Intel® IXP400 Software: Codelets

Application Note

December 2004

Document Number: 300987-002

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at http://www.intel.com.

BunnyPeople, Celeron, Chips, Dialogic, EtherExpress, ETOX, FlashFile, i386, i486, i960, iCOMP, InstantIP, Intel, Intel Centrino, Intel Centrino logo, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Inside, Intel Inside logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Xeon, Intel XScale, IPLink, Itanium, MCS, MMX, MMX logo, Optimizer logo, OverDrive, Paragon, PDCharm, Pentium, Pentium II Xeon, Pentium III Xeon, Performance at Your Command, Sound Mark, The Computer Inside., The Journey Inside, VTune, and Xircom are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Copyright © 2004, Intel Corporation

Contents

Figures

Revision History

Date	Revision	Description
December 2004	002	Updated product branding. Change bars were retained from the previous release of this document (001).
February 2004	001	Initial release.

1.0 Introduction

This document serves as a quick reference for those using some of the most commonly used Intel® IXP400 Software codelets. The document assumes that the codelets have been successfully compiled as per instructions in IXP400 software release notes and are available to download for the Intel® IXDP425 / IXCDP1100 Development Platform.

This document describes various options for running the IXP400 software codelets in the VxWorks*, Linux, and Microsoft* Windows* CE .NET environment, and also briefly describes various intricacies involved in running a codelet with the Ethernet END driver or multiple codelets at the same time.

Note: The codelets described in this document are for demonstration purposes only — they should not be considered fully functional applications.

1.1 Acronyms

AAL	ATM Adaptation Layer
ADSL	Asymmetric Digital Subscriber Line
API	Application Programming Interface
AQM	Advanced Queue Manager
DMA	Direct Memory Access
END	Enhanced Network Driver
HSS	High-Speed Serial
ISR	Interrupt Service Routine
NPE	Network Processing Engine
OAM	Operation Maintenance and Administration
USB	Universal Serial Bus
UTOPIA	Universal Test and Operations Physical Interface for ATM

2.0 Overview

IXP400 software codelets are example code that utilize the access-layer components and operating system abstraction layers of the IXP400 software. While not exhaustive examples of the functionality available to the developer, codelets do provide a good basis from which to begin code development for test harnesses, performance-analysis code, and even functional applications to take to market.

3.0 Codelets Design

The codelets themselves are an independent, standalone entity that encompass all the relevant components that they depend upon. The codelet first goes through the process of system initialization, initializing the Q-manager, setting up the dispatcher loop, downloading the NPE image, then running complete events for the desired application. It is assumed that only one of the codelets would be used to exercise a desired application at any one time.

3.1 System Initialization Sequence

Each of codelets starts with a build ID image definition for the build image that must be downloaded into the corresponding network processor engines (NPEs). Once this is done, the first step is initialization of the IxQMgr, i.e., the Q-manager software component. The initialization of IxQMgr first requires a call to ixQMgrInit(), which takes no parameters and returns success or failure. No other ixQMgr functions may be called before this. If the Q-manager has been previously initialized by another codelet or the Ethernet END driver, then the current codelet will return a failure and may exit the codelet. After initialization, the queues are configured and the dispatcher started. ixQMgrDispatcherLoopRun() may be registered as an ISR for the AQM interrupts, or it may be called from a client polling mechanism that would read the queues status at regular intervals and call the dispatcher when the queue status changes. In the ISR mode, the dispatcher is called in the context of an interrupt.

