ICP2432 User's Guide for DIGITAL UNIX

DC 900-1513C

Simpact, Inc.
9210 Sky Park Court
San Diego, CA 92123
September 1999

Simpact, Inc.
9210 Sky Park Court
San Diego, CA 92123
(858) 565-1865

ICP2432 User's Guide for DIGITAL UNIX
© 1998–1999 Simpact, Inc. All rights reserved
Printed in the United States of America

Contents

List of Figures

List of Tables

Preface

Purpose of Document

This document describes how to use the ICP2432 intelligent communications processor in a peripheral component interconnect (PCI) bus computer running the DIGITAL UNIX operating system.

Intended Audience

You should have a working knowledge of UNIX STREAMS and know how to configure and build a UNIX kernel.

Required Equipment

You must have an ICP2432 and a host system with a PCIbus.

Organization of Document

Chapter 1 is an overview of the ICP2432 STREAMS driver.

Chapter 2 describes the software installation.

Chapter 3 describes the three ICP2432 STREAMS driver modes.

Chapter 4 describes the STREAMS programmer interface supported by the driver.

Chapter 5 describes the ICP packet formats.

Chapter 6 describes the download utility program using the ICP2432 STREAMS driver.

Appendix A describes the message interface.

References

Programmer’s Guide: STREAMS in the DIGITAL UNIX Documentation Library

Revision History

The revision history of the *ICP2432 User’s Guide for DIGITAL UNIX*, Simpact document DC 900-1513C, is recorded below:

Document Revision	Release Date	Description
DC 900-1513A	February 1998	Original release
DC 900-1513B	June 1998	File name changes
DC 900-1513C	September 1999	Added ICP_Option

Customer Support

If you are having trouble with any Simpact product, call us at 1-800-275-3889 Monday through Friday between 8 a.m. and 5 p.m. Pacific time.

You can also fax your questions to us at (858)560-2838 or (858)560-2837 any time. Please include a cover sheet addressed to “Customer Service.”

We are always interested in suggestions for improving our products. You can use the report form in the back of this manual to send us your recommendations.

Chapter 1 Overview

The ICP2432 STREAMS driver has two interfaces. One interface transports data between processes in user space and the driver in kernel space using the UNIX STREAMS environment. The other interface transports data between the driver and the ICP across the PCIbus.

STREAMS is a collection of system calls, kernel resources, and kernel utility routines that can create, use, and dismantle a STREAM. A STREAM is made up of a Stream Head, optional modules, and a driver (see Figure 1–1). The Stream Head is provided by the UNIX STREAMS environment and is responsible for transferring data between user space and kernel space. If necessary, the Stream Head creates a message or messages to be sent downstream to initiate an action by the driver.

Communication between the host and the ICP2432 is accomplished by means of DMA transfers across the PCIbus with the ICP as the PCIbus Master.

The user application interface is provided by the Stream Head. The interface consists of the following set of system calls:

open	open a STREAM
close	close a STREAM
read	read data from a STREAM
write	write data to a STREAM
ioctl	perform special I/O control functions

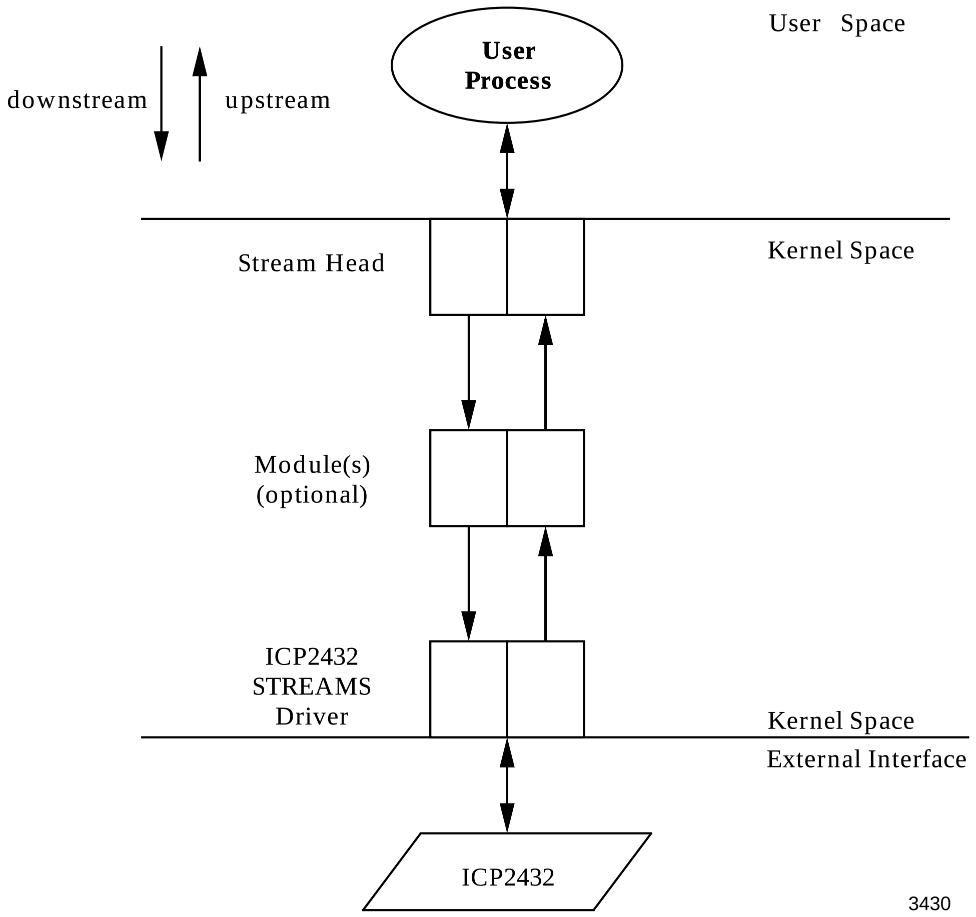

Figure 1–1: ICP2432 STREAMS Configuration

getmsg	get a message from a STREAM
putmsg	put a control message onto a STREAM (not supported)
poll	poll a STREAM and notify an application of selected events occurring on a STREAM

The ICP2432 STREAMS driver can be classified as a *raw-mode* driver. It does not require any special header or data format for the messages it processes. It does not process the data it finds in a STREAMS message in any way. Any processing of the data found within a STREAMS message is performed by modules that may be pushed into the STREAM between the Stream Head and the driver, or by software on the ICP.

