USER'S GUIDE

# SwiftNet 3.5
# Host Software

Solaris / HP 9000/700 / Digital UNIX Version

PENTEK

Pentek, Inc.
One Park Way
Upper Saddle River, NJ 07458
(201) 818-5900
http://www.pentek.com/



Manual Part No: sn35h Rev: H - December 17, 1999

## SwiftNet Host Software - Revision History

| Date | Man Rev | SwiftNet Rev | Comments |
|---|---|---|---|
| Aug 1993 - Dec 1995 | A - E | 2.0 Beta - 3.1 | Contact Pentek for revision history. |
| Nov 27, 1996 | F | 3.2 | Updated installation instructions to reflect CD-ROM media. Added installation instructions for Digital UNIX Host and device drivers. Added documentation for PNKD_ADDR and function. Added PNDISP start instructions for HP VMEbus systems. General clean-up and typographical corrections. |
| Aug 15, 1997 | F.1 | 3.2 | Force 5V interrupts now supported. Minor formatting and typographical corrections. |
| Oct 1997 | F.2 | 3.2 | Minor corrections to SwiftNet API descriptions. Note on using Force 5V with Solaris 2.5.1. Note on HP-UX A16 addressing. |
| Apr 17, 1998 | G | | Added PNKC_SENDEX function, sect. 4.2 Allocating Shared Memory. |
| Jan 4, 1999 | G.1 | 3.4 Alpha | Updated SwiftNet version numbers to reflect current shipping version. Added plog and plogf functions. Added Sections 7.2, 6.3.4, 6.3.4.1, and 6.3.4.2. Added new code examples. |
| Jan 11, 1999 | G.2 | 3.4 Alpha | Added Universal VME Interrupt Driver compatibility chart. Removed outdated Force driver revision information. Removed verification of Force driver, since it varies upon model. |
| Feb 24, 1999 | G.3 | 3.4 Alpha | **`pncfg`** Processor Map for the Model 4290/4291 was added. Added information about Bit3 616 adapter. Added note about SwiftNet I/O functions and having to declare the functions in all source files. Added section on installing an interrupt handler for 'C6x processors. Added information about memory models available. |
| May 24, 1999 | G.4 | 3.4.1 | Changed version to reflect current shipping version. |
| Dec. 17, 1999 | H | 3.5 | Changed version to reflect current shipping version. Changed Table 2-1 to reflect changes in supported bus adapters and board drivers. Added Section 2.8, Bit 3 616/617 Driver Installation. Added Section 3.1.1, Starting Swiftnet server with Bit 3 616/617. Added Section 3.1.6, Starting Swiftnet server on a Sun with the Universal VME interrupt Driver. Removed former Section 2.11, no longer supports HP-UX 9.0x. |

## Software License Agreement

Pentek grants to you a non-exclusive, non-transferable license to use the object code and source code Programs and related documentation in this package by a single user computer. A single user computer is defined as any stand-alone machine, or any individual machine connected to a network, on which only a single user is able to use the Programs. Use of the Programs by more than one user requires additional paid licenses (contact factory for details).

You may make two copies of the Programs for backup, modification, and archival purposes. Title of the Programs is not transferred to you by this license. Any sublicense, assignment, or other transfer of Programs or the rights or obligations of this agreement without the written prior consent of Pentek is prohibited.

## Software Limitations

Pentek does not warrant that the programs will be free from error or meet your specific requirements. You assume complete responsibility for decisions made or actions taken based on information obtained using the programs. Any statements made concerning the utility of the programs are not to be construed as expressed or implied warranties. Pentek makes no warranty, either expressed or implied, including but not limited to, any implied warranties of merchantability and fitness for a particular purpose, regarding the programs an makes the programs available solely on an "as is" basis. Pentek shall not be responsible of incidental or consequential damages.

# *Table of Contents*

***Page***

### *Chapter 1: Overview*



### *Chapter 2: Installation*

# Table of Contents

# *Table of Contents*

*Page*

### *Chapter 4: The SwiftNet Client Application Programming Interface*



### *Chapter 5: The SwiftNet Device Application Programming Interface*

# Table of Contents

**Page**

## Chapter 6: Standard I/O



## Chapter 7: SwiftNet Operation



## Appendix A: Demonstration Programs

# *Chapter 1: Overview*

## 1.1 General Description

Pentek's SwiftNet standard provides a simple, interrupt driven, inter-device communication protocol for distributed Digital Signal Processing applications. SwiftNet offers the following key features:

- High-level geographic (symbolic) addressing, in which each device is referred to by a user-assigned name rather than a range of addresses
- Support for a wide range interconnect schemes, including shared memory on VME and MIX buses, bus adapters, embedded hosts, TMS320C40 communication ports, and Ethernet links
- An API (Application Programming Interface) containing C language library functions that allow the user to implement the SwitfNet protocol in custom applications
- The SwiftNet File I/O Server **`snio`** and **`sniox`**, which includes two libraries that simplify the use of host resources by providing a familiar standard C interface to file and console I/O

All of Pentek's more sophisticated software offerings, including the SwiftTools DSP Software Development Package and File Transfer Language (FTL), rely upon the SwiftNet protocol to facilitate communication between the host system and DSP targets.

## 1.2 Supported Platforms

SwiftNet is available for a wide variety of platforms:

- Model 4940 is a PROM-based implementation of SwiftNet designed to run the SwiftNet Node Software on a supported Node Controller using an Ethernet link to communicate with the Host (see the ***SwiftNet Node Firmware*** manual). SwiftNet utility programs, the SwiftNet API, and SwiftNet I/O libraries are supplied on CD-ROM and described in this manual.
- Model 4941 is a VxWorks developers package for 680x0, SPARC, and PowerPC-based Node Processors (see the ***SwiftNet Server Libraries for VxWorks User's Guide***) that utilizes an Ethernet link for Host-Node Controller communications. SwiftNet utility programs, the SwiftNet API, and SwiftNet I/O libraries are supplied on CD-ROM and described in this manual.
- Model 4945 is an implementation of SwiftNet for PC compatibles running Microsoft® Windows® 95/Windows NT® (option 003), Solaris 2.x (option 004), HP 9000/700 systems running HP-UX 9.0x (option 006), and DEC Alpha systems running Digital UNIX (option 007) - Microsoft's Window's NT is not currently supported when using a DEC Alpha processor. This version supports a wide range of bus adapters and embedded processor boards, including the Bit 3 Model 466 SBus-to-VMEbus Bus Adapter (available from Pentek as Model 4221). For options 006, and 007, all required software is provided with the SwiftNet Host Software CD-ROM and is covered by this manual.

## 1.3 Supported Pentek Products

Table 1-1, below, lists all Pentek products are supported by SwiftNet, and any application notes.

**Table 1-1: Pentek Products Supported by SwiftNet**

| Model | Description | Notes |
|---|---|---|
| 4244 | Single TMS320C30 MIX Module | No SwiftNet JTAG debugging support |
| 4247 | Dual TMS320C30 MIX Module | No SwiftNet JTAG debugging support |
| 4254 | Single TMS320C40 MIX Module | No SwiftNet JTAG debugging support |
| 4257 | Dual TMS320C40 MIX Module | No SwiftNet JTAG debugging support |
| 4269 | Dual TMS320C40 VMEbus Board | No SwiftNet JTAG debugging support |
| 4270 | Quad TMS320C40 VMEbus Board/MIX Module | |
| 4283 | Single TMS320C30 DSP MIX Baseboard | No SwiftNet JTAG debugging support |
| 4284 | Single TMS320C40 DSP MIX Baseboard | No SwiftNet JTAG debugging support |
| 4285 | Octal TMS320C40 DSP VMEbus Board | Supports DSP MIX Module Models 4247 & 4257 |
| 4280 | Dual TMS320C31 VMEBus Board with Analog I/O | No SwiftNet JTAG debugging support |
| 4200 | 68030 VME-VSB-MIX Baseboard with C30/ C40 DSP MIX Co-Processors | Supports DSP MIX Module Models 4244, 4247, 4254, 4257, and 4270. (Option -020 is supported with Model 4244/4227/4257/4257). No SwiftNet JTAG debugging support. |
| 4201 | 68030 VME-MIX Baseboard with C30/C40 DSP MIX Co-Processors | |
| 4202 | VME-MIX Baseboard with C30/C40 DSP MIX Co-Processors | |
| 4288 | Multi-SHARC I/O Processor for VMEbus Systems | No SwiftNet JTAG debugging support |
| 4290 | TMS320C6201 Digital Signal Processor VMEbus Board | |
| 65xx | Multichannel, Multiple input Wideband and Narrowband Digital Receivers | No SwiftNet JTAG debugging support |
| 7110 | TMS320C44 Digital Signal Processor PMC Module | No SwiftNet JTAG debugging support |

## 1.4 SwiftNet Terminology

**Client** A SwiftNet Client is a hardware device, or a software process on the node or host, which sends a SwiftNet Signal.

**Device** A SwiftNet Device is any piece of hardware that responds to a SwiftNet Signal. The response to a SwiftNet signal consists of at least one returned value and optional additional data.

**Host** A SwiftNet Host processor is a computer system (typically a Sun or HP workstation) used for DSP code development.

**Kernel** The SwiftNet Kernel is the lowest level of inter-device communication for DSP targets. The function of the kernel is to receive a signal through an interrupt from another processor. The kernel will then either call a local Interrupt Service Routine associated with that signal, or pass the signal on to another processor.

**Node** A SwiftNet Node is a VMEbus or VXIbus card cage. A processor of some sort is required for each node to manage the interaction between SwiftNet devices and relay signals to and from other nodes and hosts. Node processors are typically MC680X0 boards or SPARC boards resident in the card cage.

**NOTE:** A SwiftNet Host may also be a Node if the SwiftNet devices are directly accessible (i.e., a Sun system with an embedded VMEbus card cage).

**Server** The SwiftNet Server is a process run on the Node processor that downloads the SwiftNet kernel to the Target DSP(s) and otherwise manages the SwiftNet protocol.

**Signal** A SwiftNet Signal is a message passed from a Client to a Device. Signals are sent via the node processor to devices within the same card cage or to devices in different cages. A Signal consists of a signal number (from 0 to 255) and optional data.

## 1.5 Obtaining Technical Support

If you need assistance with installing or using SwiftNet software, Pentek's customer service may be contacted in the following ways:

- Phone: (201) 818-5900
- Fax: (201) 818-5941
- Email: support@pentek.com

Please have the following information ready before calling, or include it in your email:

- Operating system type and version
- Model of host
- Method of Host/VMEbus connection (Embedded Processor, Bus Adapter, Ethernet)
- Target boards in system
- SwiftNet version

## 1.6 Typographical Conventions

The following conventions are used in this manual:

| | |
|---|---|
| ***monospace italic*** | Indicates a user-specified parameter. |
| **monospace** | In examples, shows system prompts, text from files, error messages, and messages printed by the system. |
| **monospace bold** | In examples, indicates that the user should type the text verbatim. |
| **<parameter>** | In examples, indicates that this parameter is mandatory. |
| **[parameter]** | In examples, indicates that this parameter is optional. |
| **[xxx,yyy]** | In examples, indicates that the user should enter one and only one of the parameters enclosed in brackets. The brackets themselves should not be entered. |

# *Chapter 2: Installation*

## 2.1 General Information

This chapter describes how to install Pentek's SwiftNet Host Software and drivers for supported bus adapters and embedded processor boards. SwiftNet Host Software is provided on CD-ROM. Contact Pentek if another media format is required.

**NOTE:** You must have **root** access to the system on which SwiftNet is being installed.

Installation is divided into four topics:

- ❑ Section 2.2 covers SwiftNet Host Software installation
- ❑ Section 2.3 covers SwiftNet Host Software configuration
- ❑ Section 2.6 covers bus adapter and embedded host device driver installation
- ❑ Section 2.13 covers target board configuration

## 2.2 SwiftNet Host Software Installation

This section provides step-by-step instructions for installing software packages from the Pentek software distribution CD-ROM. If you require another media format, contact Pentek at the phone number provided on the title page of this manual.

1) Login as **root** and mount the Pentek software distribution CD:

```
login: root

Password: ********
```

**Solaris 2.x:** `# mount -r -F hsfs /dev/sr0 /cdrom`
(When using volume management, the system automatically mounts the CD-ROM drive at **/cdrom/cdrom0**.)

**HP-UX 9.0x & 10.10:** `# mount -o ro -t cdfs /dev/c201d2s0 /cdrom`

**Digital UNIX:** `# mount -r -t cdfs -o noversion /dev/rz5c /cdrom`

Note the following:

- The CD-ROM device name may vary with system configuration.
- These examples assume the directory **/cdrom** already exists.