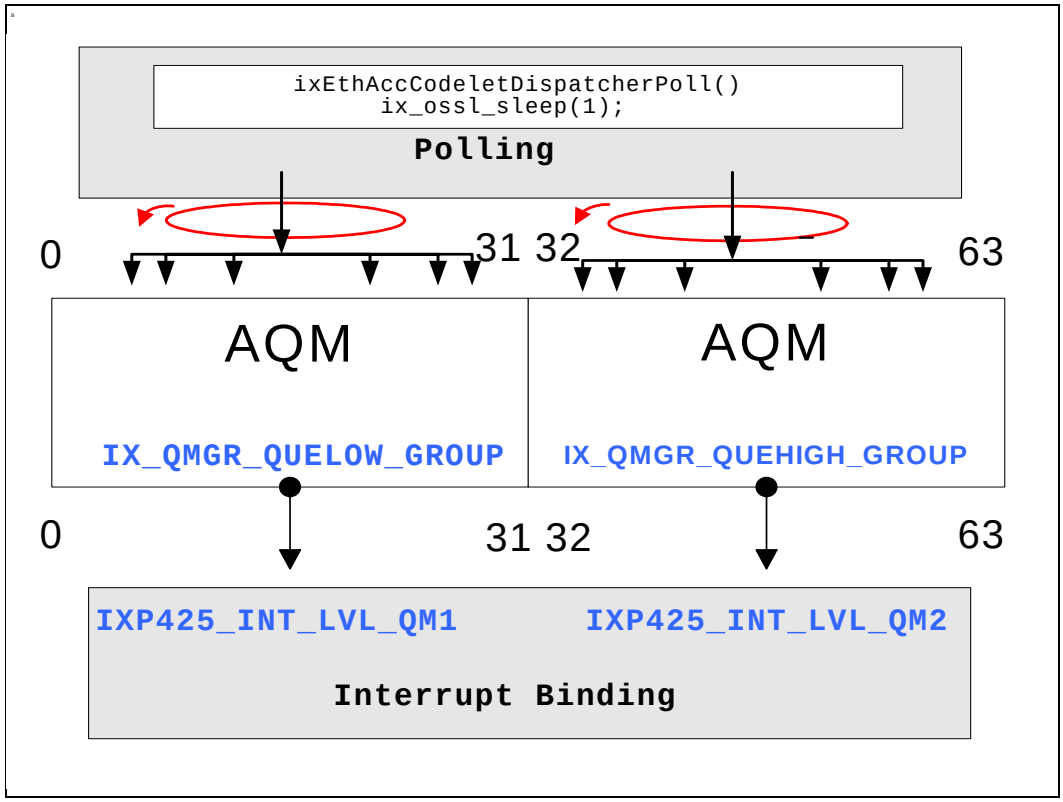
As shown in Figure 1, a parameter ‘IX_QMGR_QUEXX_GROUP’ is passed to the ixQMgrDispatcherLoopRun() function, which determines the queues that are being serviced by the dispatcher loop each time the function ixQMgrDispatcherLoopRun() is called. The parameter will specify if queues 0-31 (QUELOW) or 32-63(QUEHIGH) are serviced. Interrupt AQM 0-31 and AQM 32-63 have their own separate interrupt lines into the interrupt controller. AQM 0-31 can select interrupt sources from Full, Nearly-Full, Empty, Nearly-Empty. AQM 32-63 have an option of Nearly-Empty only. You can have the AQM queues serviced in either a polling mode or in an interrupt mode. In interrupt mode, the QUE-group needs to be bound with respective interrupts as shown in Figure 1. In polling mode, the QUE status is polled as per the polling time specified in the corresponding operating-system configuration file.

Once this is done, the next step is to download the NPE images and initialize the NPE message handlers. Initialization of NPEs is the last step of the system initialization process, after which the system is ready to run the desired application. Typically, at end of the system initialization, the next step is to initialize all the required access components and implement the required API function calls to complete the application code.

In Windows* CE .NET, the queues in the hardware queue manager triggers the interrupts. The interrupts are served by the interrupt service routine (ISR) in Kernel mode. The ISR translates each of the interrupt into an event that is sent to an interrupt service thread. The IxQmgr (Queue

Manager) software module in the Intel® IXP400 Software Access Library has an interrupt service thread that collects these queue's related events. The Queue Manager registers a dispatch loop with the IST. The dispatcher loop checks all the queues that requires processing and looks for an associated callback registered by either the device driver or the application.

Figure 1. Dispatch Loop: Polling V/S Interrupts



3.2 Component Initialization Sequence

Once the system initialization is done, the next step in making the IXP42X product line processors CSR release API calls come alive is the initialization of the access components. The following steps are used to initialize access components from the codelets application code:

- Initialize Access component
- Initialize Interface ports
- Initialize Interface PHYs
- Program MAC addresses
- Set ports to promiscuous mode if using EthAcc components
- Initialize MBUF pool

3.3 Functional Sequence

Each of codelets has a common list of functionality, starting with system initialization and followed by the component initialization. The following is a common, sequential list of functionality for the codelets:

The codelets demonstrate:

- How the NPE is initialized, how the NPE image is downloaded to the NPE, and how the NPE is started.
- How the Queue Manager is initialized to polling or interrupt mode.
- How to set up the callback functions, and, if needed, how to register contexts with Access layer component.
- How the registration should be done for the different operations using the register API.
- How the perform API can be used after successful registration.
- The use of the unregistered API in the event of re-starting the codelet.
- Demonstrate performance-monitoring tasks.