Chapter 2 Software Installation

This chapter describes Simpact's ICP2432 software installation procedure for DIGITAL UNIX.

Caution

Remember that installing new software overwrites the previous software.

2.1 ICP2432 Software Installation Procedure

Step 1:

Verify that you have installed one or more ICP2432 boards in your computer, as described in the *ICP2432 Hardware Installation Guide.*

Step 2:

Insert the software distribution media into the appropriate drive.

Step 3:

Use the **`tar x`** command as shown on the next page to retrieve the files. You might want to include the **`v`** option to display the file names as they are extracted. Some systems require that you use the **`f`** option to identify the peripheral device being used.

Here are two examples of the **tar** command (the device name on your system might be different):

```
tar xv
tar xvf /dev/rmt0h
```

The files are copied from the distribution media into a directory called `freeway`.

2.2 ICP2432 STREAMS Driver Installation Procedure

The following subsections describe how to install the ICP2432 STREAMS driver using the **icpsetup** script, rebuild the UNIX kernel, reboot the machine, and create the device files.

2.2.1 Install the STREAMS Driver using the icpsetup Script

The **icpsetup** script uses the **setld** command to install the ICP2432 STREAMS driver.

Step 1:
Log in as root.

Step 2:
Change directory to `freeway/client/axp_du_emb/bin` and run the **icpsetup** script as follows:

```
# cd freeway/client/axp_du_emb/bin
# icpsetup
```

The computer displays:

```
*** Enter subset selections ***

The following subsets are mandatory and will be installed automatically
unless you choose to exit without installing any subsets:

      * Simpact PCI ICP2432 Driver Binary Kit
```

```
You may choose one of the following options:

     1) ALL of the above
     2) CANCEL selections and redisplay menus
     3) EXIT without installing any subsets

Enter your choices or press RETURN to redisplay menus.

Choices (for example, 1 2 4-6): 1
```

Step 3:

Enter **1** to select all of the above.

The computer displays:

```
You are installing the following mandatory subsets:

      Simpact PCI ICP2432 Driver Binary Kit

You are installing the following optional subsets:

Is this correct? (y/n): y
```

Step 4:

Enter **y** to select the product kit. It is not necessary to install any optional subsets.

The computer displays:

```
1 subset(s) will be installed.

Loading 1 of 1 subset(s)....

Simpact PCI ICP2432 Driver Binary Kit
   Copying from ../kit (disk)
   Verifying

1 of 1 subset(s) installed successfully.

The SIMPACT PCI ICP2432 Driver has been successfully installed.
```

Step 5:

The next step is to configure the product kit. When prompted, enter the number of ICP2432 boards and the number of nodes in your system. (The valid range for the number of nodes is 4 through 64.) These numbers are used to make the device file.

The computer displays:

```
...
Configuring "Simpact PCI ICP2432 Driver Binary Kit" (ICPPCIBASE110)

Please answer the following questions.
* How many ICP2432 cards are in your system?[1] 2
* How many nodes do you use?[16] 8

icpsetup done.
Please make the kernel using doconfig command.
```

2.2.2 Rebuild the UNIX Kernel

After the product kit is installed, you must rebuild the UNIX kernel so that it includes the new device driver.

Step 1:

Enter the following at the prompt, where *SYSNAME* is the system name specified in uppercase:

```
# doconfig -c SYSNAME
```

Note

It is not necessary to modify the configuration file when prompted by **doconfig**.

The computer displays:

```
*** KERNEL CONFIGURATION AND BUILD PROCEDURE ***

Saving /sys/conf/SYSNAME as /sys/conf/SYSNAME.bck

Do you want to edit the configuration file? (y/n) [n]: n

*** PERFORMING KERNEL BUILD ***
        Working....Wed Feb 18 10:46:10 PST 1998
        Working....Wed Feb 18 10:48:12 PST 1998

The new kernel is /sys/SYSNAME/vmunix
```

Step 2:

The **doconfig** command builds the new UNIX kernel, putting the new image in the file /sys/*SYSNAME*/vmunix. Before making this file the new system image (using the **cp** command), it is a good idea to save the existing kernel image, which is what the first instruction shown on the next page does. After the copy is performed, the system is ready to be rebooted with the new kernel image.

Enter the following at the prompt:

```
# mv /vmunix /vmunix.sav
# cp /sys/SYSNAME/vmunix /vmunix
```

2.2.3 Reboot the Machine

Either the **reboot** or **shutdown** command can be used to reboot the system. The **reboot** command causes the machine to be rebooted immediately, whereas **shutdown** is friendlier, giving any users that are logged on an opportunity to log out before the system reboots. The following command waits five minutes before performing a controlled shutdown and reboot:

```
# shutdown -r +5 "Rebooting to install ICP2432 device driver"
```

The **-r** option causes an automatic reboot after the shutdown.

2.2.4 Create the Device Files

After the system is rebooted, you must create the device files in the `/dev` directory in order to access the ICP2432. This is done by executing the `mkdev` script file found in the `freeway/client/axp_du_emb/bin` directory. This script file requires one parameter, the major device number for the ICP2432.

Step 1:

Log in as root.

Step 2:

Change directory to `freeway/client/axp_du_emb/bin`:

```
# cd freeway/client/axp_du_emb/bin
```

Step 3:

To create the device file, this script uses the `strsetup` command to get the device major number.

