3</direction>

     <!--
          Encryption algorithm, only valid when en_alg_conf=1.
          Valid values: NONE, AES_CBC, AES_CTR, AES_ECB, AES_GCM, AES_XTS, 3DES_CBC,
ARC4
          Default: NONE
     -->
     <enc_algo>NONE</enc_algo>

     <!--
          Compression algorithm, only valid when en_alg_conf=1.
          Valid values: NONE, LZS, ELZS, GZIP, DEFLATE, ZLIB
          Default: DEFLATE
     -->
     <comp_algo>DEFLATE</comp_algo>

     <!--
          Perform stateful compression?
```

```
         Only valid if comp_algo is = DEFLATE, GZIP or ZLIB.

         0 - Stateless

         1 - Stateful

         Default: 0

     -->

     <stateful_comp>0</stateful_comp>


     <!--

         Segment hash algorithm, only valid when en_alg_conf=1

         Valid values: NONE, HASH_SHA1, HASH_SHA256, HASH_SHA384, HASH_SHA512,
HASH_MD5,

              HMAC_SHA1, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512, HMAC_MD5,

              SSLMAC_SHA1, SSLMAC_SHA256, SSLMAC_MD5, AES_GCM_MAC, AES_GMAC, AES_XCBC

         Default: NONE

     -->

     <seg_hash_algo>NONE</seg_hash_algo>


     <!--

         PK algorithm, only valid when en_alg_conf=1.

         Valid values: DH, RSA, DSA, ECDH, ECDSA, ALL (runs all PK algorithms)

         Default: RSA

     -->

     <pk_algo>RSA</pk_algo>


     <!--

         pk_mbits - The key size (in bits) for the specified pk_algo, valid when
         pk_algo != ALL.

         pk_algo = DH pk_mbits = 1024, 1536, 2048, 3072, 4096

         pk_algo = RSA pk_mbits = 1024, 2048, 4096

         pk_algo = DSA pk_mbits = 1024, 2048, 3072

         pk_algo = ECDH pk_mbits = 192, 224, 256, 384, 521

         pk_algo = ECDSA pk_mbits = 192, 224, 256, 384, 521
```

```
        pk_algo = ALL, pk_mbits is ignored

        Default: 256 for ECDH/ECDSA, 1024 for others

    -->

    <pk_mbits>1024</pk_mbits>


    <!--

        The number of bytes to retrieve for each RNG command. Must be <= 2048.

        Default: 128

    -->

    <rng_pkt_size>128</rng_pkt_size>


    <!--

        Sets the underlying algorithm that implements the DRBG.

        Valid values: AES_CTR, DUAL_EC, ALL

        "ALL" will run both AES_CTR and DUAL_EC algorithms

        Default: AES_CTR

    -->

    <drbg_algo>AES_CTR</drbg_algo>


    <!--

       The number of bits to retrieve for each DRBG command. Must be not larger than
       DRE_DRBG_MAX_NUM_BITS_PER_REQ, which is defined as (512*1024) in dre_config.h.
      It will be aligned to next byte boundary as necessary, e.g. 1020 will be
      aligned to 1024

        Default: 1024

    -->

    <drbg_request_bits>1024</drbg_request_bits>

</demo_configuration>
```

B.3 demo Log File Output

The demo application will generate a log file named *demo.log* in the same working directory where the demo application is executed.

The log will record the following information:

- Version number
- Configuration settings
- Run-time information
- Performance statistics

An example log file is shown below.

```
[root@t620-2 dx2_linux]# ./demo demo.cfg.xml
Express DX SDK demo application,version v2.2.1L, version state 0x0
Common Configurations:
    thread=36
    max_per_run=200
    stop_sec=10
PP Configurations:
    async=1
    src_data_file="demo"
    pkt_size=32768B
    sess_per_thread=1
    comp_algo=DEFLATE stateless
    enc_algo=NONE
    hash_algo=NONE
    direction=Encode & Decode
Device Information:
    number of device: 1, commands per ring: 4096
starting ENCODE-CMP_DEFLATE...
  Pkt_size(B)      Commands#     Time(ms)      Mbps     Kpps     CPU Load
  32768            1619200       10062         42184    160      22%
starting DECODE-CMP_DEFLATE...
  Pkt_size(B)      Commands#     Time(ms)      Mbps     Kpps     CPU Load
  32768            1921200       10058         50070    191      24%
demo_main: All perf test are finished
```

Appendix C: sdemo Application

This section describes the sdemo application files.

The default sdemo configuration files are written in such a way as to be easy to read and understood. Refer to the *DX SDK User Guide*, USR-0039, for a detailed explanation of the file parameters. All of the sdemo file parameters should be reviewed and set before running the application.

There are four sdemo files:

- Encode configuration file
- Decode configuration file
- Key file
- IV AAD file

C.1 sdemo Encode Configuration File Format

An example sdemo encode configuration file is shown below.