## 2.2 SwiftNet Host Software Installation (continued)

2) Using your user account, run the Pentek Software CD Extraction Utility with the commands appropriate to your platform (the CD-ROM device name may vary):

```
login: username

Password: ********

Solaris 2.x:            # cd /cdrom/cdrom0/software

                        # ./extract.sol

HP-UX 9.0x & 10.10 :    # cd /cdrom/software

                        # ./extract.hp

      or:               # cd /cdrom/SOFTWARE

                        # "./EXTRACT.HP;1"

Digital UNIX:           # cd /cdrom/software

                        # ./extract.dec
```

3) The Pentek Software CD Extraction Utility will list all Pentek software packages available on the distribution CD. Select **`SwiftNet Host Software`** option.

4) When prompted for the Destination directory, enter the name of the directory into which the software should be installed.

**NOTE:** The Destination directory must exist prior to running the Extraction Utility. The software selected in Step 3 will be installed into a subdirectory below the specified Destination directory.

5) After installation is complete, enter **`Q`** to exit the Extraction Utility or select an additional package to install.

## 2.3 SwiftNet Host Software Configuration

1) Create a directory for storing SwiftNet node configurations:

```
% mkdir /usr/local/swiftconfig
```

2) The following environment variables are required for proper SwiftNet operation and should be added to the **.login** or **.cshrc** file located in your **$HOME** directory. These variables must be set prior to launching OpenWindows or HP Vue.

Set the **SWIFTNET_HOME** environment variable to point to the SwiftNet directory created during installation:

```
setenv SWIFTNET_HOME /usr/local/swiftnet-x.x
```

Set the **SWIFTNET_CONFIG** environment variable to point to the node configuration directory created in Step 1:

```
setenv SWIFTNET_CONFIG /usr/local/swiftconfig
```

Set the **MANPATH** environment variable to point to each man page directory:

```
setenv MANPATH /usr/man:$SWIFTNET_HOME/man
```

If you are using SwiftTools (2.5 or later only) and the TI compiler with SwiftNet, set the **DSPTOOLS** environment variable to point to the directory where the TI compiler is installed:

```
setenv DSPTOOLS /usr/local/c30tools
```

If you wish to access the SwiftNet demo projects from within SwiftTools, set the **SWIFTTOOLS_PROJ** environment variable as shown below:

```
setenv SWIFTTOOLS_PROJ $SWIFTNET_HOME/examples
```

3) Add the directory **$SWIFTNET_HOME/*xxx*/bin** to your path (where ***xxx*** is **solaris** for Solaris 2.x, **hp9700** for HP-UX 9.0x/10.10, or **alpha-unix** for Digital UNIX), either by modifying your existing path or by adding the following line after it:

```
set path = ( $path $SWIFTNET_HOME/xxx/bin $SWIFTTOOLS_HOME/xxx/bin )
```

**NOTE:** This directory should be placed BEFORE any SwiftTool path entries.

## 2.3 SwiftNet Host Software Configuration (continued)

4) Specify whether SwiftNet will operate in network or local mode.

**Network operation** is typical and supports multiple connections to one or more DSP targets.

**Local operation** provides faster communication when using SBus-to-VMEbus and integral VMEbus systems, but does not permit multiple host-to-DSP target connections. In local mode, multiple SwiftTools Sessions can be opened simultaneously, but only one instance of SwiftTools can run at a time. (A complete description of SwiftTools Sessions can be found in the ***SwiftTools User's Guide***.)

The operating mode is determined by which shared library is used, and can be easily changed. The following environment variable should be placed after any existing shared library environment variable. Replace ***libx*** with **libl** for local mode or **libn** for network mode.

**Solaris 2.x:**
```
setenv LD_LIBRARY_PATH $SWIFTNET_HOME/
solaris/libx:$LD_LIBRARY_PATH
```

**HP-UX 9.0x/10.10:**
```
setenv SHLIB_PATH $SWIFTNET_HOME/hp9700/
libx:$SHLIB_PATH
```

**Digital UNIX:**
```
setenv LD_LIBRARY_PATH $SWIFTNET_HOME/
alpha-unix/libx:$LD_LIBRARY_PATH
```

Local SwiftNet operation also requires that the environment variable **SWIFTNET_PARAMS** be set with the parameters ***bus-type arg1 arg2***. For the Bit3 466 SBus-to-VMEbus adaptor, add the environment variable specific to your operating system:

**Solaris 2.x:**
```
setenv SWIFTNET_PARAMS "bts944 0 0x11"
```

Refer to Section 3.1 of this manual for a full description of these arguments.

5) ***HP-UX 9.0x/10.10 users only:*** In order for interactive utilities such as PNCFG and SwiftTools to display properly, add the following two lines to the **/.vue/sessions/current/vue.resources** file while logged in as **root**:

```
pncfg*font      9x15

swift*font      9x15
```

## 2.4 DEC Alpha Support

SwiftNet supports the Digital Equipment Alpha series processor. However, at this time SwiftNet requires using Digital UNIX OS. Microsoft Windows NT is not supported when using a DEC Alpha processor.

### 2.4.1 SwiftNet installation

A kernel device driver for the VME bus is included in the DEC UNIX Swiftnet package. This device module must be installed in the kernel before using the SwiftNet driver.To install SwiftNet, do the following steps:

1) Login to the workstation as a superuser.
   **root#**
   **cd $SWIFTNET_Home/alpha_unix/module-kit**

2) Type: **csh ./INSTALL**
   **(alpha)# csh ./INSTALL**

The SwiftNet installation script installs the SwiftNet modules and displays the following on the screen:

```
Installation is complete. Please run shutdown and reboot the
system.
```

Once the system has rebooted, the driver can be loaded and unloaded with the following commands:

```
# sysconfig -c swiftnet        (load the driver)
# sysconfig -u swiftnet        (unload the driver)
```

The may also be loaded in the system startup script if preferred.

## 2.5 Important DEC UNIX Notes

This section contains some important notes on using DEC UNIX with SwiftNet.

- Only VME boards capable of Release on Acknowledge (ROAK) interrupt response are supported. Pentek boards supporting ROAK include the Model 4285 and Model 65xx series boards. Release on Register access support is in development.

- Memory scanning outside of short address space is not supported. Before using the **pncfg** tool, you need to know the A24 and A32 memory map addresses. It is required that the memory space reserved by the inbound PCI mapping of the DEC Alpha VME board be excluded in the memory map (see the Hardware Reference for the AXPvme board--the DEC Alpha VME board). Due to limitations in the current release of the DEC UNIX operating system, access to VME address space where no VME device exists causes a panic in the DEC UNIX kernel.

## 2.5 Important DEC UNIX Notes (continued)

- DSP boards on the MIX bus are not currently supported in SwiftNet for DEC UNIX on the DEC Alpha processor.

## 2.6 Bus Adapter and Processor Board Device Driver Installation

SwiftNet supports the following bus adapter and embedded processor boards:

**Table 2-1: Supported Bus Adapter/Processor Boards**

| Platform | Board | See Section |
|---|---|---|
| Solaris 2.x | Universal VME Interrupt Driver (Force & Themis) | Section 2.7 |
| | Bit3 616/617 PCI-to-VMEbus Adapter (Pentek Models 4228/4229) | Section 2.8 |
| | Bit 3 466/467 Sbus-to-VMEbus Adapter (Pentek Models 4221/4220) | Section 2.9 |
| | Performance Technologies Inc. PT-SBS915 SBus-VME Adapter | Section 2.10 |
| | Force 2CE, 3CE, 5CE, 5V, or CPU-10 VMEbus Embedded Processor | Section 2.11 |
| | General Micro Systems V64 VMEbus Embedded Processor | |
| HP-UX 10.10 | HP 743i, 747i and 748 VMEbus Embedded Processor | Section 2.12 |

**NOTE:** You must be logged in as **`root`** to install these device drivers.

The sections that follow, contain installation instructions for each bus adapter and embedded processor board.

## 2.7 Universal VME Interrupt Driver

The Universal VME Interrupt driver allows you to process VMEbus interrupts from Themis and Force cards using the Solaris OS. Table 2-2 below provides compatibility information for the various boards and drivers. The subsection below provides the proper installation procedure for the driver. This driver is **not** compatible with the SwiftNet packages for DEC Alpha and HP-UX 9.0x/10.x.

**Table 2-2: Universal VME Interrupt Driver Compatibility**

| Board | Compatibility | Notes |
|---|---|---|
| Bit 3 466, 467 | Not Compatible | Must use Bit 3 driver |
| Performance Technologies, Inc. PT-SBS915 | Unknown | Has not been tested by Pentek. |
| Force 2CE, 3CE | Not Compatible | Universal VME Interrupt driver requires 2.x of the Force driver, these Force products use 1.x drivers. |
| Force 5CE, 5V, CPU-10, and other Force boards using version 2.x of the Force driver. | Compatible | Must be using 2.x of the Force driver. If 1.9 or earlier driver is used, the Universal VME Interrupt driver is not compatible. |
| Themis Computer boards including the themvme UPS-1 | Compatible | Must use **`themvme UPS-1 for Solaris 2.6 v1.1`** **and** a Themis Computer patch to use the Pentek leaf driver with the themvme UPS-1 board. Later versions of the themvme driver will have the patch incorporated into the binary. |

### 2.7.1 Universal VME Interrupt Driver Installation

The steps below provide directions for installing Pentek's leaf driver for compatible Themis and Force products.

**NOTE:** If you need to get the Themis patch, contact Themis Computer. They can also provide you with additional information on the patch. The updated **`themvme`** driver should reside in either: **`/kernel/drv`** or **`/platform/sun4u/kernel/drv`** depending on where the original driver was installed.

1) install the manufacturer's Nexus driver. Follow their installation instructions.

2) Copy **`frcint`** and **`frcint.conf`** into **`/kernel/drv/`**
These files originally reside **`SWIFTNET_HOME/solaris/bin`**

## 2.7 Universal VME Interrupt Driver (continued)

### 2.7.1 Universal VME Interrupt Driver Installation (continued)

3) Add the driver to the system by typing:
**`/usr/sbin/add_drv -m '* 0666 root sys' -1 'vmeint' frcint`**

   If frcint has already been added to the system, it must be removed before adding it again.

   a) Remove frcint by typing:
   **`/usr/sbin/rem_drv frcint`**

   b) Repeat step 3 to add the driver.

4) Generate /dev/vmeint symbolic link to the device node by editing **`/etc/devlink.tab`** file. The following line at the end of the **`devlink.tab`** file: **`type=ddi_pseudo; name=vmeint \MO`**

5) Verify and update, if needed, the frcint.conf file. This file is updated to use interrupt level 3 and vectors 0xd0-df by default. If you want to use other levels and vectors, you should ensure these are set properly.

6) Reboot your system.

The universal VME interrupt driver uses the following **`pndisp`** option;

**`pndisp frcvme 0 0x0`**

In order to access the VMEbus using the Themis driver, make sure read permission is assigned to u**`sr/group/others`** for **`/dev/vme16d16`**, **`/dev/vme24d16`**. To assign read permissions, do the following;

a) Login in superuser mode.

b) **`cd`** to **`/dev`**

c) **`chmod a+rw vme`**

## 2.8 Bit 3 Model616/617 PCI-to-VMEbus Adapter (Pentek Models 4228/4229) Solaris 2.x Device Driver Installation

The steps below provide directions for installing Pentek's leaf driver for a Bit 3 Model 616/617 PCI-to-VMEbus Adapter on a PCI based Sun using Bit 3's 946 driver.

1) Install the Bit 3 946 driver. Follow their installation instructions.

2) Copy **`btpint`** and **`btpint.conf`** into **`/kernel/drv/`**
   These files originally reside **`SWIFTNET_HOME/solaris/bin`**

3) Add the driver to the system by typing:
   **`/usr/sbin/add_drv -m '* 0666 root sys' -1 'vmeint' btpint`**

   If btpint has already been added to the system, it must be removed before adding it again.

   a) Remove btpint by typing:
      **`/usr/sbin/rem_drv btpint`**

   b) Repeat step 3 to add the driver.

## 2.8 Bit 3 Model616/617 PCI-to-VMEbus Adapter Solaris (Pentek Models 4228/4229) 2.x Device Driver Installation (continued)

4) Generate /dev/vmeint symbolic link to the device node by editing **`/etc/devlink.tab`** file. The following line at the end of the **`devlink.tab`** file: **`type=ddi_pseudo; name=vmeint \MO`**

5) Verify and update, if needed, the btpint.conf file. This file is updated to use interrupt level 3 and vectors 0xd0-df by default. If you want to use other levels and vectors, you should ensure these are set properly.

6) Reboot your system.

The universal VME interrupt driver uses the following **`pndisp`** option;

```
pndisp btp 0 0x0
```

## 2.9 Bit 3 Model 466/467 Sbus-to-VMEbus Adapter (Pentek Models 4221/4220) Solaris 2.x Device Driver Installation

SBus adaptor card(s) must be installed in your Sun system prior to installing the Bit 3 device driver. However, the driver will load properly with the remote system (i.e., the VME cage) powered off or with the cable disconnected.