Enter the following at the prompt:

```
# mkdev
```

For example:

```
# mkdev
ICP make device file utility VI-100-0483: ICPUTLDU 1.2-0
Icp device major number is 64.
/dev/icp0/0
/dev/icp0/1
/dev/icp0/2
/dev/icp0/3
/dev/icp1/0
/dev/icp1/1
/dev/icp1/2
/dev/icp1/3
```

The number of ICPs and nodes are set in the `freeway/client/axp_du_emb/bin/sysconfigtab` file as follows:

```
Icp_Num_Icp  = 2
Icp_Num_Node = 4
```

For more information about `Icp_Num_Icp` and `Icp_Num_Node`, see Section 4.1 on page 27.

When the script file completes, the new directory **/dev/icp0** contains the device files.

2.2.5 How to Modify the Driver's Parameters

After installation, the driver has the following parameters:

Icp_Num_Icp	Maximum number of ICP boards the driver can use.
Icp_Num_Node	Maximum number of nodes the driver can use for one ICP board.
Icp_SingleStep	When this value equals 1, the driver is in ICP SingleStep debugger mode. The default value is 0.
ICP_Option	Sets the device number (`/dev/icp0`, `/dev/icp1`, `/dev/icp2`, and so on) for the specified bus number and slot number. For more information, see Section 2.2.6 on page 23.

For more information about `Icp_Num_Icp` and `Icp_Num_Node`, see Section 4.1 on page 27.

The following steps describe how to change the driver parameters (`Icp_Num_Icp`, `Icp_Num_Node`, `Icp_SingleStep`, and `ICP_Option`).

Step 1:

Change directory to `freeway/client/axp_du_emb/bin`:

```
# cd freeway/client/axp_du_emb/bin
```

Step 2:

Edit the `sysconfigtab` file and change the necessary parameter values (`Icp_Num_Icp`, `Icp_Num_Node`, or `Icp_SingleStep`).

Step 3:

Run the `sysconfigdb` utility to configure the driver's attributes:

```
# sysconfigdb -u -f sysconfigtab icp
```

Step 4:

Reboot the system. After reboot, the new parameters will be in effect:

```
# reboot
```

Step 5:

If you changed the `Icp_Num_Icp` or `Icp_Num_Node` parameter, run the `mkdev` utility. See Section 2.2.4 on page 20 for an explanation of the major device number. If you changed the `Icp_SingleStep` or `ICP_Option` parameter, you do not need to run `mkdev`.

```
# mkdev
```

If you wish, you can check the value of each parameter in the `sysconfigdb` utility.

```
# sysconfigdb -l icp
icp:
Module_Config_Name = icp
Icp_Num_Icp = 3
Icp_Num_Node = 5
Icp_SingleStep = 0
PCI_Option = ........
...........................
```

2.2.6 ICP_Option Parameter

The driver matches the ICP device number (*n* of `/dev/icpn`) to ICP boards using the `ICP_Option` parameter. When one ICP board is removed, the device numbers of the other devices may be changed. The `ICP_Option` parameter prevents the system from changing the device number as shown below for the `sysconfigtab` file.

```
ICP_Option = Icp - 1, Pci - 2000, Slot - 7
```

The `ICP_Option` has three factors of Icp, Pci, and Slot as described below:

Icp	Defined device number (*n* of `/dev/icpn`)
Pci	PCI bus number (this is the bus number multiplied by 1000)
Slot	Slot number in the PCI bus.

In the example above, the device number is 1 for the ICP device installed in bus 2 and slot 7.

You can display the bus and slot values on the console by using the following command. The actual display will differ depending on your system. The following example is for an AlphaServer2100A system.

```
P00>>>show config
Digital Equipment Corporation
AlphaServer 2100A 4/275

SRM Console V5.3-10 VMS PALcode V5.56-7, OSF PALcode V1.45-12

Component Status Module ID
CPU 0 P B2024-AA DECchip (tm) 21064A-2
Memory 0 P B2022-DA 128 MB
I/O 24283-01
dva0.0.0.1000.0 RX26/RX23

Slot Option Hose 0, Bus 0, PCI
2 Intel 82375 Bridge to Bus 1, EISA
3 DECchip 21050-AA Bridge to Bus 2, PCI
8 000412A1
9 S3 Trio64/Trio32
```

```
Slot Option Hose 0, Bus 1, EISA

Slot Option Hose 0, Bus 2, PCI
1 NCR 53C810 pka0.7.0.2001.0 SCSI Bus ID 7
dka0.0.0.2001.0 RZ26L
dka100.1.0.2001.0 RZ26L
dka500.5.0.2001.0 RRD45
mka600.6.0.2001.0 TLZ07
6 DECchip 21040-AA ewa0.0.0.2006.0 00-00-F8-22-32-DF
7 000412A1
```

The ICP2432 has the value `00`*xx*`12A1` for the option (*xx* is 02, 04, 08, 12, 14, or 18). The two ICP2432s are installed in the system. One is at bus 0, slot 8, and the other is at bus 2, slot 7.

You can also confirm your setting on the console. If your setting is correct, you will see a message similar to the following:

```
Simpact ICP2432 Driver VI-100-0482:DRV2432DU 1.2-0 Aug 12 1999
icp1 at pci2000 slot 7.
icp1: Simpact ICP2432 4 port card. 4
```

Note

The `Icp_Num_Icp` parameter must be set to a larger number than the device number `Icp` of `ICP_Option`. For example, when one ICP device is installed in the system and the device number is set to 1, `Icp_Num_Icp` must be set to 2 (even though one ICP device is in the system).

Chapter 3 ICP2432 STREAMS Driver Modes

The ICP2432 STREAMS driver functions in one of three operating modes: `reset`, `download`, and `exec`.