```
<sdemo_configuration>
    <!--
         The src_data_file specifies the file to be processed.
         Both absolute and relative paths are supported.
         Make sure the file path and name are valid.
         Default: (none, value must be specified)
    -->
    <src_data_file>README.public</src_data_file>

    <!--
         The dst_data_file specifies the file to save the processed data.
         Both absolute path and relative path are supported.
         The file will be created if it does not already exist.
         Default: (none, value must be specified)
    -->
    <dst_data_file>README.public.encode</dst_data_file>

    <!--
        Direction
        1: Encode
        0: Decode
        Default: 1
    -->
    <direction>1</direction>

    <!--
         Operation type
         Valid value: COMP, ENC, SEG_HASH, PASSTHRU
         Default: COMP
    -->
    <op_type>COMP</op_type>
```

```
<!--
    Compression algorithm. Only valid when op_type is COMP.
    Valid values: LZS, ELZS, DEFLATE, GZIP, ZLIB
    Default: DEFLATE
-->
<comp_algo>DEFLATE</comp_algo>

<!--
    Enable stateful compression.
    Only valid when op_type is COMP and comp_algo is DEFLATE, GZIP or ZLIB.
    0: Stateless
    1: Stateful
    Default: 0
-->
<stateful_comp>0</stateful_comp>

<!--
    Submit the source file in single command or block by block ?
    Only valid when stateful_comp is disabled.
    A setting of 0 forces the source file to be submitted in a single command.
    Otherwise the file will be submitted block by block.
    Default: 0
-->
<block_size>0</block_size>

<!--
    Compression mode
    Only valid when op_type is COMP and comp_algo is DEFLATE, GZIP or ZLIB.
    Valid values: STORE, STATIC, DYNAMIC, OPTIMAL
    Default: OPTIMAL
-->
<comp_mode>OPTIMAL</comp_mode>

<!--
    Encryption algorithm. Only valid when op_type is ENC.
    Valid values:
        AES_ECB_128, AES_ECB_192, AES_ECB_256,
        AES_CBC_128, AES_CBC_192, AES_CBC_256,
        AES_CTR_128, AES_CTR_192, AES_CTR_256,
        AES_GCM_128, AES_GCM_192, AES_GCM_256,
        AES_XTS_256, AES_XTS_512,
        3DES_CBC, ARC4
    Default: AES_CBC_128
-->
<enc_algo>AES_CBC_128</enc_algo>

<!--
    Segment hash algorithm. Only valid when op_type is SEG_HASH.
    Valid value for both encode and decode:
        HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512
        SSLMAC_MD5, SSLMAC_SHA1, SSLMAC_SHA256,
        AES_GMAC_128, AES_GMAC_192, AES_GMAC_256,AES_XCBC
    Valid value for encode, but not decode:
        HASH_MD5, HASH_SHA1, HASH_SHA256, HASH_SHA384, HASH_SHA512
    Default: HASH_SHA1
-->
```

```
    <seg_hash_algo>HASH_SHA1</seg_hash_algo>

    <!--
        CRC configuration
        0: no crc
        1: host crc
        2: hw crc
        Default: 0
    -->
    <crc_config>0</crc_config>

    <!--
        Command target
        0~31: User selects the specific device 0~31, the max number of devices
supported is 32
        32:   swlib
        33:   SDK selects the device via load balancing
        Default: 33
    -->
    <cmd_target>33</cmd_target>
</sdemo_configuration>
```

C.2 sdemo Decode Configuration File Format

An example sdemo decode configuration file is shown below.