3.4 Running Multiple Codelets

Since each of the codelets initialize the system from its lowest level, prior to running another codelet special care should be taken to prevent re-initialization of pre-configured components that are already in use. The Ethernet END driver for the NPEs by can be viewed as a codelet by itself. The Linux network device driver and VxWorks END driver for the NPE-based Ethernet interfaces act in a similar manner as the codelets provided with the IXP400 software. Hence, special care should be taken while exercising the Ethernet END driver along with a codelet or while exercising multiple codelets at the same time.

The following is a list of modifications that need to be made while running multiple codelets/ Ethernet END driver at the same time.

- Only one of the codelets should define the NPE build IDs.

  In case the codelets have a different build ID, choose the one that has a greater build ID number.

- Only one of the codelets should initialize the IxQMgr Queue Manager.
- Only one of the codelets should make a call to the dispatcher loop.

  With exception of the ADSL/ATM codelet, all codelets use the lower queue in the IxQMgr dispatch group queues. Hence, while using multiple codelets with ATM/ADSL, two dispatcher loops will be defined.

- Following system initialization, component initialization should only be done once.
- If you are using Ethernet END drivers in VxWorks, comment-out the ‘endFindByName’ IF statements to allow the Ethernet END driver to run in parallel with any other codelet except for the Ethernet codelet.

  Two codelets using the same queue in the dispatcher loop cannot be used at the same time; this can cause contention for the callback functions, i.e., the Ethernet codelet cannot be used with the Ethernet END driver.

4.0 Running the Codelets

This application note assumes that the codelets have been compiled successfully as per the instructions in the IXP400 software release notes and that the codelets modules have already been downloaded onto the IXDP425 / IXCDP1100 platform.

Note: It is assumed that the IXP400 software access module has already been loaded prior to loading the codelets; for more information, refer to the appropriate software release notes.

4.1 Ethernet Codelet

4.1.1 EthAcc Codelet in VxWorks*

For the Ethernet codelet, the ixEthAccCodeletMain() function is used as a single point of execution for the EthAcc codelet. This function allows the user to enter a selection for different types of supported operations as described below:

Usage:

- Load the Ethernet codelet module from your target server console as follows:

```
ld < Codelet_EthAccTest.out
```

- Call the main function from your target server console as follows:

```
ixEthAccCodeletMain (operationType)
```

Where operationType:

1 = To sink received frames as fast as possible for available ports.

2 = To software loopback received frames to the same port for available ports.

3 = To generate and transmit frames from Port 1, remote loopback by using an external cross-over cable to Port 2, and received on Port 2 (TxGenRxSink).

4 = To generate frames and perform PHY loopback on the same port for available ports.

5 = To transmit any frame received on one port through the other one (bridge).

6 = To activate Ethernet MAC learning facility.

4.1.2 EthAcc Codelet in Linux

For the Ethernet codelet in Linux, the ixEthAccCodeletMain() serves as a single point of execution for the EthAcc codelet. The operation selected will be executed when the user issues '`insmod`' in command prompt as follows:

```
insmod ixp400_codelets_ethAcc.o operationType=<x>
```

Where x:

1 = To sink received frames as fast as possible for available ports.

2 = To software loopback received frames to the same port for available ports.

3 = To generate and transmit frames from Port 1, remote loopback by using an external cross cable to Port 2, and received on Port 2 (TxGenRxSink).

4 = To generate frames and perform PHY loopback on the same port for available ports.

5 = To transmit any frame received on one port through the other one (bridge).

6 = To activate Ethernet MAC learning facility.