3.1 Reset Mode

The ICP enters `reset` mode upon the following events:

- The PCIbus `reset` signal is asserted (on power-up and system boot)
- The ICP receives a `reset` command from the ICP2432 STREAMS driver

In `reset` mode, the ICP executes code from the on-board ROM.

3.2 Download Mode

The ICP2432 STREAMS driver enters `download` mode when the ICP accepts the "ready to download" command sent by the driver after issuing a `reset` command. In this mode, the driver only accepts `reset`, `download write`, and `init procedure` requests. The driver enters `exec` mode when it receives the response of the `init` procedure request from the ICP2432.

3.3 Exec Mode

In `exec` mode, the ICP2432 STREAMS driver provides a communication channel between processes executing on the host and the ICP. This is the normal operating mode of the driver after the ICP has been downloaded. The driver will not accept `download write` and `init procedure` requests while in `exec` mode. It will, however, accept `reset` requests that will place the driver into `reset` mode.

3.4 Downloading Software Images to the ICP

Downloading software images to the ICP2432 requires putting the ICP into `reset` mode, then initiating the proper sequence of `ioctl` commands to go from `reset` mode to `exec` mode. The host download program issues an `INIT_ICP` `ioctl` to reset the target ICP and place it into `download` mode. When the `INIT_ICP` `ioctl` completes, the host download program can begin to download software images to the ICP. The `ICPDL` `ioctl` is provided to transfer each block of the file(s) to be downloaded from the host onto the ICP at a specified address. A separate `ICPDL` `ioctl` call is required for each block to be downloaded. When all software images have been written to the ICP2432, the host download program issues an `INIT_PROC` `ioctl` to start the execution of the software on the ICP. See Section 4.5 on page 34 for more information on `ioctl`s.

The product kit includes the `icpload` program for the download utility. After the `icpload` program completes, the ICP2432 is in `exec` mode. See Chapter 6 on page 43 for more information.

Chapter 4 Programmer Interface

The ICP2432 STREAMS driver supports a standard interface that provides `open`, `close`, `read`, `write`, `getmsg`, `poll`, and `ioctl` system calls through the Stream Head. The `putmsg` system call is not supported because the device driver has no service interface protocol. The driver is accessed from the application level using system calls to initiate action by the Stream Head. The Stream Head then sends the proper control message to the driver.

A module within the STREAM can communicate with the device driver by creating the proper STREAMS message and sending it downstream to the driver. See Appendix A for more information on the message interface to the ICP2432 STREAMS driver.

4.1 Naming Convention

The naming convention for the `/dev` entries is as follows:

`/dev/icp`*b*`/`*n*

where *b* is the ICP board number and *n* is the node number minus 3. The communication nodes are described in Section 4.4.1 on page 29. The entries in `/dev` were created during driver installation (refer to Step 5 on page 18 of Section 2.2.1).

The maximum number of boards in the system is defined by `Icp_Num_Icp` and the maximum number of communication nodes per ICP2432 is defined by `Icp_Num_Node`. `Icp_Num_Icp` and `Icp_Num_Node` are defined in `freeway/client/axp_du_emb/bin/sysconfigtab`. The valid range for `Icp_Num_Node` is 4 through 64. To modify `Icp_Num_Icp` and `Icp_Num_Node`, see Section 2.2.5 on page 21.

Each communication node is assigned a minor device number, beginning at zero and incrementing continually. The communication node numbers for each ICP2432 are defined beginning at 3.

For example, if `Icp_Num_Icp` = 2 and `Icp_Num_Node` = 17, the `/dev` directory contains the following entries:

```
/dev/icp0/0   =   ICP 0     node 3    minor device 0
/dev/icp0/1   =   ICP 0     node 4    minor device 1
/dev/icp0/2   =   ICP 0     node 5    minor device 2
          .   .   .
          .   .   .
/dev/icp0/16  =   ICP 0     node 19   minor device 16
/dev/icp1/0   =   ICP 0     node 3    minor device 17
/dev/icp1/1   =   ICP 0     node 4    minor device 18
          .   .   .
          .   .   .
/dev/icp1/16  =   ICP 0     node 19   minor device 33
```

4.2 Open

A user process must issue an `open` system call to the device representing the node number before performing any I/O requests. The format of the `open` system call is as follows:

```
fdev = open("/dev/icpb/n", O_RDWR);
```

where `fdev` is the returned file descriptor of the STREAM, *b* is the ICP board number, and *n* is the node number minus 3.

4.3 Close

When an application has completed its I/O requests to a particular STREAM, it must issue a `close` system call to dismantle the STREAM. The format of the `close` system call is as follows:

```
close(fdev);
```

where `fdev` is the file descriptor for the STREAM (returned by `open`).

4.4 Data Transfer Functions

The Stream Head is responsible for the user interface in a STREAMS environment. All `write` requests come downstream to the driver as `M_DATA` messages from the Stream Head. The ICP2432 STREAMS driver forwards the information needed to perform a DMA transfer down to the ICP. `Read` requests do not travel downstream; instead they are held at the Stream Head. When the ICP completes a `read` request from the device driver (transfers the data from the ICP to host memory), the device driver sends the message upstream. These messages are queued at the Stream Head until a `read` request from the user is made on the stream. When a `read` request is made, the message at the top of the queue is passed to the user.

4.4.1 Node Numbers

Communication between a host process and a task on the ICP2432 can be considered interprocess communication and some method of determining the source and destination of a message is required. The concept of node numbers was designed for this purpose. For each ICP2432, node numbers 1 through `Icp_Num_Node` plus 3 are used to identify a path between a particular host process and a task on the ICP.

Node 1 is the standard node for writing most command and data packets to the ICP2432. Since buffering limitations on the ICP2432 may cause delays in completion of writes to the ICP2432, node 2 is typically reserved for writing command packets to terminate application sessions with a port or to terminate an application's use of node for reading packets from the ICP2432. Only node 1 or 2 is used to write to the ICP2432. The other nodes are used to read from the ICP2432.