The files required by the Bit 3 adaptor are located in the directory **$SWIFTNET_HOME/bit3/944/v3.4** (Solaris 2.x). These directories contain the following subdirectories:

**sys** Contains source files, makefile, and installation script for the Bit 3 SBus device driver.

**src** Contains source files and makefile for all Bit 3 example programs. The makefile can be used to compile all example programs for a particular SBus adaptor model.

**util** Contains utilities to manipulate the Bit 3 adapter.

1) Change to the Bit 3 driver directory:

```
# cd $SWIFTNET_HOME/bit3/944/v3.4/sys
```

2) Execute the installation script with the following command:

```
# make install
```

The installation script creates links in **/usr/include/sys** and copies the installation scripts and compiled object file to the **/dev** directory.

3) When the installation script presents the option of modifying **/etc/rc.local** to automatically load the SBus device driver on boot, answer **Y** to allow modification or **N** to leave **/etc/rc.local** untouched. In most cases, this option should be selected as **Y**.

4) When the installation script prompts for the number of devices to create, enter the number of Bit 3 SBus adapters installed. A unique entry for each adaptor will be added to the **/dev** directory, starting with **/dev/bts0**. Devices are numbered by the sequence of their installation rather than by their SBus slot numbers. Thus, the first Adaptor installed will be **/dev/bts0**, the next will be **/dev/bts1**, etc.

5) Verify that the hardware and device driver are properly installed by entering the command below:

```
# /usr/sbin/modinfo
```

## 2.9 Bit 3 Model 466/467 Sbus-to-VMEbus Adapter (Pentek Models 4221/4220) Solaris 2.x Device Driver Installation (continued)

Look for the following statement in the output of this command:

```
Node 'Bit3,bts', unit#0
```

If this statement does **not** appear, the SBus Adaptor card is not properly installed. If that statement is accompanied by the comment **(no driver)**, the card is properly installed but the device driver is not.

**NOTE:** Both the Model 943 and Model 944 device drivers support host interrupts from the DSP board.

## 2.10 Performance Technologies Inc. PT-SBS915 SBus-VME Adapter

1) Install the PTI drivers as described in the Performance Technologies manual.

The PTI driver handles are named **ptvme*xx*d*yy*** where ***xx*** and ***yy*** are the VMEbus access type. The A24/D16 driver handle would be **ptvme24d16**. These handles must be symbolically linked to names in the format of **vme*xx*d*yy***. This can be done by issuing the following commands, assuming the PTI driver handles were installed in the **/dev** directory:

```
# cd /dev

# ln -s ptvme24d16 vme24d16
```

This must be done for each driver handle.

## 2.11 Force 2CE, 3CE, 5CE, 5V, or CPU-10 VMEbus Embedded Processor Solaris 2.x Device Driver Installation

Each manufacturer provides VMEbus device drivers specific to their hardware. Refer to the manufacturer's documentation for installation instructions. To determine if your Force driver and product is compatible with Pentek's Universal VME Interrupt Driver, see Table 2-2, on Page 17. 2CE, 3CE, and 5V board users should observe the following instructions.

**NOTE:** Force 2CE and 3CE users should use version 1.9 of the Force driver, which is not compatible with Pentek's Universal VME Interrupt Driver. 5CE users who wish to use the 1.9 Force driver should follow the steps below. If you want to use the Universal VME Interrupt Driver, follow the installation instructions starting on Page 17.

## 2.11 Force 2CE, 3CE, 5CE, 5V, or CPU-10 VMEbus Embedded Processor Solaris 2.x Device Driver Installation (continued)

*2CE, 3CE, and 5CE boards only:*

1) Copy the following files:

```
# cp $SWIFTNET_HOME/solaris/bin/vmeint_2ce /usr/kernel/drv

# cp $SWIFTNET_HOME/solaris/bin/vmeint_2ce.conf /usr/kernel/drv
```

2) Add the driver:

```
# /usr/sbin/add_drv -m '* 0666 root sys' -i vmeint vmeint_2ce
```

3) Enter the following command to confirm that the driver has been created:

```
# ls /devices/vme@1,efe00000/vmeint:vmeint
```

If the system reports that the file is not found, driver installation has failed.

4) Symbolically link the Force driver by typing:

```
# ln -s /devices/vme@1,efe00000/vmeint:vmeint /dev/vmeint
```

*5V boards only:*

1) Copy the following files:

```
# cp $SWIFTNET_HOME/solaris/bin/vmeint_5v /usr/kernel/drv

# cp $SWIFTNET_HOME/solaris/bin/vmeint_5v.conf /usr/kernel/drv
```

2) Add the driver:

```
# /usr/sbin/add_drv -m '* 0666 root sys' -i vmeint vmeint_5v
```

3) Enter the following command to confirm that the driver has been created:

```
# ls /devices/iommu@0,10000000/VME@0,7ffffe00/vmeint:vmeint
```

If the system reports that the file is not found, driver installation has failed.

4) Symbolically link the Force driver by typing:

```
# ln -s /devices/iommu@0,10000000/VME@0,7ffffe00/
        vmeint:vmeint/dev/vmeint
```

## 2.11 Force 2CE, 3CE, 5CE, 5V, or CPU-10 VMEbus Embedded Processor Solaris 2.x Device Driver Installation (continued)

**NOTE:** The 2CE/3CE/5V device driver supports host interrupts from the DSP board under Solaris 2.4 and 2.5.

2CE/3CE installation requires Solaris patch #101318-75 (available from Sun) to work under Solaris 2.3.

## 2.12 HP 743i, 747i, and 748 VMEbus Embedded Processor HP-UX 10.10 Device Driver Installation

SwiftNet includes kernel drivers for HP-UX 10.10, however before installing these Pentek drivers, the following HP-UX patches must first be installed:

- **`PHKL_7223`**
- **`PHCO_7074`**
- **`PHSS_7224`**

These patches are available from the January 1997 Extensions CD, or from the Hewlett-Packard web site at: **`http://us-support.external.hp.com/`**

1) Copy the following files to the appropriate system directories:

```
# cd swiftnet-3.4/hp9700-10/lib/ptk

# cp libptk.a /usr/conf/lib

# cp ptk /usr/conf/master.d
```

2) Open the **`ptk`** file and edit, for each device, by replacing the **`xx`** in **`ptkxx`** with the numbers shown in Table 2-3 below for the pentek device you are using. Then remove the asterisk (comment symbol) on those lines.

**Table 2-3: Codes used for Editing `ptk` File**

| For Pentek Board | Replace `xx` in `ptkxx` |
|---|---|
| 65xx | 65 |
| 4270 | 70 |
| 4280 | 80 |
| 4283 | 83 |
| 4284 | 84 |
| 4285 | 85 |
| 4290/4291 | 90 |

## 2.12 HP 743i, 747i, and 748 VMEbus Embedded Processor HP-UX 10.10 Device Driver Installation (continued)

3) Create the system file:

```
# cd /stand/build

# /usr/lbin/sysadm/system_prep -s system
```

4) Edit **/stand/build/system** to add the appropriate driver. For example, for the Model 4285, add **ptk85**. The system file will look similar to the listing below:

```
Drivers and
Subsystems
CentIf
CharDrv
asio0
asp
audio
c700
c720
cdfs
clone
core
dlpi
echo
eeprom
eisa
ffs
foreign
framebuf
graph3
hil
(listing continued)
hpstreams
inet
inet_clts
inet_cots
ite
klog
lan2
lasi
ldterm
lv
lvm
netdiag1
netman
nfs
ni
pa
pckt
pipedev
pipemod
ptem
(listing continued)
ptm
pts
sad
sc
sctl
sdisk
strlog
strpty_included
timod
tirdwr
tpiso
uipc
vme2
vxbase
wsio
ptk85
* Kernel Device Info
* Tunable parameters
default_disk_ir 1
nstrpty 60
```

5) Configure/Build the kernel:

```
# /usr/sbin/config -s system

# make -f config.mk
```

6) Backup old kernel and copy new kernel:

```
# mv /stand/system /stand/system.prev
# mv /stand/vmunix /stand/vmunix.prev
# mv /stand/build/system /stand/system
# mv /stand/build/vmunix_test /stand/vmunix
```

## 2.12 HP 743i, 747i, and 748 VMEbus Embedded Processor HP-UX 10.10 Device Driver Installation (continued)

7) Append the following configuration information to **/sbin/lib/vme/vme.cfg:**

```
//
// Configure a 4284:      A16 base address = 0x0000
//                        A24 base address = 0x000000
//                        A32 base address = 0x00000000
//                        A16 size = 0x100
//                        A24 size = 4MB
//                        A32 size = 16MB

// Processor Declaration
//             Name       CPU/Card   ID    Options
//                        Number
proc           Model_4284 *          *

// Memory Declaration Record
memory A16 {
//       address          name             card
         0x0000:0x0100    host_ctrl84      Model_4284
}

memory A24 {
//       address          name             card
         0x000000:4M      dual24_port84    Model_4284
}

memory A32 {
//       address          name             card
         0x00000000:16M   dual32_port84    Model_4284
}
```

These example files define a Model 4284. Similar files can be created for models 4280, 4283, and 4270 by changing the names. The base addresses and memory sizes for each memory type (A16, A24 and A32) must match the hardware configuration on the Pentek board. Refer to the operating manual of the board you will be accessing for a complete description.

For a detailed information on the **vme.CFG** file, refer to the *VME Configuration Guide for HP-UX*.

8) Start **vme_config** by typing:

```
# vme_config
```

By default, this command reads configuration information from **/etc/vme/vme.CFG** and writes to the nonvolatile memory of the processor on which it is executed.

## 2.12 HP 743i, 747i, and 748 VMEbus Embedded Processor HP-UX 10.10 Device Driver Installation (continued)

9) If any errors occur, refer to ***Appendix B: Warning and Error Messages*** listing in the ***VME Configuration Guide for HP-UX***.

10) Once you have a correct configuration (indicated by a message from **`vme_config`** that a configuration was successfully generated), you must ensure that your VME card configuration matches that specified in the **`vme.CFG`** file. The file **`/etc/vme/system.log`** contains a description of your current configuration.

11) Reboot the system.

12) Run **`pndisp`**:

```
# pndisp vme2 null 0x0
```

## 2.13 Target Board Configuration

1) Configure each target board as described in their respective Operating manuals, keeping the following in mind:

   - ❑ Dual Port RAM Base Addresses for each board must be set so as not conflict with any other device in the VME card cage. (Note that on Pentek Models 4254, 4257, 4244, and 4247, the Base Address of Global SRAM is hard-wired and cannot be set in hardware or software.)

   - ❑ The VME Interrupt Level must be set to match the setting in the PNCFG Target Configuration Utility using the on-card jumpers. (See the Operating Manual of your specific board for more details.)

   - ❑ Models 4269 and 4270 must also be configured such that both or all four processors can issue VME Interrupts on the same level (this is the factory default setting, at Level 3). See Section 2.2.6 of the 4269 or 4270 Operating Manual for further details.

2) Use the **`pncfg`** utility to configure SwiftNet (see Section 3.2 of this manual for detailed information on PNCFG).

3) Use the **`sntest`** utility to verify that your hardware and software are communicating properly:

```
% sntest <node-name> <board-name> <proc-num>
```

   A count of successful transmissions will run continuously until interrupted by the user with **`^C`**.

***This page is intentionally blank***

# *Chapter 3: SwiftNet Utility Programs*

## 3.1 PNDISP SwiftNet Server Program

SwiftNet runs on various hardware configurations, including:

1) An Sbus-based machine with a bus adapter to a VMEbus or VXIbus cage containing Pentek hardware

2) A Sun workstation with an integrated VMEbus where Pentek hardware shares the system bus with the Sun processor board

3) A VMEbus cage with an embedded SPARC processor board (Force 2CE, 3CE, 5CE, CPU-10, and General Micro Systems V64)

4) An HP workstation with an integrated VMEbus/VXIbus or a VMEbus/VXIbus cage with an embedded HP processor board

5) A VMEbus processor board (680x0 node controller) which communicates with a host through an Ethernet connection

In the first three configurations, the SwiftNet Server PNDISP must be started on the host with a specific driver for the hardware being used. In the fourth configuration, when using an HP embedded processor board or an HP workstation with an integrated VMEbus, PNDISP is required. In the case of an Ethernet connected node controller, the SwiftNet Server is started under VxWorks or is supplied as firmware in the form of an EPROM (see the ***SwiftNet Server Libraries for VxWorks*** or the ***SwiftNet Node Firmware*** manual depending upon your hardware configuration).

**NOTE:** PNDISP should **not** be run when the local SwiftNet libraries are used.

The general form of the PNDISP command is in the form:

```
# pndisp <bus-type> <argument-1> <argument-2>
```

where:

**<bus-type>** The type of bus adapter or embedded VMEbus processor board.

**<argument-1>** This will depend on the interface type.

**<argument-2>** This will depend on the interface type.

The following sections describe the specific arguments for each hardware and operating system configuration.