4.1.3 EthAcc Codelet in Windows* CE .NET

1. To build the codelet, browse to the specific codelet you want to build in the Catalog window. All the codelets are listed under ‘BSPs’>‘INTEL IXDP425: ARMV4I’>‘IXP4XX Processor Variants’>‘IXP425’.
2. Select the codelet you wish to build. Right-click and select ‘Add to Platform’ from the drop down menu.
3. Build the platform.
4. Once you download the platform you can run the codelet by using ‘Target’ and then select ‘Run Programs...’
5. Select the IxEthAccCodelet.exe codelet from the list.
6. Click ‘Run’.
7. The codelet uses SerConsole and so the menus will be printed out and input accepted from the UART ‘COM1’ port as follows:

********** ethAcc Codelet **********

```
  1. All frames received (from external device) will be sinked for
available ports.
  2. All frames received are software loopbacked to the same port
for available ports.
  3. Frames generated and transmitted from port 1, remote
loopbacked to port 2 by using cross cable
     and received on port 2.
  4. Frames generated and PHY loopbacked on the same port for
available ports.
  5. Frames received on one port will be transmitted through the
other port.
  6. Ethernet Learning Facility where it adds some static and
dynamic entries. Dynamic entries are then aged and verified that
they no longer appear in the database.


100. Exit ethAcc Codelet
Enter Test Number:
```

4.2 Crypto Codelet

The Crypto codelet demonstrates how the Security Hardware Accelerator API can be used for cryptographic purposes. This codelet demonstrates the following three different cryptographic operations:

1. Encryption and decryption. Default is DES-CBC.

2. Authentication calculation and check. Default is SHA1.

3. Combined service of encryption/authentication calculation and authentication check/ decryption. Default is DES-CBC mode with SHA1.

4.2.1 Crypto Codelet in VxWorks*

The ixCryptoAccCodeletMain() function is used as the entry point of execution for the Crypto codelet. This function allows the user to enter a selection for different operations as described below with different packet lengths.

Usage:

ixCryptoAccCodeletMain (operationType, packetLen)

Where operationType:

1 = To encrypt and decrypt packets using selected cipher algorithm.

2 = To authenticate packets using selected authentication algorithm.

3 = To encrypt/authenticate and authenticate/decrypt packets using selected cipher algorithm and selected authentication algorithm.

Where packetLen:

Packet length ranges from 64 bytes to 65456 bytes, if cipher

Note: If the algorithm is DES/3DES, packet length must be multiples of 8 bytes (cipher block length); AES algorithms must have a packet length that is multiples of 16 bytes.

4.2.2 Crypto Codelet in Linux

With Linux, the ixCryptoAccCodeletMain() function is used as the entry point of execution for the cryptoAcc codelet. This function allows the user to enter a selection for different operations as described below with different packet lengths. The selected operation will be executed when user issues '`insmod`' at command prompt.

```
insmod csr_codelets_cryptoAcc.o operationType=<x> packetLen=<y>
```

Where x:

1 = To encrypt and decrypt packets using selected cipher algorithm.

2 = To authenticate packets using selected authentication algorithm.

3 = To encrypt/authenticate and authenticate/decrypt packets using selected cipher algorithm and selected authentication algorithm.

Where y:

Packet length ranges from 64 bytes to 65456 bytes, if cipher

Note: If the algorithm is DES/3DES, packet length must be multiples of 8 bytes (cipher block length); AES algorithms must have a packet length that is multiples of 16 bytes.

4.2.3 Crypto Codelet in Windows* CE .NET

1. To build the codelet, browse to the specific codelet that you want to build in the Catalog window. All the codelets are listed under ‘BSPs’>‘INTEL IXDP425: ARMV4I’>‘IXP4XX Processor Variants’>‘IXP425’.
2. Select the codelet you wish to build. Right-click and select ‘Add to Platform’ from the drop-down menu.
3. To build the Crypto codelet, be sure to also include “NpeDl component” in the build.
4. Build the platform.
5. Once you download the platform you can run the codelet by using ‘Target’ and select ‘Run Programs...’
6. Select the IxCryptoAccCodelet.exe codelet from the list.
7. Click ‘Run’.

The codelet uses SerConsole, so the menus will be printed out and input accepted from UART ‘COM1’ as follows:

```
****ixCrypto Codelet****
1 = To encrypt and decrypt packets using selected cipher
algorithm.
2 = To authenticate packets using selected authentication
algorithm.
3 = To encrypt/authenticate and authenticate/decrypt packets using
selected cipher algorithm and selected authentication algorithm.
Enter Test Number:
```

4.3 HSS Codelet

The HSS codelet supports the following top-level operations:

1. Test Packetised and Channelised Services, with the codelet acting as data source/sink and HSS as loopback.
2. The codelet will transmit data and verify that data received is the same as is transmitted. The codelet runs for IX_HSS_CODELET_DURATION_IN_MS ms.

where ms = time in milliseconds

Assumptions:

In Channelised service, the codelet transmits traffic continuously. When the codelet runs up to IX_HSS_CODELET_DURATION_IN_MS ms, the Tx counter is larger than the Rx counter. This is due to the fact that traffic submitted to the NPE (i.e., the Tx counter increases) does not get transmitted out by the NPE when HSS service is disabled. This type of traffic will be dropped and not loopbacked at HSS (hence, the Rx counter does not increase).