```
<sdemo_configuration>
    <!--
         The src_data_file specifies the file to be processed.
         Both absolute and relative paths are supported.
         Make sure the file path and name are valid.
         Default: (none, value must be specified)
    -->
    <src_data_file>README.public.encode</src_data_file>

    <!--
         The dst_data_file specifies the file to save the processed data.
         Both absolute path and relative path are supported.
         The file will be created if it does not already exist.
         Default: (none, value must be specified)
    -->
    <dst_data_file>README.public.check</dst_data_file>

    <!--
        Direction
        1: Encode
        0: Decode
        Default: 1
    -->
    <direction>0</direction>

    <!--
         Operation type
         Valid value: COMP, ENC, SEG_HASH, PASSTH
         Default: COMP
```

```
-->
<op_type>COMP</op_type>

<!--
    Compression algorithm. Only valid when op_type is COMP.
    Valid values: LZS, ELZS, DEFLATE, GZIP, ZLIB
    Default: DEFLATE
-->
<comp_algo>DEFLATE</comp_algo>

<!--
    Enable stateful compression.
    Only valid when op_type is COMP and comp_algo is DEFLATE, GZIP or ZLIB.
    0: Stateless
    1: Stateful
    Default: 0
-->
<stateful_comp>0</stateful_comp>

<!--
    Submit the source file block by block ?
    Only valid when stateful_comp is disabled.
    A setting of 0 forces the source file to be submitted in a single command.
    Otherwise the file will be submitted block by block.
    Default: 0
-->
<block_size>0</block_size>

<!--
    Compression mode
    Only valid when op_type is COMP and comp_algo is DEFLATE, GZIP or ZLIB.
    Valid values: STORE, STATIC, DYNAMIC, OPTIMAL
    Default: OPTIMAL
-->
<comp_mode>OPTIMAL</comp_mode>

<!--
    Encryption algorithm. Only valid when op_type is ENC.
    Valid values:
        AES_ECB_128, AES_ECB_192, AES_ECB_256,
        AES_CBC_128, AES_CBC_192, AES_CBC_256,
        AES_CTR_128, AES_CTR_192, AES_CTR_256,
        AES_GCM_128, AES_GCM_192, AES_GCM_256,
        AES_XTS_256, AES_XTS_512,
        3DES_CBC, ARC4
    Default: AES_CBC_128
-->
<enc_algo>AES_CBC_128</enc_algo>

<!--
    Segment hash algorithm. Only valid when op_type is SEG_HASH.
    Valid value for both encode and decode:
        HMAC_MD5, HMAC_SHA1, HMAC_SHA256, HMAC_SHA384, HMAC_SHA512
        SSLMAC_MD5, SSLMAC_SHA1, SSLMAC_SHA256,
        AES_GMAC_128, AES_GMAC_192, AES_GMAC_256,AES_XCBC
    Valid value for encode, but not decode:
        HASH_MD5, HASH_SHA1, HASH_SHA256, HASH_SHA384, HASH_SHA512
        Default: HASH_SHA1
    -->
    <seg_hash_algo>HASH_SHA1</seg_hash_algo>

    <!--
        CRC configuration
        0: no crc
        1: host crc
        2: hw crc
        Default: 0
    -->
    <crc_config>0</crc_config>

    <!--
        Command target
        0~31: User selects the specific device 0~31, the max number of devices
supported is 32
        32:   swlib
        33:   SDK selects the device via load balancing
        Default: 33
    -->
    <cmd_target>33</cmd_target>
</sdemo_configuration>
```

C.3 sdemo Key File Format

An example sdemo key configuration file is shown below.

```
<sdemo_key>
    <!--
        This file includes all the sections required to configure the key for
        the Encryption and MAC algorithms supported by sdemo.
        The key used by sdemo is determined by the "op_type" and "xxx_algo"
        settings in the sdemo configuration file.

        Users may change the settings in this file to suit their specific test.