In a STREAMS environment, a minor device is assigned to each of the node numbers used for reading, so that each of these node numbers can be accessed through a separate STREAM. Each ICP2432 minor device identifies a particular ICP device and a node number used for reading from that device. Only nodes 1 and 2, used for writes, are shared with all minor devices on the same ICP.

When the ICP completes the request, it returns the node number to the device driver, which uses the node number to determine which STREAM is associated with the request.

A particular host process may open more than one minor device in order to communicate simultaneously with multiple nodes on one or more ICP2432 devices. However, if two or more host processes attempt to open the same minor device, they will share the same STREAM (and node number). Data received from the ICP2432 on a particular node number is sent upstream by the driver, but cannot be directed to a particular process by the Stream Head. The driver cannot guarantee that a particular process will receive the messages intended for it. To insure data integrity between host processes and ICP tasks, no two host processes should ever use the same minor device (STREAM) at the same time.

4.4.2 I/O Types

The ICP2432 STREAMS driver supports both blocking and non-blocking I/O. The default is blocking I/O. With blocking I/O, the application blocks (sleeps) at the Stream Head until its I/O request can be satisfied. To use non-blocking I/O, the application must issue an `fcntl` request to set the `O_NDELAY` flag for the STREAM. This enables a non-blocking I/O session with the Stream Head. Any I/O requests that cannot be satisfied return an error (-1) and set `errno` to `EAGAIN`. The request must be retried until it completes successfully.

A good completion for a `write` request means that the Stream Head was able to send a message downstream. It does not imply that the data was successfully sent to the ICP2432, only that it was sent downstream by the Stream Head. A good completion for a `read` request means that a message (or messages) previously sent upstream by the driver was available at the Stream Head for return to the application.

4.4.3 Error Conditions

Due to the asynchronous nature of STREAMS, a fatal error may occur when transferring data between the ICP2432 and the driver after the user request has completed successfully. If such an error occurs, an `M_ERROR` message is sent upstream and the STREAM is effectively shut down. Only `poll` and `close` system calls can then be made on the STREAM. All others are rejected by the Stream Head and `errno` is set to indicate the cause of the error. Any system calls in progress at the time of the error are subject to failure or hanging depending on the nature of the call.

Once a fatal error has been detected, the user must `close` the STREAM and then reopen it with an `open` call to restart operations.

4.4.4 Read

In a STREAMS environment, the user must select the type of `read` desired. This typically depends on the software running on the ICP2432 and the programming style of the application. There are three types of reads:

RNORM	Byte-stream mode
RMSGN	Read message, non-discard mode
RMSGD	Read message, discard mode

The default is byte-stream mode (RNORM). This mode can be used when the size of the buffer to be read is known. The `read` request does not complete until the number of bytes requested have been read from the Stream Head. Non-discard mode (RMSGN) is the most common form used with ICP communications because the size of the buffer to be read is not always known. With this mode, the `read` request completes when the user's buffer is full or on a message boundary. Discard mode (RMSGD) should not be used because it may cause a loss of data.

To set the read options for a particular STREAM, an application can issue an I_SRDOPT ioctl. This sets the read options for the STREAM on which the ioctl is made. The format of the I_SRDOPT ioctl is as follows:

```
result = ioctl(fdev, I_SRDOPT, read_option);
```

where fdev is the file descriptor of the STREAM and read_option is the type of read (RNORM, RMSGN, or RMSGD).

I_SRDOPT, RNORM, RMSGN, and RMSGD are defined in the system include file stropts.h.

There are two system calls that can be made to read data from the Stream Head: read and getmsg.

The format of the read system call is as follows:

```
result = read(fd, p_rbuf, count);
```

where fd is the stream file descriptor, p_rbuf is the pointer to the buffer where data will be placed, and count is the number of bytes requested (RNORM) or the maximum number of bytes that can be accepted (RMSGN).

Based upon the setting of the STREAMS read option discussed previously, the read completes when the count has been satisfied (RNORM or RMSGN) or at a message boundary (RMSGN). A read system call only returns data found in messages of type M_DATA at the Stream Head.

The format of the getmsg system call is as follows:

```
&result = getmsg(fd, ctlptr, dataptr, flags);
```

where fd is the stream file descriptor, ctlptr is the pointer to the control part of message (always NULL), dataptr is the pointer to the data part of message, and flags should be set to 0 to specify normal and high-priority message reception.

Getmsg receives a message from the Stream Head. The ctlptr parameter will be NULL because the ICP2432 STREAMS driver defines no special control message interface.

Data is returned in the buffer pointed to by the `dataptr` parameter. `Flags` should be set to 0 to receive both normal and priority messages. Unlike a `read` system call, `getmsg` retrieves all message types. The only message that the driver can send upstream other than an `M_DATA` message is an `M_ERROR` message. This is sent upstream if an error occurs during an I/O operation between the driver and the ICP. Setting `flags` to 0 allows reception of the priority `M_ERROR` message.

For more information, refer to the manual pages for read(2) and getmsg(2) in the UNIX documentation.

4.4.5 Write

The `write` system call sends data downstream to the ICP2432 STREAMS driver for transfer to the ICP. The format of the `write` system call is as follows:

```
result = write(fd, p_wbuf, count);
```

where `fd` is the stream file descriptor, `p_wbuf` is the pointer to data to be sent, and `count` is the number of bytes of data to be sent.

The driver does not process any of the data found in the `write` buffer before sending it to the ICP2432. The `write` buffer should have the format expected by the software on the ICP.

4.4.6 Error Codes

A return code of -1 from a `read`, `write`, or `getmsg` call indicates that an error has occurred. The system error code variable `errno` indicates the type of error. The following error codes are returned by the ICP2432 STREAMS driver:

EFAIL	The request is an invalid sequence. The target ICP must be downloaded.
EINVAL	An invalid parameter was found in the `write` system call.