## 3.1 PNDISP SwiftNet Server Program (continued)

### 3.1.1 Starting the SwiftNet Server on a Sun with a Bit3 Model 616/617 PCI-to-VMEbus Adaptor(Pentek Models 4228/4229)

The SwiftNet Server is started with a Bit3 616/617 Sbus-to-VMEbus adaptor as follows:

**Solaris 2.x:** `# pndisp btp 0 0x00`

In this configuration host interrupts from the DSP boards are supported.

### 3.1.2 Starting the SwiftNet Server on a Sun with a Bit3 Model 466 Sbus-to-VMEbus Adaptor (Pentek Model 4221)

The SwiftNet Server is started with a Bit3 466 Sbus-to-VMEbus adaptor as follows:

**Solaris 2.x:** `# pndisp bts944 0 0x11`

In this configuration host interrupts from the DSP boards are supported.

### 3.1.3 Starting the SwiftNet Server on a Sun with an integrated VMEbus

The SwiftNet Server is started on a Sun with an integrated VMEbus as follows:

**Solaris 2.x:** `# pndisp vme null 0x0`

In this configuration host interrupts from the DSP boards are not supported.

## 3.1 PNDISP SwiftNet Server Program (continued)

### 3.1.4 Starting the SwiftNet Server on a Sun with a National Instruments VXI-SB2020 Sbus-to-VXI Adapter with SunOS 4.1.3

The SwiftNet Server is started with a National Instruments VXI-SB20202 Sbus-to-VXI adapter as follows:

```
# pndisp nivxi 0 0x00
```

The National Instruments adapter accesses A32 space in 8 MB windows. The last argument sets the base of this window. 0x00 for the first 8 MB, 0x01 for the second 8 MB, etc.

### 3.1.5 Starting the SwiftNet Server on a Sun with a Performance Technologies Inc. PT-SBS915 Sbus-to-VMEbus Adapter

The SwiftNet Server is started with a Performance Technologies Inc. PT-SBS915 Sbus-to-VMEbus adapter as follows:

**Solaris 2.x:** `# pndisp vme null 0x0`

When running under Solaris 2.x, host interrupts from the DSP boards are not supported.

### 3.1.6 Starting the SwiftNet Server on a Sun with the Universal VME Interrupt Driver

The SwiftNet Server is started on a Sun with an with the Universal VME Interrupt Driver as follows:

**Solaris 2.x:** `# pndisp frcvme null 0x0`

In this configuration host interrupts from the DSP boards are supported.

## 3.1 PNDISP SwiftNet Server Program (continued)

### 3.1.7 Starting the SwiftNet Server on a Force 2CE, 3CE, 5CE or CPU-10, or General Micro Systems V64 (Solaris 2.x)

The SwiftNet Server is started on a Force or General Micro Systems board as follows:

**2CE or 3CE:** `# pndisp vme2ce null 0x0`

**5CE:** `# pndisp vme null 0x0`

**CPU-10:** `# pndisp vme null 0x0`

**V64:** `# pndisp vme null 0x0`

Host interrupts from the DSP boards are supported on the 2CE and 3CE. They are not supported on the CPU-10 and V64.

### 3.1.8 Starting the SwiftNet Server on an HP 743i, 747i, or 748 Embedded VMEBus Processor Board

The SwiftNet Server is started on an HP V743i/747i/748 board as follows:

```
# pndisp vme2 null 0x0
```

Host interrupts from the DSP board are supported on models 4280, 4283, 4284, 4270.

### 3.1.9 Starting the SwiftNet Server on an HP V743 Embedded Processor Board

The SwiftNet Server is started on an HP V743 board as follows:

```
# pndisp sicl vxi 0x0
```

At the time of this release, host interrupts from the DSP board are only supported on the models 4200 and 4201.

## 3.2 PNCFG - Target Configuration Program

Before a Target device may be opened in SwiftTools, the Node Server requires certain information about the configuration of the target devices. The program PNCFG facilitates this process in a menu-driven environment. This program must be run before attempting to communicate with the hardware devices in your card cage. The program has two modes, interactive and command-line.

### 3.2.1 PNCFG - Execution

PNCFG can be run in two modes: interactive and command line. To start PNCFG in the interactive mode type:

```
# pncfg
```

After node configuration files have been saved using **PNCFG** in the interactive mode, subsequent launches of **PNCFG** can be done from the command line. The syntax is as follows.

```
# pncfg [-rdvqk] <node_name> [node_name...]
```

This command will retrieve the node configuration file stored by a Configure-Save command in interactive mode. At least one node name must be specified. The command line options are described below.

**-r** Reset the node(s) before configuration.

**-d** Delete the old configuration before re-configuration.

**-v** Verbose mode. This mode displays all the configuration steps.

**-q** Quiet mode. This mode suppresses all messages except errors. Quiet mode overrides Verbose mode. If neither **-q** or **-v** is specified, only warnings and errors are displayed.

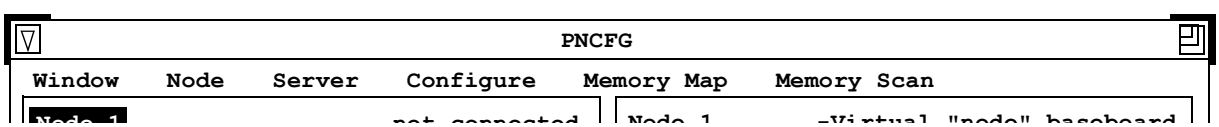
**-k** Kill a previously started PNDISP process.

### 3.2.2 PNCFG - Interactive Mode

When PNCFG is started in the interactive mode a screen similar to the following is displayed:

## PNCFG Screen Layout

| Menu | -Item | Description |
|---|---|---|
| Window | -Redraw | Redraw the screen |
| | -Resize | Resize the screen |
| | -Help | Open the on-line help window |
| | -About | Display the PNCFG version |
| | -Quit | Quit PNCFG |
| Node | -Create | Create an entry in the Node table |
| | -Delete | Delete an entry in the Node table |
| Server | -Connect | Enable a Node to be accessed by SwiftTools |
| | -Disconnect | Disable SwiftTools access to a Node |
| | -Kill | Kill the PNDISP process |
| Configure | -Load | Load the selected Node configuration file |
| | -Save | Store the selected Node configuration to a file |
| | -Add | Add a new Target to the selected Node |
| | -Edit | Edit a Target's configuration |
| | -Remove | Remove a Target from the selected Node |
| | -Update | Update the Target's configuration |
| | -Show | List the Targets associated with the selected Node |
| Memory Map | -A16/D16 | Display occupied address ranges in VME A16/D16 Space |
| | -A16/D32 | Display occupied address ranges in VME A16/D32 Space |
| | -A24/D16 | Display occupied address ranges in VME A24/D16 Space |
| | -A24/D32 | Display occupied address ranges in VME A24/D32 Space |
| | -A32/D16 | Display occupied address ranges in VME A32/D16 Space |
| | -A32/D32 | Display occupied address ranges in VME A32/D32 Space |
| Memory Scan | -A16/D16 | Scan VME A16/D16 Space |
| | -A16/D32 | Scan VME A16/D32 Space |
| | -A24/D16 | Scan VME A24/D16 Space |
| | -A24/D32 | Scan VME A24/D32 Space |
| | -A32/D16 | Scan VME A32/D16 Space |
| | -A32/D32 | Scan VME A32/D32 Space |



## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.2 PNCFG - Interactive Mode (continued)

Each Node listed above represents a single VME card cage. A single node may contain one or more baseboard(s) or MIX Stack(s). The pull-down menus and their selections are described on the following pages. Be aware that the screen shown on the previous page is not what you should expect to see when the program starts. Initially, only the Window and Node menus are available.

The methods used to make selections from the pull-down menus in PNCFG are summarized below.

**Menu Selections**

- Point to the desired PNCFG menu bar selection and click the left mouse button. Then point to the desired menu item and click the left mouse button. When selecting items from the Node (left) window, the mouse cursor must point at the Node Name field.

- Select the menu bar with **<esc><esc>**. Use the ← and → arrow keys to select the desired pull-down menu. Move to the desired menu item with the ↑ and arrow keys and enter the selection with **<return>**.

### 3.2.3 PNCFG - Window Menu

This window contains system level functions, including options to change a window's appearance, to display version information, to provide on-line help and to quit the program after configuration operations have been completed.

**Window-Redraw** Redraws the PNCFG screen.

**Window-Resize** Returns the PNCFG screen windows to their original sizes. Window size may be changed by dragging a corner or edge with the mouse.

**Window-Help** Opens a help window.

**Window-About** Displays the PNCFG version.

**Window-Quit** Quits PNCFG and returns to the operating system prompt.

## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.4 PNCFG - Node Menu

The first step in configuring a Target is to assign a name to its physical location. The Node menu is used to create reference names for different instrumentation crates, or groups of one or more instruments within a given crate.

**Node-Create** Adds a Node name to the list in the Node (left) Window.

**Node-Delete** Removes a name from the list in the Node (left) window This option is unavailable if no Nodes have been created.

**NOTE:** When creating Node names, they are based on the computer's TCP/IP name. When using a VXWorks node, SwiftNet Node names are based on the VXWorks card name.

### 3.2.5 PNCFG - Server Menu

After Nodes have been created, use the Server Menu to enable communications between SwiftTools and one and more of the defined Nodes. Only nodes that are connected in PNCFG can be used as Target devices in SwiftTools. This menu is unavailable if no Nodes have been created.

**Server-Connect** Enable communications between the selected Node and SwiftTools.

**Server-Disconnect** Disable communications between the selected Node and SwiftTools.

**Server-Kill** Kills the PNDISP process which was started prior to executing PNCFG. This option is only available when using an SBus-to-VME adaptor.

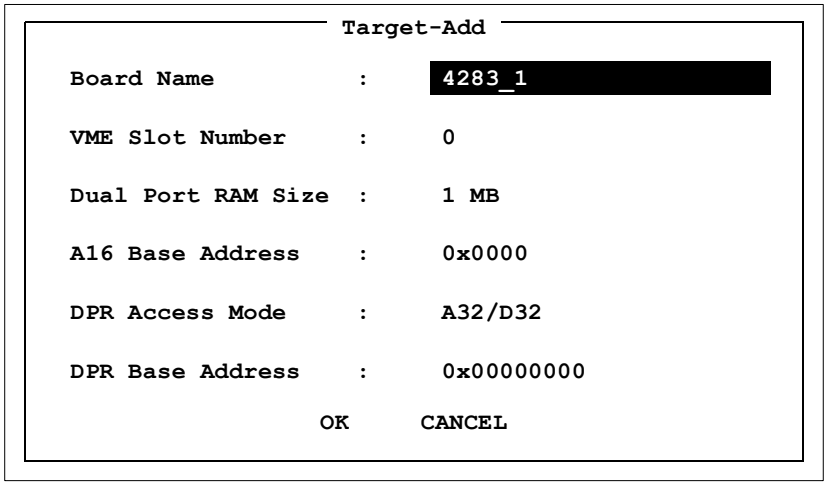
### 3.2.6 PNCFG - Configure Menu

The Configure Menu is used to create the list of hardware devices that comprise the Targets residing in the connected Nodes. Pop-up windows are used to allow the user to assign a name to each Target device within a given node, and to define the VME addressing mode and the base (offset) address of the Dual-Port or Global RAM on a MIX Baseboard or VME Processor board. Another menu choice allows the user to save the node configuration to a file.

**Configure-Load** Load a previously saved Node Configuration.

**Configure-Save** Store the configuration information entered using the menu options below to the file ***node*.pn**.

**Configure-Add** This option displays a window showing the model numbers of Pentek Target boards supported by this release of SwiftTools. Selecting a board from the list displays a second window requesting additional information. A baseboard MUST be added to the Node before any MIX modules may be added. Up to three MIX modules may be added to the Node Configuration after the baseboard configuration has been defined. The Target-Add sub-menu for a Model 4283 baseboard is shown below. The Target-Add menu for other baseboards will be quite similar in appearance. The Target-Add menu for VME processor boards, such as the Model 4269, will not have the Dual Port RAM size field. Descriptions of each of the menu fields begin on the following page.



## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.6 PNCFG - Configure Menu (continued)

**Board Name** Enter the name you wish to use in SwiftTools when referring to this Target board. This field has no default. If no board name is entered, the Target-Add operation fails.

**Slot Number** Enter the number of the VME slot that houses the Target. The default is 0. This entry is for user information only and is not used by the program in any way, since VME systems access connected boards by memory addresses, not by slot numbers.

**Configure-Add** (continued)

**Dual Port RAM Size** This field indicates the amount of Global Dual Port RAM present on the Target processor board. For the model 4283, this may be 1 MB, 4 MB or 8 MB. The allowable values may be cycled through by hitting the space bar or by placing the mouse cursor within the field and clicking the left button.

**A16 Base Address** The base (offset) address used for VME transactions in A16 space. The default value is 0x0000. The content of this field should match the on-card setting, generally accomplished using jumpers. See the Target board's Operating Manual for details.