Note:
        1. For an encryption key, the key size SHOULD be correct. It will be
           truncated to the correct size if a longer key is specified, but will
           not be padded if a shorter key is specified.
        2. HMAC/SSL-MAC keys share the same key entry, but have
           different key size restrictions. See the comments below.
    -->

    <!--  AES CBC key -->

<cbc_128>ed00d865dc8ae060c9f07e18d365b173</cbc_128>
<cbc_192>2a424e00ffc79cd1206b46aa8b974a19aa3c00a26f5368ff</cbc_192>
<cbc_256>1922ba48dc46ca41cb049154e58be9186e49779f87d1063369538a2015fa73b7</cbc_256>

    <!-- AES CTR key -->

<ctr_128>484af300834b82194a346437869690c4</ctr_128>
<ctr_192>5ded9a18eda19c6d8626d28bbb282e54e359fa0304af97db</ctr_192>
```

```
<ctr_256>78ee29dc7a1accc46e7d3e8b6d1fe62b7c6838fd97ef32fc46ca51ad5bd632f5</ctr_256>

    <!-- AES ECB key -->

<ecb_128>e88e61d2a9355afb5f60915cba46aadf</ecb_128>
<ecb_192>d1098dbc83a7b65d08252bb8f40cb5772482af24348769e1</ecb_192>
<ecb_256>a5f7b427a5d71b5bf56b62ed9294f61c138184fc6547a1c66d1851200f58d9b1</ecb_256>

    <!-- AES GCM key -->

<gcm_128>41d00fa7fddf3b186705e320df4786ca</gcm_128>
<gcm_192>c06748d29481f891cc584ccb12b4deeaabc23dd28d4fb451</gcm_192>
<gcm_256>9088314230e29ca87989e34825dce1a7393022b3e6fb5894224f2c75c5c8321e</gcm_256>

    <!-- AES XTS key -->

<xts_256>1f9dd0a558e783ef4af5ec76ded235304f4a8b6ffb9b44f2a452d66c8d8797bb</xts_256>
<xts_512>e31f8dd23a669da7687f2e95690099dbed765d64f4b3743ae5f47a21a133e2f73203b78d88788
0d5b1ad41fe437e72ac551c93fb153078b53a20096c3c53a693</xts_512>

    <!-- 3DES CBC key: each byte of the key must have the correct parity, or will
return an error -->

<des3_cbc_192>3e498f1a4f324c680d7064f15285c44fbfc7d32c948aae76</des3_cbc_192>

    <!-- ARC4 key -->

<arc4_2048>c64a7ca00863257eeeb4960c0aa18a46a2fb03c2a044a518bc389df4b8b8a38ac64a7ca0086
3257eeeb4960c0aa18a46a2fb03c2a044a518bc389df4b8b8a38ac64a7ca00863257eeeb4960c0aa18a46a
2fb03c2a044a518bc389df4b8b8a38ac64a7ca00863257eeeb4960c0aa18a46a2fb03c2a044a518bc389df
4b8b8a38ac64a7ca00863257eeeb4960c0aa18a46a2fb03c2a044a518bc389df4b8b8a38ac64a7ca008632
57eeeb4960c0aa18a46a2fb03c2a044a518bc389df4b8b8a38ac64a7ca00863257eeeb4960c0aa18a46a2f
b03c2a044a518bc389df4b8b8a38ac64a7ca00863257eeeb4960c0aa18a46a2fb03c2a044a518bc389df4b
8b8a38a</arc4_2048>

    <!--
        MAC-MD5 key, applies to both SSLMAC and HMAC
        For HMAC-MD5:   supports key size from 1 byte to 64 bytes. Will be
                        truncated to 64 bytes if longer key is specified.
        For SSLMAC-MD5: supports key size 16 bytes. Will be truncated to 16
                        bytes if longer key is specified.
    -->


<md5_mac>ff0a32ba141ec97ae55e79747507a1d4</md5_mac>

    <!--
        MAC-SHA1 key, which applies to both SSLMAC and HMAC

For HMAC-SHA1:   supports key sizes from 20 byte to 64 bytes. Will be
                         truncated to 64 bytes if longer key is specified.
        For SSLMAC-SHA1: supports 20 bytes key size. Will be truncated to 20
                         bytes if longer key is specified.
    -->
<sha1_mac>cfb0d0bbd7733fd1cf403f035bbcb97424936c1f</sha1_mac>

    <!--
```

```
        MAC-SHA256 key, applies to both SSLMAC and HMAC
        For HMAC-SHA256:   supports key sizes from 1 byte to 64 bytes. Will be
        truncated to 64 bytes if longer key is specified.
        For SSLMAC-SHA256: supports 32 bytes key size. Will be truncated to 32
        bytes if longer key is specified.
    -->