EIO	An I/O error has occurred during a read or write system call.
ENOBUFFS	No buffers are available to complete the requested operation.
EOPNOTSUPP	The requested operation is not supported by the target ICP.

4.5 Ioctl Commands

The ICP2432 STREAMS driver supports three ioctl functions. All are used during board initialization to reset and download the ICP2432. The icpload program included in the product kit uses all of the functions described in the following subsections.

The most common form of an ioctl command comes from a system call made by an application program. The format of an ioctl command from an application is as follows:

```
ret = ioctl(fdev, I_STR, p_strioctl)
```

where fdev is the file descriptor of the target ICP2432, I_STR indicates that the command is a STREAMS ioctl, and p_strioctl is a pointer to a strioctl structure. The standard STREAMS strioctl structure is defined in <sys/stropts.h> as follows:

```
struct strioctl
{
       int   ic_cmd;     /* Ioctl command          */
       int   ic_timout;  /* ACK/NAK timeout length  */
       int   ic_len;     /* Length of data included */
       char  *ic_dp;     /* Pointer to ioctl data   */
};
```

An ioctl command appears on the STREAM as an M_IOCTL message.

An M_IOCTL message can be generated by the Stream Head as a result of a user application request or by an upstream module that wants to use the ioctl functions of the ICP2432 STREAMS driver. The data portion of each M_IOCTL message received by the driver must conform to the iocblk structure as defined by STREAMS in the file <sys/stream.h>. If an ioctl request is made by a user application, the Stream Head places the user's data in the iocblk structure. Applications use the strioctl structure when making an ioctl request. M_IOCTL requests coming from upstream modules must place their requests into the proper iocblk format before sending them downstream to the driver. For more information on these structures, see the *Programmer's Guide: STREAMS* in the DIGITAL UNIX Documentation Library.

Each of the ioctl functions described in this section causes either an M_IOCACK (successful) or M_IOCNAK (unsuccessful) message to be sent upstream from the driver. At the user level, the result is a good (0) or an error (-1) return on the ioctl call. If an error is returned, check the system error code variable errno for more information on the error.

The following subsections describe the ioctl functions performed by the driver. Each description is followed by an example of a system call from a user application.

4.5.1 INIT_ICP

When the ICP2432 STREAMS driver receives the INIT_ICP ioctl command, it resets the target ICP2432. This causes the ICP to perform its on-board self-test. If the self-test is successful, the ICP is placed into download mode by the driver and an M_IOCACK message is sent upstream. An unsuccessful self-test results in an M_IOCNAK message being sent upstream with the error code found in the M_IOCNAK message set appropriately. There is no data associated with this ioctl.

```
strioctl.ic_cmd     = INIT_ICP;
strioctl.ic_timout  = 0;
strioctl.ic_len     = 0;
strioctl.ic_dp      = (char *)NULL;
result = ioctl(fdev, I_STR, &strioctl);
```

The default timeout value of the ioctl is 15 seconds. Some ICP boards require more time for initialization. If you get the System call timed out (ETIME) error, increase the value of strioctl.ic_timeout to more than 15 seconds.

4.5.2 ICPDL

The `ICPDL ioctl` command sends blocks of data to be downloaded to the target ICP2432. The data portion of the message block should have the following structure:

```
typedef struct icp_download
   {
      bit32 io_dl_addr;       /* ICP address to download
                                 this block */
      bit8  data[8192];       /* Data to download */
   }  ICP_DOWNLOAD;
```

The data block is loaded onto the ICP2432 at the `io_dl_addr` specified. Note that the maximum transfer size for this operation is 8192 bytes.

```
size_t count;
ICP_DOWNLOAD icpdl;

icpdl.io_dl_addr    = 0x801200;
count = fread(icpdl, data, sizeof(char), 8192, fp); /* Read data from
                                                       file */

strioctl.ic_cmd     = ICPDL;
strioctl.ic_timout  = 0;
strioctl.ic_len     = count + 4;           /* Size of data plus
                                              io_dl_addr field */
strioctl.ic_dp      = (char *)&icpdl;

result = ioctl(fdev, I_STR, &strioctl);
```

4.5.3 INIT_PROC

The `INIT_PROC` `ioctl` command specifies the ICP2432 address at which the code downloaded to the ICP should begin executing. The data portion is one longword (`ic_len` = 4) that must be set to the desired ICP2432 address. This command is sent to the target ICP which begins executing code at the specified address.

```
int initaddr = 0x818000;

strioctl.ic_cmd     = INIT_PROC;
strioctl.ic_timout  = 0;
strioctl.ic_len     = 4;
strioctl.ic_dp      = (char *)&initaddr;

result = ioctl(fdev, I_STR, &strioctl);
```

4.5.4 Error Codes

A return code of -1 from an `ioctl` call indicates that an error has occurred. The system error code variable indicates the type of error. `Errno` can be set by the ICP2432 STREAMS driver or by the UNIX system. The following error codes are returned by the driver.

EFAIL	The request is an invalid sequence. See Chapter 3 on page 25.
EINPROGRESS	The requested operation is already in progress.
EINVAL	An invalid parameter was found in the `ioctl` request.
EIO	An error has occurred during an I/O operation between the ICP and the driver.
EOPNOTSUPP	The requested operation is not supported by the target ICP.
ETIME	The ioctl function timed out. See Section 4.5.1 on page 35.
ETIMEDOUT	The driver has timed out waiting for a response from the target ICP.

Chapter

5

ICP Packet Formats

Simpact's Intelligent Communications Processor (ICP) board is a general-purpose serial data-link front-end processor. The ICP supports a variety of serial data-link protocols. Simpact packages each protocol in the form of downloadable ICP software with a protocol programmer's guide. The protocol programmer's guide describes the format of packets the application may write to or read from the ICP.