Note that under HP-UX, boards should be configured for addresses 0x0000 to 0x0F00, on boundaries of 0x100.

**DPR Access Mode** This field allows you to select the mode in which the VMEbus interacts with the Target Baseboard's Dual-Ported Global RAM. The default mode is A32/D32. Other options are A24/D32, A32/D16, and A24/D16. You can cycle through these four options by hitting the space bar with the field highlighted, or by placing the mouse cursor within the field and clicking the left button.

## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.6 PNCFG - Configure Menu (continued)

**DPR Base Address** This field allows the user to define the base (offset) address of the Target's Global Dual-Port RAM for VME Transactions in A24 and/or A32 space. The default address is 0x00000000. The content of this field should match the on-card setting, which may be accomplished using jumpers, switches or by programming a register in A16 space. See the Target board's Operating Manual for details.

**Interrupt Level** Set this field to correspond to the VME interrupt level selected on-card. See the Target board's Operating Manual for details on making this selection. The default value for this field is 0x0, which disables the Target device as an Interrupter and, consequently, as a SwiftTools Target. In previous releases of PNCFG, the Interrupt level was hard-coded to IRQ3 and this field was not on the menu.

**Configure-Add** (continued)

**Interrupt Vector** Set this field for an Interrupt vector which is NOT used by any other devices in your VME system. Consult the documentation provided with the cards in your VME cage to determine which vectors they use (this information may also often be found in device driver header files). The default value for this field is 0x00.

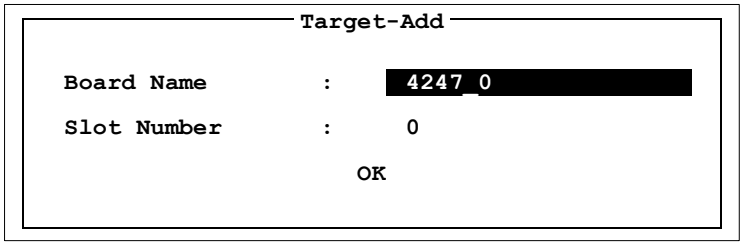
**JTAG Debugging** Enables JTAG debugging support (currently only supported with Pentek Models 4270 and 4285).

**OK** When all the above options have been set to match the required Target Baseboard configuration, this field is used to clear the pop-up window and add the Target's name to the list in the Target (right) window. Press **<return>** when this field is highlighted, or click on the field with the left mouse button. Selecting **OK** does NOT save the Node configuration to a file. That is accomplished with the Configure-Save command. In all the previous cases, an entry in any field can be terminated simply by moving the cursor away from the field. Except for **OK**, it is not necessary to press the **<return>** key in any field.

## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.6 PNCFG - Configure Menu (continued)

After a baseboard configuration has been defined, additional Target devices (either VME boards, additional baseboards or MIX modules attached to defined baseboards) may be added to the Node. When the Configure-Add option is selected after a baseboard is configured, the Board List window is displayed again. If an additional baseboard is selected from this window, a window similar to the one described above appears, and the same procedures are followed. If a MIX module is selected, a window appears listing the defined baseboards. Choose the baseboard to which the module will be attached by clicking on its line with the left mouse button or by highlighting its line with the ↑ and  arrow keys and then pressing **<return>**. This displays a shorter Target-Add window, shown on the next page. The only items requested in this window are:




**Board Name** Enter the name you wish SwiftTools to use when referring to this Target board. Board names may be a maximum of eight characters long. This field has no default. If no board name is entered, the Target-Add operation fails.

**Slot Number** Enter the MIX bus "slot" number (i.e. board position in the stack) for this board. The default is 0, and the only other acceptable entries into this field are 1 or 2. The list of acceptable entries may be cycled through by pressing the space bar or by clicking the left mouse button in the field.

**OK** When the above options have been set to meet your required Target Module configuration, this field is used to clear the pop-up window and add the Target's name to the list in the Target (right) window. Press **<return>** when this field is highlighted, or click on the field with the left mouse button.

**SHARCPAC Processors** When using the Model 4288, it indicates how many, if any, processors are installed on the SHARCPAC.

## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.6 PNCFG - Configure Menu (continued)

**Configure-Edit** This option displays a window showing all Target boards in the selected node. Select the board to be edited by clicking on its line with the left mouse button, or by highlighting its line with the ↑ and arrow keys and then pressing **<return>**. The Target-Edit sub-menu appears for editing. All commands are the same as Target-Add.

**Configure-Remove** Removes the selected board from the Target list. When this option is selected, a window appears showing all Target boards in the selected node. Select the board to be removed by clicking on its line with the left mouse button, or by highlighting its line with the ↑ and arrow keys and then pressing **<return>**. A baseboard may not be removed until all MIX modules associated with it have been removed first.

**Configure-Update** This menu option reads and lists the Target configuration files from the Node.

**Configure-Show** Displays a window containing a list of all Target boards in the selected Node. An individual Target may be selected from the list and its configuration may be displayed by clicking on its line with the left mouse button, or by highlighting its line with the ↑ and arrow keys and then pressing **<return>**. The window may be cleared by pressing **<return>** or clicking on it with the left mouse button.

### 3.2.7 PNCFG - Memory Map Menu

PNCFG creates a memory map for each VMEbus address space as they are scanned using the Memory-Scan option or as boards are added to a target's configuration. This option is only enabled after a node has been connected.

**Memory Map-A16/D16** Displays a memory map of A16/D16 space.

**Memory Map-A16/D32** Displays a memory map of A16/D32 space.

**Memory Map-A24/D16** Displays a memory map of A24/D16 space.

**Memory Map-A24/D32** Displays a memory map of A24/D32 space.

**Memory Map-A32/D16** Displays a memory map of A32/D16 space.

**Memory Map-A32/D32** Displays a memory map of A32/D32 space.

## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.8 PNCFG - Memory Scan Menu

PNCFG scans each VMEbus address space for boards occupying address ranges. This is useful for determining occupied address before attempting to add a board using Configure-Add or for confirming user set base addresses on each board. After an address space is scanned the results are saved and can be re-displayed with the Memory-Map option.

**Memory Scan-A16/D16.** Scans A16/D16 address space for occupied ranges.

**Memory Scan-A16/D32.** Scans A16/D32 address space for occupied ranges.

**Memory Scan-A24/D16.** Scans A24/D16 address space for occupied ranges.

**Memory Scan-A24/D32.** Scans A24/D32 address space for occupied ranges.

**Memory Scan-A32/D16.** Scans A32/D16 address space for occupied ranges.

**Memory Scan-A32/D32.** Scans A32/D32 address space for occupied ranges.

**NOTE:** The Memory Scan feature scans ALL locations in the address space. This may cause unpredictable behavior when some locations are accessed on non-Pentek boards.

#### 3.2.8.1 Pentek Model 4285 Additional Configuration Information

When setting up Pentek's Model 4285 the configuration window appears with an address. Since the Model 4285 can contain up to eight 'C40 processors, this address changes depending on the configuration. The table below contains some sample configurations of addresses. To determine the proper address for your 4285 board, look at the LED's on the 4285 front panel. Put an X into the table for each illuminated LED. Treat each X as a binary one, and convert it to Hexadecimal to determine the address.

**Table 3-1: Model 4285 'C40 Address Configuration Map**

| H | G | F | E | D | C | B | A | Address |
|---|---|---|---|---|---|---|---|---|
| - | - | - | X | - | - | - | X | 0x11 |
| - | - | X | X | - | - | X | X | 0x33 |
| - | X | X | X | - | X | X | X | 0x77 |
| X | X | X | X | X | X | X | X | 0xFF |
| First hex digit | | | | Second hex digit | | | | |
| - = No 'C40 Processor x = 'C40 Processor present | | | | | | | | |

## 3.2 PNCFG - Target Configuration Program (continued)

### 3.2.8 PNCFG - Memory Scan Menu (continued)

#### 3.2.8.2 Pentek Model 4290/4291 Configuration Information

Like the Model 4285, when setting up Pentek's Model 4290/4291 the configuration window appears with an address. The Model 4290/4291 can contain up to four 'C6x processors, so this address changes depending on the configuration. Table 3-2, below, contains some sample configurations of addresses. To determine the proper address for your Model 4290/4291 board, look at the LED's on the front panel. Put an X into the table for each illuminated LED. Treat each X as a binary one, and convert it to Hexidecimal to determine the address.

**Table 3-2: Model 4290/4291 'C6x Address Configuration Map**

| D | C | B | A | Address |
|---|---|---|---|---|
| - | - | - | X | 0x1 |
| - | - | X | X | 0x3 |
| - | X | X | X | 0x7 |
| X | X | X | X | 0xF |

- = No 'C6x Processor  x = 'C6x Processor present

## 3.3 PNLOAD - COFF Upload Utility Program

Existing COFF (Common Object File Format) object files may be directly uploaded to and executed on a specified Target device from the SunOS/UNIX command line using the PNLOAD utility. The command syntax is as follows:

```
# pnload [-lrvqdnbp] <node-name> <board-name> <processor-number>
<coff-file>
```

where: **l**, **r**, **v**, **q**, **d**, **n**, **b**, and **p** are command line options described below

**<node-name>**, **<board-name>**, and **<processor-number>** have been assigned to the desired Target using PNCFG.

**<coff-file>** is the pathname of the COFF file to be uploaded and executed.

## 3.3 PNLOAD - COFF Upload Utility Program (continued)

### 3.3.1 PNLOAD Options

**`-l`** Load only, don't execute.

**`-r`** Run previously loaded code only, don't load.

**`-v`** Verbose mode.

**`-q`** Quiet mode (overrides Verbose mode).

**`-d`** Debug mode, reads back data and compares as it loads.

**`-n`** No reboot before loading.

**`-b`** Don't clear the **`.bss`** section (usually used for uninitialized data).

**`-p`** Load the object file found in the directory specified by the Project name, i.e.:

```
pnload -p node_1 board_1 0 demo
```

This command uses the name of the project instead of the COFF file. It loads and executes the file **`demo.xxx`**, which is found in the project directory named demo, and where ***`xxx`*** is automatically set to the type of target specified by **`board_1`**. If **`board_1`** is a 4270 the file **`demo.x70`** is loaded, if the board is a 4284, the file **`demo.x84`** is loaded.

## 3.4 SNTEST - SwiftNet Test Utility

SNTEST is a diagnostic utility that loads and executes a small pre-compiled test program on a processor of a DSP target board. The syntax of SNTEST is as follows:

```
sntest <node-name> <board-name> <processor-number>
```

**`<node-name>`**, **`<board-name>`**, and **`<processor-number>`** must have been previously assigned to the desired Target using PNCFG.

SNTEST displays a count of each successful transmission, and runs until interrupted by the user with **`^C`**.

## 3.5 SNINFO - Target Information Utility

SNINFO displays information about a specific Target board. The command syntax is as follows:

```
sninfo [-f] <node-name> <board-name>
```

where:

**<node-name>** and **<board-name>** are the names assigned to the desired Target using PNCFG.

**-f** Used to find SwiftNet nodes and print out the revision.

**NOTE:** When using the **-f** option, the network must support reverse DNS lookup in order for the names of the nodes to appear, otherwise only the IP numbers are displayed.

# Chapter 4: The SwiftNet Client Application Programming Interface

## 4.1 General Information

The SwiftNet Client Application Programming Interface (API) provides a suite of functions that simplify the process of sending SwiftNet signals to SwiftNet devices. A SwiftNet device may be a ***real*** device such as a TMS320C30 or 'C40 DSP board, or a ***virtual*** device created on your SPARCstation or VxWorks processor. This API is available for Solaris 2.x (SPARC), HP-UX 9.0x/10.10 (HP 9000 Series 700), VxWorks 5.1 (680x0, SPARC), and Pentek DSP boards. In all the function names below, the prefix **`pnkc`** is an acronym for Pentek Network Kernel Client.

When using the Swiftnet API, only one Swiftnet Signal can be sent to a SwiftNet device simultaneously. So for example, if a signal is being sent to an open SwiftNet device, using **`PNKC_SEND`** or **`PNKC_SENDEX`**, it is necessary to wait for it to complete before calling **`PNKC_SEND`**, **`PNKC_SENDEX`**, **`PLOG`**, or **`PLOGF`**.

When compiling code that uses this API on the host, one of the following defines must be set: **`PENTEK_SOLARIS`**, **`PENTEK_OSF1`**, or **`PENTEK_HPUX`**. In addition, the library **`libsocket`** must be linked in. Previous code based on the SwiftNet API needs to be re-compiled using the SwiftNet 3.4 header files and libraries. It is also recommended that you change previous occurrences in your code of **`u_long`** types to **`u_int_32`**.

A new function, added in SwiftNet 3.4, **pnkc_sendEx()** enables the use a data buffer larger than 1024 longwords.