<sha256_mac>39bf946b56ba906ef562db4a206c1944e58813edda8afc67736be77c5b874328</
sha256_mac>

    <!--
        MAC-SHA384 key, applies to HMAC only
        For HMAC-SHA384: supports key sizes from 1 byte to 64 bytes. Will be
                         truncated to 64 bytes if longer key is specified.
    -->


<sha384_mac>39bf946b56ba906ef562db4a206c1944e58813edda8afc67736be77c5b874328</
sha384_mac>

    <!--
            MAC-SHA512 key, applies to HMAC only
            For HMAC-SHA512: supports key sizes from 1 byte to 64 bytes. Will be
            truncated to 64 bytes if longer key is specified.
    -->


<sha512_mac>e31f8dd23a669da7687f2e95690099dbed765d64f4b3743ae5f47a21a133e2f73203b78d88
7880d5b1ad41fe437e72ac551c93fb153078b53a20096c3c53a693</sha512_mac>

</sdemo_key>
```

C.4 sdemo IVAAD File Format

An example sdemo IVAAD file is shown below.

```
<sdemo_ivaad>
    <!--
        IV length in bytes.
        The valid/supported values are 16 for AES and 8 for 3DES.
    -->
    <iv_length>16</iv_length>

    <!--
        IV data is specified in hex format, and its length must be iv_length bytes.
        Notes:
        a. AES-XTS uses a 16 byte IV, and the last 8 bytes must be 0. This is
           a HW restriction. The first 8 bytes can be any value.
        b. The IV data specified length must be iv_length bytes.
        -->
    <iv_data>1e2393d86d660c9ff02176ae00000001</iv_data>

    <!--
        AAD length in bytes.
    -->
    <aad_length>64</aad_length>

    <!--
        AAD data is specified in hex format, and its length must be aad_length bytes.
    -->

<aad_data>04893f6520ab965bd3165cfbfc9c9ef62039149121fee851c806996e2e0b48f5af4806bda46f
a1011a0d1bc01e8b8f000aea2e085f10d197a7adfde51e6665c2</aad_data>
</sdemo_ivaad>
```

C.5 sdemo DRBG Configuration File Format

An example sdemo DRBG configuration file is shown below.

```
<sdemo_drbg>
    <!--
        DRBG configuration. Must enable DRE_DEBUG_DRBG in h/dre_config.h.
        The user must have a very good understanding of DRBG before editing this file
and testing the DRBG related functions.
    -->
    <!--
        DRBG type: The underlying algorithm type to implement the DRBG
        Valid values: AES_128, AES_192, AES_256, P_256, P_384, P_521
    -->
    <drbg_type>AES_256</drbg_type>

    <!--
        Hash algorithm: The underlying hash algorithm used by the DRBG
        Valid values: SHA_1, SHA_224, SHA_256, SHA_384, SHA_512
    -->
    <hash_algo>SHA_512</hash_algo>

    <!--
        Use a derivation function to derive a seed from the seed material.
        0: No
        1: Yes
    -->
    <use_df>0</use_df>

    <!--
        Prediction resistance.
        0: Disable
        1: Enable
    -->
    <prediction_resistance>0</prediction_resistance>

    <!--
        The length of input entropy in bits
    -->
    <entropy_len>384</entropy_len>

    <!--
        The length of input nounce in bits
    -->
    <nonce_len>128</nonce_len>
    <!--
        The length of personal string in bits
    -->
    <personalstr_len>384</personalstr_len>

    <!--
        The length of additional input in bits
    -->
    <additionalinput_len>384</additionalinput_len>

    <!--
        The length of the required security strenth in bits
    -->
    <required_len>128</required_len>

    <!--
        The entropy data in hex format, which must be entropy_len bytes
    -->

<entropy_data>fbcf1b6116897823f5d896e34e640b299a3ff8a5edf2fedb16ca7f10fa5e18762c635e96
cfb3d6fcaf9939289c61e8b3</entropy_data>

    <!--
        If prediction_resistance = 1, this is the entropy data for the first
        prediction resistance use, otherwise used for the reseed function.
    -->

<entropy_data_pr>917976eee0cf9ec2d5d4239b128c7e0ab7d28bd67ca3c6e50eaac76bae0dfa530679a
1ed4d6a0ed89dbe1b31937becfb</entropy_data_pr>