The ICP is used in Simpact's Freeway communications servers, and may also be used as an embedded front-end processor in compatible host computer equipment. The programmer's guide for each protocol assumes that Simpact's Freeway DLI application program interface is to be used. Although Simpact's DLI is available for use on DIGITAL UNIX computers, it currently supports access to Freeway ICPs only. An application on DIGITAL UNIX computers must instead use the ICP2432 STREAMS driver interface to access embedded ICP boards.

The packet format defined for the driver interface differs slightly from that defined for Simpact's DLI. This section describes the differences between these two formats.

5.1 DLI Packet Format

The data link interface (DLI) provides a session-level interface between a client application and the protocol software resident on the Freeway ICP. From the application's perspective, these packets consist of data buffers supplemented with the DLI optional arguments structure to provide the protocol-specific information required for Raw operation. Figure 5–1 shows the "C" definition of the DLI optional arguments structure.

```
typedef struct     _DLI_OPT_ARGS
{
   unsigned short  usFWPacketType;   /* Server's packet type */
   unsigned short  usFWCommand;      /* Server's command sent or received */
   unsigned short  usFWStatus;       /* Server's status of I/O operations */
   unsigned short  usICPClientID;    /* Old su_id */
   unsigned short  usICPServerID;    /* Old sp_id */
   unsigned short  usICPCommand;     /* ICP's command */
            short  iICPStatus;       /* ICP's command status */
   unsigned short  usICPParms[3];    /* ICP's extra parameters */
   unsigned short  usProtCommand;    /* Protocol command */
            short  iProtModifier;    /* Protocol command's modifier */
   unsigned short  usProtLinkID;     /* Protocol link ID */
   unsigned short  usProtCircuitID;  /* Protocol circuit ID */
   unsigned short  usProtSessionID;  /* Protocol session ID */
   unsigned short  usProtSequence;   /* Protocol sequence */
   unsigned short  usProtXParms[2];  /* Protocol extra parameters */
}  DLI_OPT_ARGS;
```

Figure 5–1: "C" Definition of DLI Optional Arguments Structure

5.2 Packet Format

The ICP2432 STREAMS driver interface provides a block-transfer interface between a client application and the protocol software resident on the embedded ICP. From the application's perspective, these packets consist of message blocks composed of a header structure followed by an optional data array. Figure 5–2 shows the "C" definition of this ICP packet structure.

When preparing a packet to write to the ICP, the application must initialize the `usICPCount` field with the size in bytes of the `PROT_HDR` structure (16) plus the size of the data array that follows it. After reading a packet from the driver, the application may compute the size of the data array that follows the `PROT_HDR` structure by subtracting 16 from the value of the `usICPCount` field in the `ICP_HDR` structure.

Note that the `ICP_HDR` structure is required to be in network byte-order (Big Endian). The `PROT_HDR` structure may be in the host computer's natural byte order, whether Big

```
typedef struct     _ICP_PACKET
{
   ICP_HDR         icp_hdr;          /* Network-ordered header */
   PROT_HDR        prot_hdr;         /* Host-ordered header */
   char            *data;            /* Variable length data array */
} ICP_PACKET;

typedef struct     _ICP_HDR
{
   unsigned short  usICPClientID;    /* Old su_id */
   unsigned short  usICPServerID;    /* Old sp_id */
   unsigned short  usICPCount;       /* Size of PROT_HDR plus data */
   unsigned short  usICPCommand;     /* ICP's command */
            short  iICPStatus;       /* ICP's command status */
   unsigned short  usICPParms[3];    /* ICP's extra parameters */
} ICP_HDR;

typedef struct     _PROT_HDR
{
   unsigned short  usProtCommand;    /* Protocol command */
            short  iProtModifier;    /* Protocol command's modifier */
   unsigned short  usProtLinkID;     /* Protocol link ID */
   unsigned short  usProtCircuitID;  /* Protocol circuit ID */
   unsigned short  usProtSessionID;  /* Protocol session ID */
   unsigned short  usProtSequence;   /* Protocol sequence */
   unsigned short  usProtXParms[2];  /* Protocol extra parameters */
} PROT_HDR;
```

Figure 5–2: "C" Definition of ICP Packet Structure

Endian or Little Endian. Each protocol programmer's guide describes how the application declares its natural byte order to the protocol software resident on the ICP.

5.3 DLI_OPT_ARGS and ICP_PACKET Structures Compared

The careful reader will note that the `ICP_PACKET` structure differs only slightly from the `DLI_OPT_ARGS` structure. The `ICP_PACKET` structure omits the three Freeway server header fields (`usFWPacketType`, `usFWCommand`, and `usFWStatus`) and adds one new field (`usICPCount`). See Table 5–1 for a comparison between the header fields in the `DLI_OPT_ARGS` and `ICP_PACKET` structures.

Table 5–1: Comparison of DLI_OPT_ARGS and ICP_PACKET Structures

DLI_OPT_ARGS field name	ICP_PACKET field name	Field Description
`usFWPacketType`	omitted	Server's packet type
`usFWCommand`	omitted	Server's command sent or received
`usFWStatus`	omitted	Server's status of I/O operations
`usICPClientID`	`icp_hdr.usICPClientID`	Old `su_id`
`usICPServerID`	`icp_hdr.usICPServerID`	Old `sp_id`
omitted	`icp_hdr.usICPCount`	Size of `PROT_HDR` plus data
`usICPCommand`	`icp_hdr.usICPCommand`	ICP's command
`iICPStatus`	`icp_hdr.iICPStatus`	ICP's command status
`usICPParms[0]`	`icp_hdr.usICPParms[0]`	ICP's extra parameter
`usICPParms[1]`	`icp_hdr.usICPParms[1]`	ICP's extra parameter
`usICPParms[2]`	`icp_hdr.usICPParms[2]`	ICP's extra parameter
`usProtCommand`	`prot_hdr.usProtCommand`	Protocol command
`iProtModifier`	`prot_hdr.iProtModifier`	Protocol command's modifier
`usProtLinkID`	`prot_hdr.usProtLinkID`	Protocol link ID
`usProtCircuitID`	`prot_hdr.usProtCircuitID`	Protocol circuit ID
`usProtSessionID`	`prot_hdr.usProtSessionID`	`usProtSessionID`
`usProtSequence`	`prot_hdr.usProtSequence`	`usProtSequence`
`usProtXParms[0]`	`prot_hdr.usProtXParms[0]`	Protocol extra parameter
`usProtXParms[1]`	`prot_hdr.usProtXParms[1]`	Protocol extra parameter
`omitted` [a]	data	Data array