**Table 4-1: SwiftNet Client API Functions Summary**

| Function | Description | See Section |
|---|---|---|
| **pnkc_init** | Initialize the SwiftNet client API | Section 4.3 |
| **pnkc_open** | Open a connection to a SwiftNet device | Section 4.4 |
| **pnkc_info** | Get information about an opened SwiftNet device | Section 4.5 |
| **pnkc_send** | Send a SwiftNet signal to an open device | Section 4.6 |
| **pnkc_sendEx** | Send a SwiftNet signal to an open device. | Section 4.7 |
| **pnkc_free** | Free memory allocated by pnkc_send() | Section 4.8 |
| **pnkc_close** | Close a connection to a SwiftNet device | Section 4.9 |
| **plog** | Allow DSP Target to output a message to the Server Console | Section 4.10 |
| **plogf** | Allow DSP Target to output a message to the Server Console | Section 4.11 |

Prototypes for SwiftNet Client API functions are found in **`pnkcapi.h`**.

## 4.1 General Information (continued)

The SwiftNet Client API consists of the following files:

Solaris 2.x API
**$SWIFTNET_HOME/solaris/include/*.h**
**$SWIFTNET_HOME/solaris/[libl,libn]/libswiftnet.so.2.x**

HP-UX 9.0x/10.10 API
**$SWIFTNET_HOME/hp9700/include/*.h**
**$SWIFTNET_HOME/hp9700/[libl,libn]/libswiftnet.sl**

VxWorks 5.1 API - 68020 & 68030
**$SWIFTNET_HOME/68020-5.1/include/*.h**
**$SWIFTNET_HOME/68020-5.1/lib/swiftnet.a**

VxWorks 5.1 API - 68040
**$SWIFTNET_HOME/68040-5.1/include/*.h**
**$SWIFTNET_HOME/68040-5.1/lib/swiftnet.a**

VxWorks 5.1 API - SPARC
**$SWIFTNET_HOME/sparc-5.1/include/*.h**
**$SWIFTNET_HOME/sparc-5.1/lib/swiftnet.a**

VxWorks 5.3 API - PowerPC
**$SWIFTNET_HOME/ppc603-vx-5.3/include/*.h**
**$SWIFTNET_HOME/ppc603-vx-5.3/lib/swiftnet.a**

DSP API
**$SWIFTNET_HOME/[c30,c40]/include/*.h**
**$SWIFTNET_HOME/[c30,c40]/lib/[swftnt.lib,swftntr.lib]  (see note)**
**$SWIFTNET_HOME/[4283,4270, etc]/include/*.h**
**$SWIFTNET_HOME/[4283,4270, etc]/lib/[*.lib,*.cmd]**

**NOTE:** There are four libraries to choose from; **swftnt.lib**, **swftntr.lib**, **swftntb.lib**, and **swftntbr.lib**, depending on the memory module you want to use. No suffix - small memory model, **r** - register passing parameter model, **b** - big memory model, and **br** - big memory and register passing parameter model.

## 4.2 Allocating Shared Memory: `shMalloc`

```
ptr=shMalloc(size, sec)
```

***size*** = buffer amount allocated in longwords.

***sec*** = section of memory to use. There can be up to four sections of memory defined on the DSP. If **sec=0**, then section 1 is used as the starting point and continues to the next segment of memory on the DSP if needed.

To allocate shared memory use **shmalloc()**. Memory boundaries are defined in the linker file with **_shmem1_base** & **_shmem1_top**.

## 4.2 Allocating Shared Memory (continued)

### 4.2.1 Target Routine: `shmint`

**`shmint(mode)`**

Zeros out the shared memory area. This routine needs to be used prior to allocating shared memory.

**Mode:** 0: Zero out shared memory only from cold boots.
1: Zero out shared memory unconditionally.

In most cases shared memory is zeroed from a cold boot. However, it may be necessary to use mode 1 when debugging, or in other special conditions.

### 4.2.2 Target Routine: `shmFree`

**`shmFree(ptr)`**

Releases the buffer allocated with the shMalloc routine.

### 4.2.3 Symbol: `_shmemX_base`

**`_shmem1_base=`** **`_shmem2_base=`**
**`_shmem3_base=`** **`_shmem4_base=`**

DSP memory base address of sections 1, 2, 3, and 4. You can define up to four sections of memory. **`shMalloc`** stops when any **`shmemX_base`** amount is set to equal zero (i.e. **`_shmem2_base=0`**).

### 4.2.4 Symbol: _shmemX_top

**`_shmem1_top=`** **`_shmem2_top=`**
**`_shmem3_top=`** **`_shmem4_top=`**

DSP memory ending address range for each section. Unused sections should be set to 0 (zero).

## 4.3 PNKC_INIT

**STATUS pnkc_init()**

The **pnkc_init()** function initializes the SwiftNet client API. It must be called before any other function in the SwiftNet client API.

**Parameters** None

**Return Values** Returns **OK** on success, **ERROR** on failure

**See Also** **pnkc_open**

## 4.4 PNKC_OPEN

```
STATUS pnkc_open(pnkcHndl, nodeName, deviceName)
PNKCAPI_PTR  pnkcHndl;
char         *nodeName;
char         *deviceName;
```

The **pnkc_open()** function establishes a connection with a SwiftNet device.

**Parameters**

***pcHndl***
Pointer to a **PNKCAPI_TYPE** structure; used as a handle when calling other SwiftNet client API functions

***nodeName***
Node name as defined by PNCFG

***deviceName***
Device name as defined by PNCFG

**Return Values** Returns **OK** on success, and **ERROR** or **RPC_ERROR** on failure.

**Comments** The **PNKCAPI_TYPE** structure members should be considered hidden and should not be accessed by user programs. The **pnkc_info()** routine may be used to obtain information about a device once it is opened.

**See Also** **pnkc_init, pnkc_info, pnkc_send, pnkc_close**

## 4.5 PNKC_INFO

**PNK_DEVHDR_PTR pnkc_info(*pnkcHndl*)**
**PNKCAPI_PTR *pnkcHndl*;**

The **pnkc_info()** function returns a pointer to the device header structure of the device specified by the handle, ***pnkcHndl***.

**Parameters**

***pnkcHndl***

Pointer to a **PNKCAPI_TYPE** structure; used as a handle when calling other SwiftNet client API functions. The **PNK_DEVICE_TYPE** and **PNK_DEVHDR_TYPE** structures are described below:

**Device Type Structure**

```
struct PNK_DEVICE_TYPE {
   int arch;           /* device architecture     */
   char *parent;       /* parent device name      */
   char *name;         /* device name             */
   struct {
      u_int params_len;/* # of parameters         */
      u_int_32 params_val;/* pointer to parameter
buffer                      */
   } params;
};
typedef struct PNK_DEVICE_TYPE PNK_DEVICE_TYPE;
typedef PNK_DEVICE_TYPE *PNK_DEVICE_PTR;
```

**Device Header Structure**

```
struct PNK_DEVHDR_TYPE {
   char *mach;      /* dev machine (port)
architecture                */
   char*class;      /* device class            */
   u_int_32type;    /* device type             */
   u_int_32 num_ports;/* number of ports on the
device                      */
   u_int_32 buff_size;/* device buffer size      */
   u_int_32 dev_id; /* device ID assigned by node
server                      */
   u_int_32 server; /* server ID               */
   PNK_DEVICE_TYPE device;/* device structure    */
};
typedef struct PNK_DEVHDR_TYPE PNK_DEVHDR_TYPE;
typedef PNK_DEVHDR_TYPE *PNK_DEVHDR_PTR;
```

**Return Values**

Returns a pointer to the device header structure on success, **NULL** on failure

**Comments**

The **PNK_DEVICE_TYPE** and **PNK_DEVHDR_TYPE** data structures have changed starting with release 3.0. Normal use of the API should only require minor changes in calls to **pnkd_createDev()**.

This function is not yet available for DSP targets.

**See Also**

**pnkc_init, pnkc_open**

## 4.6 PNKC_SEND

```
STATUS pnkc_send(pnkcHndl, portNum, sig, ret)
PNKCAPI_PTR       pnkcHndl;
u_int_32          portNum;
PNK_SIGNAL_PTR    sig;
PNK_RETURN_PTR    ret;
```

The **pnkc_send()** function sends a SwiftNet signal to a previously opened SwiftNet device.

**Parameters**

***pnkcHndl***

The argument ***pnkcHndl*** is the handle from a **pnkc_open()** call.

***portNum***

The signal is routed to the device port specified by ***portNum***. For DSP devices, each port represents a different processor on a board.

***sig***

The ***sig*** argument is a pointer to a SwiftNet signal structure (described below).

***ret***

***ret*** is a pointer to a SwiftNet return structure (described below).

**Sig Structure**

```
struct PNK_SIGNAL_TYPE {
   u_int_32 num;       /* the signal number       */
   struct {
      u_int args_len;  /* argument buffer length  */
      u_int_32 *args_val; /* argument buffer pointer
*/
   } args;
   struct {
      u_int data_len;  /* data buffer length      */
      u_int_32 *data_val; /* data buffer pointer  */
   } data;
   u_int_32 timeout_ticks;  /* # of ticks before a
                               timeout         */
   u_int_32 tick_ulen;/* tick length given in uS */
};
typedef struct PNK_SIGNAL_TYPE PNK_SIGNAL_TYPE;
typedef PNK_SIGNAL_TYPE *PNK_SIGNAL_PTR;
```

The signal structure pointed to by ***sig*** must be fully initialized before the **pnkc_send()** call. The **pnkc_send()** function does not modify any contents of this structure.

The ***num*** member is the SwiftNet signal number (**0x000xFF**).

## 4.6 **PNKC_SEND** (continued)

**Sig Structure** (continued)

The args substructure is used to specify a buffer of arguments. The **args.args_len** member is the size of the buffer (in units of unsigned longs), and the **args.args_val** member is a pointer to the argument buffer. Similarly, the data substructure is used to specify a buffer of data.

The use of two buffers is purely a convenience to the user. The SwiftNet API does not interpret them in any way. The only constraint is that the sum of the sizes of the two buffers must be less than or equal to the device buffer size obtained using **pnkc_info()**. SwiftNet protocol guarantees a buffer size of at least 1024 unsigned longs.

If a buffer is not to be used, set the buffer length to zero and the buffer pointer to **NULL**.

The **timeout_ticks** member specifies the number of ticks to wait before timing-out, and the **tick_ulen** member defines the length of each tick in microseconds. If a signal timeout occurs, **pnkc_send()** will return **ERROR**.

Since the system clocks responsible for calculating timeout delays vary, **timeout_ticks** and **tick_ulen** may be translated. If **tick_ulen** is less than the clock period, the clock period is used as the actual tick length. Otherwise, the actual tick length is calculated as the largest multiple of the clock period less than or equal to **tick_ulen**. Furthermore, the actual number of ticks is adjusted such that the total timeout period is the smallest multiple of the actual tick length that is equal to or greater than the product of the original **timeout_ticks** and **tick_ulen**.

Specifying zero for **timeout_ticks** causes **pnkc_send()** to return without waiting for return data from the device. In this case, the return structure, ***sig***, is left untouched. Specifying timeout_ticks equal to -1 disables timeouts, causing **pnkc_send()** to wait indefinitely until the device returns data. One pitfall is that much of SwiftNet is based on RPC calls which will timeout after about 60 seconds, causing **pnkc_send()** to return an **ERROR**.

**Return Structure**

```
struct PNK_RETURN_TYPE {
   u_int_32 val;          /* return value                */
   struct {
      u_int data_len;     /* return data buffer size     */
      u_int_32 *data_val; /* return data buffer pointer  */
   } data;
};
typedef struct PNK_RETURN_TYPE PNK_RETURN_TYPE;
typedef PNK_RETURN_TYPE *PNK_RETURN_PTR;
```

## 4.6 PNKC_SEND (continued)

**Sig Structure** (continued)

Although the return structure is used to return data from the device, some of its members must be initialized before calling **pnkc_send()**. You have two options; either specify an allocated return buffer and size, or let **pnkc_send()** allocate a buffer of the appropriate size. In the latter case, you may use **pnkc_free()** to de-allocate the returned data buffer.

To specify an allocated buffer, set the **data.data_len** and **data.data_val** members to the buffer size and buffer pointer respectively. To have **pnkc_send()** allocate a buffer, set these members to zero and **NULL** respectively. In either case, **pnkc_send()** sets the **data.data_len** member to the actual size of data returned. This implies that you must re-initialize this value before calling **pnkc_send()** again.

The **val** member is used as a return value from the device, analogous to a function call returning a value.

**Return Values**

Returns **OK** on success, **ERROR** on failure. Failure to send a signal may result from various conditions and there are currently no distinguishing error codes. However, error messages describing the failure are usually printed to **stderr**.

**Comments**

Make sure that the **data.data_len** and **data.data_val** members of the return structure are correctly set before each call to **pnkc_send()**.