    <!--
        If prediction_resistance = 1, this is the entropy data for the second
        prediction resistance use, otherwise is not used and ignored.
    -->
    <entropy_data_pr2></entropy_data_pr2>

    <!--
        The nonce data in hex format, which must be nonce_len bytes
    -->
    <nonce_data>1296f052f38d81cfde86f2994396b9f0</nonce_data>

    <!--
        The personal string data in hex format, which must be personalstr_len bytes
    -->

<personal_str>630d78f5908e3247b04d37600996bcbf977a621445bd8dcc69fb03e1801cc7e22af9373f
664d62d910e0adc89af0a86d</personal_str>

    <!--
        The additional input data in hex format, which must be additionalinput_len
bytes
    -->

<additional_input>36c61360bb14ad22b038aca61816932586b7dcdc36982bf96833d3c6ffce8d155982
76ed6f8d49742fdadc1f17d0de17</additional_input>
```

```
    <!--
        The additional reseed input data in hex format, which must be
additionalinput_len bytes
    -->

<additional_reseed_input>d2465022101463f7ea0fb97e0de19407af094431ea64a4185bf9d8c2fa034
7c53943d53b628664ea2c738cae9d989829</additional_reseed_input>


<additional_input2>8cab18f8c3ec185cb31e9dbe3f03b400989daeebf494f8428fe33907e1c9ad0b1fe
dc0baf6d1ec27867bd6559b60a5c6</additional_input2>

    <!--
        The DRBG output data in hex format, which must be required_len bytes
    -->
    <returned_bits>efd2d85cdc62259faa1e2c67f60232e2</returned_bits>

</sdemo_drbg>
```

D Document Revision History

This section lists the additions, deletions, and modifications made to this document for each release of this document.

Document Revision A01

Initial release.

Document Revision A02

Update 1. Updated to 2.0.0Lb throughout.

Update 2. Section 4.2 Makefile Options: added options BUILD_HW_FPGA_TEST and BUILD_SW_FPGA_TEST to the Exar package.

Update 3. Section 4.3.2 Public + Exar Package Build Results: removed the file dx.cfg.

Update 4. Section 6.2 Installing the Driver: updated instructions for disabling no_snoop.

Update 5. Section 7.3.1 Packet Processing example Application: added description of new FPGA example cases.

Update 6. Section 7.3.1.1 Packet Processing User Mode example Application: added output for new FPGA example cases.

Update 7. Section B.2 demo Configuration File Format: updated this file.

Document Revision A03

Update 1. Updated to 2.0.0L throughout.

Update 2. Chapter 1 Introduction: updated list of supported OS.

Update 3. Section 6.2 Installing the Driver: removed the section describing the no_snoop feature.

Update 4. Section A.2 Driver Configuration File Format: removed the parameter HwFailToSoftware.

Document Revision A04

Update 1. Updated to 2.1.0L throughout.

Update 2. Section 1 Introduction: added new supported operating systems. SLES11 SP2 kernel version 3.0.10 and Fedora 19 kernel version 3.9.4

Update 3. Section 4.3 Support for Exar Hardware: added this new section.

Update 4. Section 4.4 Build the SDK: changed all references to kernel version 2.6.18-53.el5 to 2.6.32-220.el6.x86_64. Changed all references to libpan.so.2.0.0 to libpan.so.2.1.0.

Update 5. Section 7.1.1 Editing the sdemo Configuration File: changed encode comp operation to DEFLATE.

Update 6. Section 7.2.1 Editing the demo Configuration file: added parameters test_raw_drbg and stop_sec (these were actually added to SDK 2.0.0L).

Update 7. Section A.2 Driver Configuration File Format: replaced the configuration file with the latest version.

Update 8. Appendix B demo Application: replaced all configuration files with the latest versions.

Update 9. Appendix C sdemo Application: replaced all configuration files with the latest versions.

Document Revision A05

Update 1. Updated to 2.2.0L throughout.

Update 2. Chapter 1 Introduction: added support for Ubuntu 14.04 and CentOS 7.

Document Revision A06

Update 1. Updated to 2.2.1L throughout.

48720 Kato Road
Fremont, CA 94538
p: 510.668.7000
www.exar.com