[a] An application using Simpact's DLI specifies data separately from the DLI_OPT_ARGS structure.

Chapter 6 Downloading the ICP

This chapter describes the ICP download program, `icpload`, that uses most of the `ioctl` commands described in Chapter 4. The `icpload` utility is part of the product kit and is in the `freeway/client/axp_du_emb/bin` directory.

NAME

`icpload`

SYNTAX

`icpload device command_file`

For example:

```
icpload /dev/icp0/3 /freeway/boot/awsload
```

DESCRIPTION

The program `icpload` is used to reset an ICP2432 and then download that ICP2432 with the specified memory images. Upon conclusion of the download, the on-board initialization procedure is executed to start the run-time image.

`icpload` requires the ICP device name as the first parameter. The `device` parameter specifies the name of the target ICP.

The second parameter is the command file used by `icpload`. This file specifies the memory images and associated load initialization and addresses. The command file consists of any number of lines, one command per line, with the following two possible command types:

A `LOAD` command line has the following format:

```
LOAD pathname load_address
```

`LOAD` identifies the command type, `pathname` is the pathname of the executable image to be loaded, and `load_address` is the on-board starting load address in hexadecimal.

An `INIT` command line has the following format:

```
INIT execution_address
```

`INIT` identifies the command type and `execution_address` is the on-board execution address in hexadecimal.

Parameters on a command line must be separated with one or more spaces or with tabs.

Upon successful completion, `icpload` returns a value of zero. Otherwise, a value is returned that represents the error encountered.

The contents of a typical command file (for the AWS protocol) are:

```
LOAD /usr/local/freeway/icpcode/icp2432/osimpact/xio_2432.mem     801200
LOAD /usr/local/freeway/icpcode/icp2432/protocols/aws_fw_2432.mem 818000
INIT                                                              818000
```

This loads the OS/Impact operating system (`xio_2432.mem`) and the AWS protocol software (`aws_fw_2432.mem`) onto the ICP2432, and executes the protocol software's INIT procedure.

If the path name of the protocol software files (MEM files) is the same as the command file, the pathname does not need to be set in the command file.

Note

Make sure that the pathnames of the download files are correct within the load file. Also remember that pathnames are case-sensitive.

DIAGNOSTICS

The diagnostics provide error messages for the following:

- Incorrect number of parameters supplied
- Could not open target device
- Could not open command file
- Command file format error
- Could not open executable image file

Appendix A Message Interface

The ICP2432 STREAMS driver has a simple message interface consisting of three messages it can accept from upstream and four messages it can send upstream. An application program does not create any of these messages. The Stream Head interprets the user's request and creates the appropriate message to be sent downstream, or dismantles a message coming upstream and sends the appropriate piece(s) to the user. For modules in the STREAM above the driver, the messages can be created and passed to the driver or received from the driver in the formats shown below.

A.1 Downstream

The messages that the ICP2432 STREAMS driver can accept at its downstream interface (from upstream) are `M_FLUSH`, `M_IOCTL`, and `M_DATA`.

Any `M_DATA` message received by the driver will be sent to the ICP intact. An `M_DATA` message should only be sent to the driver if it contains data intended for the ICP.

`M_IOCTL` messages contain the `ioctl` command and data (if necessary) described in Section 4.5 on page 34. The data must conform to the `iocblk` structure defined by STREAMS.

If an `M_FLUSH` message is received, the driver flushes its queues in the algorithm described by STREAMS. See the *Programmer's Guide: STREAMS* in the DIGITAL UNIX Documentation Library for more information.

A.2 Upstream

The messages that the ICP2432 STREAMS driver can send upstream are `M_IOCACK`, `M_IOCNAK`, `M_DATA`, and `M_ERROR`.

An `M_DATA` message contains data received from the ICP2432. The data is not altered in any way by the driver.

`M_IOCACK` and `M_IOCNAK` messages complete an `ioctl` request. `M_IOCACK` indicates a good completion and `M_IOCNAK` indicates an error. For an `M_IOCNAK` message, the error code can be found in the `ioc_error` field of the `iocblk` structure.

An `M_ERROR` message is sent upstream when a fatal error occurs during the direct memory access of data between the driver and the ICP. The error code is included in the `M_ERROR` message as the first byte of the data portion.

Once the driver generates an `M_ERROR` message, the STREAM is effectively shut down, allowing only `poll` and `close` calls to be made.

Index

Customer Report Form

We are constantly improving our products. If you have suggestions or problems you would like to report regarding the hardware, software or documentation, please complete this form and mail it to Simpact at 9210 Sky Park Court, San Diego, CA 92123, or fax it to (619) 560-2838.

If you are reporting errors in the documentation, please enter the section and page number.

Your Name: ______________________________

Company: ______________________________

Address: ______________________________

____

____

Phone Number: ______________________________

Product: ______________________________

Problem or
Suggestion: ______________________________

____

____

____

____

____

Simpact, Inc.
Customer Service
9210 Sky Park Court
San Diego, CA 92123