**pnkc_send()** will return an **ERROR** due to an RPC timeout after about 60 seconds. This could happen if you specify a total timeout period larger than 60 seconds, or if you disable device timeouts by specifying timeout_ticks equal to -1. If longer delays are expected, or the delay is non-deterministic (such as when waiting for user input), it is recommended that the device log a request and return data immediately. The device can process the request, and then explicitly send a signal back to the requester when done. This method requires a two-way connection, i.e., both ends must be SwiftNet devices (real or virtual) and have a client connection open to the other.

An **ERROR** will be returned by **pnkc_send()** if the device returns a data buffer larger than the size specified in the return structure. This is only true if the user provides an allocated buffer (see above).

The maximum allowable signal length using **pnkc_send** is 1024 longwords. You can use **pnkc_sendEx** to use larger signal lengths.

**See Also**

**pnkc_init, pnkc_open, pnkc_free, pnkc_sendEx**

## 4.7 PNKC_SENDEX

*Target version:* **STATUS pnkc_sendEx(*pnkcHndl*, *portNum*, *sig*, *ret*)**

*Host version:* **STATUS pnkc_sendEx(*pnkcHndl*, *portNum*, *sig*, *ret*, *dpr_args*, *dpr_data*, *dpr_ret*)**

```
PNKCAPI_PTR         pnkcHndl;
u_int_32            portNum;
PNK_SIGNAL_PTR      sig;
PNK_RETURN_PTR      ret;
u_int_32            dpr_args;          /* USED ONLY ON HOST */
u_int_32            dpr_data;          /* USED ONLY ON HOST */
u_int_32            dpr_ret;           /* USED ONLY ON HOST */
```

The **pnkc_sendEx()** function sends a SwiftNet signal to a previously opened SwiftNet device.

The function **pnkc_sendEx()** provides the same function as **pnkc_send** and all information in the **pnkc_send** description applies to **pnkc_sendEx**. However, the transfer of data through SwiftNet is no longer limited to 1024 longwords since data is located in shared memory (dpr).

When using **pnkc_sendEx** on a host, three additional arguments are required; **dpr_args**, **dpr_data**, and **dpr_ret**. When using **pnkc_sendEx** on the target end, the signal structure/return structure must contain pointers to allocated shared memory buffers. The signal structure members (**sig.args.args_val** and **sig.data.data_val**) and return structure member (**ret.data.data_val**) are described in the **pnkc_send** section.

Parameters

***pnkcHndl***

The argument ***pnkcHndl*** is the handle from a **pnkc_open()** call.

***portNum***

The signal is routed to the device port specified by ***portNum***. For DSP devices, each port represents a different processor on a board.

***sig***

The ***sig*** argument is a pointer to a SwiftNet signal structure.

***ret***

***ret*** is a pointer to a SwiftNet return structure.

***dpr_args***

**dpr_args** is the DSP address which points to the shared memory buffer for arguments. You can only used this parameter on the host.

## 4.7 PNKC_SENDEX (continued)

***dpr_data***

**dpr_data** is the DSP address which points to the shared memory buffer for data. You can only used this parameter on the host.

***dpr_ret***

**dpr_ret** is the DSP address which points to the shared memory buffer for return data. You can only used this parameter on the host.

**Sig Structure** The sig Structure is code is identical to **pnkc_send.** See **pnkc_send** for sample code. As previously stated, when using the **pnck_sendEx()** function you are not limited to 1024 Longword buffer. However, you must allocate the buffer prior to using the **pnkc_sendEx()** function using the **shmalloc** routine. **shmalloc** and related routines are explained in Section 4.2.

**Return Values** Returns **OK** on success, **ERROR** on failure.

**See Also** **pnkc_init, pnkc_send**

---

## 4.8 PNKC_FREE

```
STATUS pnkc_free(ret)
PNK_RETURN_PTR ret;
```

The function **pnkc_free()** frees memory previously allocated for return data contained in the structure pointed to by ***ret***. It also sets the structure members **data.data_len** and **data.data_val** to zero and **NULL** respectively. This function may only be used when the memory was allocated by **pnkc_send()**. See **pnkc_send** for details.

**Parameters** ***ret***

***ret*** is a pointer to a SwiftNet return structure.

**Return Values** Returns **OK** on success, **ERROR** on failure.

**See Also** **pnkc_init, pnkc_send**

## 4.9 PNKC_CLOSE

```
void pnkc_close(pnkcHndl)
PNKCAPI_PTR pnkcHndl;
```

The function **pnkc_close()** closes a connection to a SwiftNet device. The argument ***pnkcHndl*** is the handle associated with the **pnkc_open()** call.

**Parameters** ***pnkcHndl***
Pointer to a **PNKCAPI_TYPE** structure; used as a handle when calling other SwiftNet client API functions

**Return Values** This function returns no values.

**See Also** **pnkc_init, pnkc_open**

## 4.10 PLOG

```
int plog (char *s);
```

The function **plog** allows the DSP Target to output a message to the Server Console. The **plog** function is provided with the Swiftnet API and doesn't require running SNIO. Carriage returns are automatically inserted at the end of lines, so **\n** statements are not required. It is also important not to call **plog** from a signal handler.

**Parameters** *None*

**Return Values** No values returned.

**See Also** **plogf**

## 4.11 PLOGF

```
int plogf (char * format, ...);
```

The function **plogf** is allows the DSP Target to output a formatted message to the Server Console. The **plogf** function is provided with the Swiftnet API and is identical to the stdio.h **printf** function except **plogf** doesn't require running SNIO. The **plogf** function is similar to **plog** in that carriage returns are automatically inserted at the end of lines, so **\n** statements are not required, and you should not call **plogf** from a signal handler.

**Parameters** *format*
Format is identical to stdio.h's printf function format parameter .

**Return Values** No values returned.

**See Also** **plog**

# Chapter 5: The SwiftNet Device Application Programming Interface

## 5.1 General Information

The SwiftNet Device Application Programming Interface (API) provides a suite of functions that simplify the connection of Interrupt Service Routines (ISRs) to SwiftNet signals. Additional functions are provided which allow hosts and node processors to create virtual devices for communication within and between card cages. In all the function names below, the prefix **pnkd** is an acronym for Pentek Network Kernel Device.

When compiling code that uses this API on the host, one of the following defines must be set: **PENTEK_SUNOS**, **PENTEK_SOLARIS**, **PENTEK_OSF1**, or **PENTEK_HPUX**. In addition, the library **libsocket** must be linked in.

This API is available for Solaris 2.x (SPARC), HP-UX 9.0x (HP 9000 Series 700), VxWorks 5.1 (680x0, SPARC), and Pentek DSP boards for developing applications that receive SwiftNet signals. For SPARCstations and VxWorks processors, this API creates virtual devices that can receive SwiftNet signals just like real DSP devices such as TMS320C30 and 'C40 boards. Signals are received asynchronously via hardware interrupts on real DSP devices and via UNIX signals for virtual SwiftNet devices.

**Table 5-1: SwiftNet Device API Functions Summary**

| Function | Description | See Section |
|---|---|---|
| **pnkd_init** | Initialize the SwiftNet device API | Section 5.2 |
| **pnkd_createDev** | Create a virtual SwiftNet device | Section 5.3 |
| **pnkd_connect** | Connect a function to a SwiftNet signal | Section 5.4 |
| **pnkd_addr** | Retrieve target information | Section 5.5 |
| **pnkd_signal** | Retrieve an incoming signal | Section 5.6 |
| **pnkd_return** | Acknowledge an incoming signal and return data | Section 5.7 |
| **pnkd_pause** | Wait for a SwiftNet signal | Section 5.8 |
| **pnkd_disable** | Disable incoming SwiftNet signals | Section 5.9 |
| **pnkd_enable** | Enable incoming SwiftNet signals | Section 5.10 |
| **pnkd_wait** | Wait for a SwiftNet signal | Section 5.11 |

Prototypes for SwiftNet Client API functions are found in **pnkdapi.h**.

The SwiftNet Client API consists of the following files:

Solaris 2.x API

```
$SWIFTNET_HOME/solaris/include/*.h
$SWIFTNET_HOME/solaris/[libl,libn]/libswiftnet.so.2.x
```

HP-UX 9.0x/10.10 API

```
$SWIFTNET_HOME/hp9700/include/*.h
$SWIFTNET_HOME/hp9700/[libl,libn]/libswiftnet.sl
```

## 5.1 General Information (continued)

VxWorks 5.1 API - 68020 & 68030

**$SWIFTNET_HOME/68020-5.1/include/*.h**
**$SWIFTNET_HOME/68020-5.1/lib/swiftnet.a**

VxWorks 5.1 API - 68040

**$SWIFTNET_HOME/68040/include/*.h**
**$SWIFTNET_HOME/68040/lib/swiftnet.a**

VxWorks 5.1 API - SPARC

**$SWIFTNET_HOME/68040/include/*.h**
**$SWIFTNET_HOME/68040/lib/swiftnet.a**

VxWorks 5.3 API - PowerPC

**$SWIFTNET_HOME/ppc603-vx-5.3/include/*.h**
**$SWIFTNET_HOME/ppc603-vx-5.3/lib/swiftnet.a**

DSP API

**$SWIFTNET_HOME/[c30,c40]/include/*.h**
**$SWIFTNET_HOME/[c30,c40]/lib/[swftnt.lib,swftntr.lib]  (see note below)**
**$SWIFTNET_HOME/[4283,4270, etc]/include/*.h**
**$SWIFTNET_HOME/[4283,4270, etc]/lib/[*.lib,*.cmd]**

**NOTE:** There are four libraries to choose from; **swftnt.lib**, **swftntr.lib**, **swftntb.lib**, and **swftntbr.lib**, depending on the memory module you want to use. No suffix - small memory model, **r** - register passing parameter model, **b** - big memory model, and **br** - big memory and register passing parameter model.

## 5.2 PNKD_INIT

UNIX and VxWorks

```
STATUS pnkd_init(buffSize)
u_int_32 buffSize;
```

DSP

```
STATUS pnkd_init()
```

The **pnkd_init()** function initializes the SwiftNet device API. It must be called before any other function in the SwiftNet device API.

**Parameters** ***buffSize*** (UNIX and VxWorks versions only)
***buffSize*** is the maximum buffer size for any virtual devices that may be created by **pnkd_createDev()**, and must be at least 1024.

**Return Values** Returns **OK** on success, **ERROR on** failure.

**Comments** **pnkd_init()** did not take an argument in SwiftNet versions 2.1 and earlier. Therefore, if you are upgrading from SwiftNet 2.1 or earlier, you must make this change and re-compile any VxWorks code that uses this API.

**See Also** **pnkd_createDev, pnkd_connect**

## 5.3 PNKD_CREATEDEV

```
STATUS pnkd_createDev(dev)
PNK_DEVICE_PTR dev;
```

The function **pnkd_createDev()** creates a virtual SwiftNet device on a workstation or VxWorks processor.

**Parameters**

***dev***

***dev*** is a pointer to a device structure **PNK_DEVICE_TYPE** that contains the information about the virtual device to be created. This same structure is used for both real and virtual devices.

**Device Type Structure**

```
struct PNK_DEVICE_TYPE {
    char *arch;                 /* device architecture         */
    char *parent;               /* parent device name          */
    char *name;                 /* device name                 */
    struct {
        u_int params_len;       /* # of parameters             */
        u_int_32 params_val;    /* pointer to parameter buffer */
    } params;
};
typedef struct PNK_DEVICE_TYPE PNK_DEVICE_TYPE;
typedef PNK_DEVICE_TYPE *PNK_DEVICE_PTR;
```

The member arch represents the architecture of a device. For real devices this would be the board architecture such as "4270" for the Model 4270 or "4257" for the Model 4257. Since virtual devices are in fact processes, the architecture identifies the program. For example, the architecture for SNIO, the SwiftNet Standard I/O Server is simply "snio". Note that in releases prior to 3.0 this member was an integer. The constant **SWIFTNET_3X** is #defined in the SwiftNet header files and may be used for conditional compilation.

The member parent is the name of the parent device if there is one. For example, a MIX module's parent is its base board. A null string, "", means that there is no parent device, i.e., it is a base board. Virtual devices have no parents, therefore, this field must be set to "".

The name member is the device name that is unique within a particular SwiftNet node.

The params substructure is used for defining any device specific parameters. The **params.params_len** and params.params_val members should be set to the number of unsigned long parameters and the pointer to the parameter buffer respectively. If there are no parameters (typically this is true for virtual devices) these fields must be set to zero and **NULL** respectively.

**Return Values** Returns **OK** on success, **ERROR** on failure.

**See Also** **pnkd_init, pnkd_connect**

## 5.4 PNKD_CONNECT

```
STATUS pnkd_connect (sigNum, newFunc, oldFunc)
u_int_32     sigNum;
FUNCPTR      newFunc;
FUNCPTR      *oldFunc;
```

The function **pnkd_connect()** connects a function to a SwiftNet signal.

**Parameters**

***sigNum***

The SwiftNet signal number, from the range 0-255, to which the function ***newFunc*** is connected.