In Packetised-raw mode service (client 1 and 3), the Rx counter will be larger than the Tx counter because in this service idle packets are received by the Intel XScale® Core, which causes the Rx counter to be larger than the Tx counter. As for packetised-HDLC service, idle packets are handled in the HDLC coprocessor and not passed to the Intel XScale core (hence, the Rx counter does not increase).

4.3.1 HSS Codelet in VxWorks*

In VxWorks, the ixHssAccCodeletMain() function is used as a single point of execution for the HssAcc Codelet. It allows user to enter an operation type as below:

ixHssAccCodeletMain (operationType, portMode, verifyMode)

Where operationType:

1 = Packetised Service Only.

2 = Channelised Service Only.

3 = Packetised Service and Channelised Services.

Where portMode:

1 = HSS Port 0 Only.

2 = HSS Port 1 Only.

3 = HSS Port 0 and 1.

Where verifyMode:

1 = codelet verifies traffic received in hardware loopback mode.

2 = codelet does not verify traffic received in hardware loopback mode.

4.3.2 HSS Codelet in Linux

The ixHssAccCodeletMain() is a single point of execution for the HssAcc codelet. The operation selected will be executed when user issues '`insmod`' in command prompt.

Usage:

```
insmod csr_codelets_hssAcc.o operationType=(a) portMode=(b)
verifyMode=(c)
```

Where a:

1 = Packetised Service Only.

2 = Channelised Service Only.

3 = Packetised Service and Channelised Services.

Where b:

1 = HSS Port 0 Only.

2 = HSS Port 1 Only.

3 = HSS Port 0 and 1.

Where c:

1 = codelet verifies traffic received in hardware loopback mode.

2 = codelet does not verify traffic received in hardware loopback mode.

4.3.3 HSS Codelet in Windows* CE .NET

1. To build the a codelet, browse to the specific codelet you want to build in the Catalog window. All the codelets are list under ‘BSPs’>‘INTEL IXDP425: ARMV4I’>‘IXP4XX Processor Variants’>‘IXP425’.
2. Select the codelet you wish to build. Right-click and select ‘Add to Platform’ from the drop-down menu.
3. Build the platform.
4. Once you download the platform you can run the codelet by using ‘Target’ and select ‘Run Programs...’
5. Select the IxHssAccCodelet.exe codelet from the list.
6. Click ‘Run’.
7. The codelet uses SerConsole and so the menus will be printed-out and input accepted from UART ‘COM1’ as follows:

**** HSS Codelet*****

```
1 = Packetised Service Only.
2 = Channelised Service Only.
3 = Packetised Service and Channelised Services.
Enter operationType:
1 = HSS Port 0 Only.
2 = HSS Port 1 Only.
3 = HSS Port 0 and 1.
Enter portMode:
1 = codelet verifies traffic received in hardware loopback mode.
2 = codelet does not verify traffic received in hardware loopback
mode.
Enter Verify mode:
```

4.4 DMA Codelet

There are two ways to run the DMA codelet: The first is to run the codelet main function that displays the transfer run in an endless loop; the second is to initialize the DmaAcc codelet and execute DMA transfer using the ixDmaAccCodeletTestPerform() function for various DMA transfer modes, addressing modes, and transfer widths. The block size used in this codelet are 8, 1024, 16384, 32768, 65528 bytes. For each DMA configuration, the performance is measured and the average rate (in Mbps) is displayed.

4.4.1 DMA Codelet in VxWorks*

Load the DMA codelet module from your target server console as follows

```
ld < Codelet_DMATest.out
```

Call the main function from your target server console as follows:

```
ixDmaAccCodeletMain()
```

- Once the function is executed, the codelet will display the results
- The formulae to calculate the rate is:

Rate (in Mbps) = ((length * 8) / (ticks / 66))

4.4.2 DMA Codelet in Linux

Load the DMA codelet using the '`insmod`' command at the command prompt, as follows:

```
insmod ixp400_codelets_dmaAcc.o
```

- Once the function is executed, the codelet will display the results
- The formulae to calculate the rate is:
  Rate (in Mbps) = ((length * 8) / (ticks / 66))

The API ixDmaAccCodeletTestPerform (transfer length, transfer mode, address mode, transfer width) allows the user to perform a DMA transfer of block size 0 to 65535 bytes between two locations in the SRAM. The user can specify any combination of the following modes.

DMA Transfer Modes

1. Copy
2. Copy and Clear Source
3. Copy with Bytes Swap
4. Copy with Bytes Reversed

DMA Addressing Modes

1. Incremental Source to Incremental Destination Addresses
2. Fixed Source to Incremental Destination Addresses
3. Incremental Source to Fixed Destination Addresses

DMA Transfer Widths

1. 32-bit Transfer

2. 16-bit Transfer
3. 8-bit Transfer
4. Burst Transfer

The user must initialize the system with ixDmaAccCodeletInit prior to calling the function ixDmaAccCodeletiTestPerform.

4.5 USB Codelet

4.5.1 USB Codelet in VxWorks*

In VxWorks, the USB codelet is executed by calling the ixUSBRNDISStart function. No parameters are needed to be passed in. The start function first loads and initializes the device driver. This function then loads the device driver into the MUX, then calls muxDevStart(). The muxDevStart starts the device that has already been initialized and handles registering the driver's interrupt service routine and anything else necessary to handle receiving and transmitting. The function finally assigns a default IP address to the RNDIS interface.

Usage:

```
ld < codelets_usbTest.out
```

To run the codelet:

```
ixUSBRNDISStart()
ifAddrSet "usb0", "x.x.x.x" (Where x.x.x.x is the desired ip
address of the interface)
```

4.5.2 USB Codelet in Linux

For the USB codelet in Linux, the rndisInit() function serves as a single point of execution for the USB codelet. The operation selected will be executed when the user issues the '`insmod`' command at the command prompt, as follows:

```
insmod ixp400_codelets_usb.o
ifconfig usb0 x.x.x.x (Where x.x.x.x is the desired ip address of
the interface)
```

4.6 ATM Codelet

In addition to the basic ATM functionality listed below, the ATM codelet also allows the user to execute OAM ping either in UTOPIA or Software Loopback mode.

OAM ping in UTOPIA Loopback mode performs the following sequence in an endless loop:

- Send AAL packets
- Display the transmit and receive statistics
- Perform OAM Ping F4 and F5 (ETE and Segment) and display OAM statistics

OAM ping in Software Loopback performs the following sequence in a continuous loop:

- Display the transmit and receive statistics
- OAM Ping F4 and F5 (ETE and Segment) and display OAM Statistics

4.6.1 Using ATM Codelet in VxWorks*

In VxWorks, the ixAtmCodeletMain() function is used as a single point of execution for the ATM codelet. This function allows the user to enter selections for different type of modes and AAL type. In all modes, the transmit and receive statistics are displayed every 15 seconds.

Usage :

```
ixAtmCodeletMain (modeType, aalType)
```

modeType:

0 = UTOPIA Loopback Mode

1 = Software Loopback Mode

2 = Remote Loopback Mode

3 = F4 and F5 cells OAM Ping in UTOPIA Loopback mode

4 = F4 and F5 cells OAM Ping in Software Loopback mode

aalType:

1 = AAL5

2 = AAL0_48

3 = AAL0_52

4.6.2 Using ATM Codelet in Linux

The ixAtmCodeletMain() function serves as a single entry point for execution, and also allows users to execute OAM ping. Similarly, all modes display the transmit and receive statistics every 15 seconds.

When the user issues the '`insmod`' command, the ATM codelet object will begin to send AAL packets and display the transmit and receive statistics every 15 seconds.

Note: Inability to '`rmmod`' (or the inability to enter anything in the command line) may be due to:

a. A carriage return without any error message, indicating that the task is sending AAL packets.

b. A failure to kill the thread that is still transmitting AAL packets.

This function is executed when the user issues '`insmod`' at the command prompt.

Usage :

```
insmod ixp400_codelets_atm.o modeType=<x> aalType=<y>
```

Where x:

0 = UTOPIA Loopback Mode

1 = Software Loopback Mode

2 = Remote Loopback Mode

3 = F4 and F5 cells OAM Ping in UTOPIA Loopback mode

4 = F4 and F5 cells OAM Ping in Software Loopback mode

Where y:

1 = AAL5

2 = AAL0_48

3 = AAL0_52

This page is intentionally left blank.