***newFunc***

***newFunc*** is a SwiftNet signal handler function. It is called as an ISR (Interrupt Service Routine) on real SwiftNet devices. Normal C functions cannot be ISRs because they do not save and restore the entire processor state. Section 4.6 of the 1991 edition of the ***TMS320 Floating-Point DSP Optimizing C Compiler*** user's guide describes the naming convention used to create special C functions that can be used as ISRs (The form is **c_int*XX***, where ***XX*** is any two digit number. This form notifies the compiler that the processor state must be saved (with pushes) before executing the function, and restored (with pops) before returning). For virtual SwiftNet devices, newFunc is called from a UNIX signal handler and has no naming restrictions.

If ***newFunc*** is equal to **NULL**, the corresponding SwiftNet signal is unconnected. An error will be returned to any client trying to send a signal to a device that has that signal unconnected.

***oldFunc***

***S***et to the previous connected handler if ***oldFunc*** is not equal to **NULL**.

**Return Values** Returns **OK** on success, **ERROR** if ***sigNum*** is out of range.

**Comments** For real devices, signals 0x80-0xFF are reserved for system use. Users should only connect functions to signals 0x00-0x7F on real devices.

**See Also** **pnkd_init, pnkd_signal, pnkd_return, pnkd_pause, pnkd_disable, pnkd_enable**

## 5.5 PNKD_ADDR

```
STATUS pnkd_addr(nodeName, devName, portNum)
char        *nodeName;
char        *devName;
u_int_32    *portNum;
```

This function returns the SwiftNet node name, device (board) name, and processor number via the passed pointers.

**Parameters**

***nodeName***
Pointer to a string that will hold the node name.

***devName***
Pointer to a string that will hold the device name.

***portNum***
Pointer to a unsigned long that will hold the port (processor) number.

**Return Values** Returns **OK** or **ERROR**.

## 5.6 PNKD_SIGNAL

**PNK_SIGNAL_PTR pnkd_signal()**

The function **pnkd_signal()** retrieves the incoming signal when called from the handler function. **W**hen a signal is received, all signals are disabled and the appropriate handler function is invoked. Signals remain disabled until the handler returns. The contents of the signal argument and data buffers are valid until **pnkd_return()** is called.

**Parameters** None.

**Return Values** Returns a pointer to the incoming signal structure, **NULL** if no signal is pending.

**See Also** **pnkdapi, pnkd_init, pnkd_connect, pnkd_return**

## 5.7 PNKD_RETURN

```
STATUS pnkd_return(ret)
PNK_RETURN_PTR ret;
```

The **pnkd_return()** function acknowledges an incoming signal and returns data to the client. The incoming signal data obtained with **pnkd_signal()** is no longer valid after this call.

**Parameters** ***ret***

A pointer to a **PNK_RETURN_TYPE** structure. This function must be called from every SwiftNet signal handler before it returns.

**Return Values** Returns **OK** on success, **ERROR** on failure.

**See Also** **pnkd_init, pnkd_signal**

## 5.8 PNKD_PAUSE

```
void pnkd_pause()
```

The function **pnkd_pause()** enables SwiftNet signals and waits until a signal has been processed before returning. Signal are disabled when **pnkd_pause()** returns.

This function differs from **pnkd_wait()** on DSPs in that it will return immediately after the SwiftNet signal's ISR returns. It does not wait for the node controller to finish accessing shared memory.

**Parameters** None.

**Parameters** This function returns no values.

**Comments** For DSPs, **pnkd_pause()** returns after a SwiftNet signal's ISR has returned. The node controller reads return data from shared memory after **pnkd_return()** has been called by the ISR. It is possible that this shared memory access is still going on when **pnkd_pause()** returns. If the DSP application accesses shared memory immediately following **pnkd_pause()** such that it locks out the node controller (this is dependent on the DSP board being used), bus timeouts can occur.

The function **pnkd_wait()** may be used to avoid this problem. **pnkd_wait()** will not return until the shared memory access is complete (see **pnkd_wait()**).

**See Also** **pnkd_init, pnkd_disable, pnkd_enable**

## 5.9 PNKD_DISABLE

**`void pnkd_disable()`**

The function **`pnkd_disable()`** disables all incoming SwiftNet signals. Any pending signal will not be received until signals are re-enabled with **`pnkd_enable()`**. Although the client will not receive an error immediately as is the case when a particular signal is unconnected, it may eventually timeout and cause an error.

**Parameters** None.

**Return Values** This function returns no values.

**Comments** Since disabling signals for too long may cause timeouts, it is strongly recommended that signals only be disabled for very short periods of time during critical regions of code.

SwiftTools uses SwiftNet signals to accomplish debugging. Therefore, SwiftTools will be unable to interact with a device while signals are disabled.

Signals are enabled by default on real DSP devices.

**See Also** **`pnkd_init, pnkd_pause, pnkd_enable`**

## 5.10 PNKD_ENABLE

**`void pnkd_enable()`**

The function **`pnkd_enable()`** enables all incoming SwiftNet signals.

**Parameters** None.

**Return Values** This function returns no values.

**Comments** Signals are enabled by default on real DSP devices.

**See Also** **`pnkd_init, pnkd_pause, pnkd_disable`**

## 5.11 PNKD_WAIT

**`void pnkd_wait()`**

The function **`pnkd_wait()`** enables SwiftNet signals and waits until a signal has been processed before returning. Signals are disabled when **`pnkd_wait()`** returns. This function differs from **`pnkd_pause()`** in that it will not return until the shared memory accesses by the node controller are complete. This avoids possible bus timeouts when both the node controller and the DSP are trying to access shared memory at the same time.

**Parameters** None.

**Return Values** This function returns no values.

**See Also** **`pnkd_init, pnkd_pause, pnkd_disable, pnkd_enable`**

***This page is intentionally blank***

# *Chapter 6: Standard I/O*

## 6.1 General Information

The SwiftNet File I/O Server allows the programmer to use familiar standard C I/O functions, such as **printf**, **scanf**, **fopen**, etc., on Pentek DSP target processors. All I/O requests are transferred to a host computer running the Swiftnet I/O server. The package consists of a host server executable (**snio** or **sniox**), two libraries (**snio.lib** or **sniox.lib,** and **snstdio.lib**), and two header files (**snio.h** and **snstdio.h**).

The **snio** library contains system level I/O functions, which are similar to those found UNIX. The **snstdio** library contains buffered standard I/O functions.

**sniox** and **sniox.lib** use the extended SwiftNet API functions, otherwise they are identical to **snio** and **snio.lib.** Throughout this chapter where **snio** is mentioned, **sniox** can be substituted if SwiftNet's extended API is used. It should also be noted that **sniox** uses the same header file as **snio** (**snio.h**).

**NOTE:** There are eight (8) different libraries that can be used depending on the memory model and SwiftNet API function desired. Both **snio.lib** and **sniox.lib** can have four different memory models. No suffix - small memory model, **r** - register passing parameter model (**snior.lib**), **b** - big memory model (**sniob.lib**), and **br** - big memory model and register passing parameter model (**sniobr.lib**).

## 6.2 Target Operation

Before using **snio** library functions, you must first call the appropriate initialization function. If you plan to only use the low-level functions found in **snio.lib**, you must call **snio_init()**. If you plan to use the buffered I/O functions found in **snstdio.lib**, you must call **snstdio_init()**. You do not need to call **snio_init()** if you have called **snstdio_init()**.

The parameter passed to **snstdio_init()** depends on the method used to load programs. When using another program to load the file (i.e., PNLOAD, SwiftTools, or Code Composer), the parameter passed to **snstdio_init()** should be **HOSTNAME** in quotes: **snstdio_init("HOSTNAME")**. In this case, SNIO should be called with the following parameters: **snio node board proc#**. When using SNIO stand-alone, the parameter passed to **snstdio_init()** should be **NULL**: **snstdio_init(NULL)**. In stand-alone operation, SNIO should be called with the following parameters: **snio node board proc# filename**.

## 6.2 Target Operation (continued)

The server must be properly shut down prior to the termination of your program. This can be accomplished by calling **snio_shutdown()** when using only low-level functions, or **snstdio_shutdown()** when using buffered I/O. The appropriate shutdown function may be added to the **atexit()** list, as illustrated by the following example:

```
#include <stdlib.h>
#include <snstdio.h>
main()
{
          if (snstdio_init(NULL) != OK)
          {
                plog("snstdio_init() failed\n");
                return(-1);
          }
          atexit((void (*)())snstdio_shutdown);
          printf("Hello world!\n");
}
```

## 6.3 Host Operation

The host portion of the standard I/O package consists of a single executable: **snio**. It can be used to both load the target executable, and act as a server for processing I/O requests from a DSP board.

### 6.3.1 SNIO Usage

The usage for snio is as follows:

```
snio [-p] <node-name> <dev-name> <proc-number> [coff-file]
```

**node-name** network node name of the system that contains the Pentek DSP board

**dev-name** board name of the DSP, as defined in PNCFG

**proc-number** processor number of the DSP board, starting with 0 (should be set to 0 for a single processor board)

**coff-file** filename of the DSP executable

**-p** This option uses the name of the project instead of the COFF file. It will load and execute the file **projectname.*xxx*** which is found in the project directory named **projectname** and where ***xxx*** is automatically set to the type of target specified by **board_1**. If **board_1** is a 4270 the file **projectname.x70** will be loaded, the board is a 4284 the file loaded will be **projectname.x84**, etc.

### 6.3.2 SNIO Library

The following is a listing of the **snio** library. The prototypes and macros for this library are in **snio.h**.

**int chdir(char *path)** Change current directory to path. Returns 0 on success, -1 on failure.

**int close(int fd)** Close an open file. Returns 0 on success, -1 on failure.

**char *getenv(char *var)** Returns contents of an environment variable on the host system. **NULL** is returned on failure.

**char *getcwd(char *buf, int size)** Returns a pointer to a string containing current directory path . If **buf** is **NULL**, **size** bytes will be malloc'd to hold the directory path; otherwise, **buf** must be at least **size** bytes long. **NULL** is returned on error.

**char *getwd (char pathname)** Alternate form of **getcwd()**. Copies the current working directory path to pathname. **NULL** is returned on error.

**int lseek(int fd, int offset, int whence)** Sets the seek pointer associated with the open file. **whence** must be either set to **SEEK_SET**, **SEEK_CUR**, or **SEEK_END**. If **whence** is **SEEK_SET**, the seek pointer is set to **offset** from the beginning of the file. If **whence** is **SEEK_CUR**, the seek pointer is set to **offset** from the current file position. If whence is **SEEK_END**, the seek pointer is set to the file size plus **offset**. On success, the current file location is returned; on failure, -1 is returned.

**int isatty(fd)** Returns 1 if the specified file descriptor (**fd**) is associated with a terminal device; otherwise, 0 is returned.

## 6.3 Host Operation (continued)

### 6.3.2 SNIO Usage (continued)

**`int ioctl(int fd, int request, caddr_t arg)`**

Performs a special function on an open file. Currently, ioctl only supports two function requests: **`SNIO_IOCTL_SET_DATSIZ`**, and **`SNIO_IOCTL_GET_DATSIZ`**.

**`SNIO_IOCTL_SET_DATSIZ`** takes an argument of either 8, or 32. It switches the I/O data size, of the specified path, to either 8 or 32 bits. 8 Bit data transfers are needed for character I/O, while 32 bit data transfers should be used to maintain maximum efficiency for binary data transfers. By default, the data size is set to 8 bits for tty devices; otherwise, it is set to 32 bits. The **`SNIO_IOCTL_GET_DATSIZ`** function request will return the data size of the specified path number. The argument **`arg`** is a pointer to an integer.

**`int open(char *path, int flags, int mode)`**

Open will open a path to a file, or device, and return a path number to that device. The flags argument must contain a single mode flag, and a modifier flag, and can be obtained by ORing the values together.

**Modes:**

| | |
|---|---|
| **`O_RDONLY`** | Open for reading only |
| **`O_WRONLY`** | Open for writing only |
| **`O_RDWR`** | Open for reading and writing |

**Modifiers:**

| | |
|---|---|
| **`O_CREAT`** | Create file |
| **`O_TRUNC`** | Truncate (overwrite) file, if exists |
| **`O_APPEND`** | Append to end of file, if exists |

An optional **`mode`** parameter can be specified, which contains various permission bits set.

## 6.3 Host Operation (continued)

### 6.3.2 SNIO Library (continued)

| | |
|---|---|
| **`int putenv (char *string)`** | Sets an environment variable on the host computer. String must be in the form **`name=value`**. 0 is returned on success, non-zero on failure. |
| **`int read(int fd, char *buf, int n)`** | Reads **`n`** bytes or longwords from the file corresponding to **`fd`**, into **`buf`**. On success, the number of bytes or longwords read is returned. On failure, -1 is returned. |
| **`int rename(char *path1, char *path2)`** | Renames **`path1`** file to **`path2`**. 0 is returned on success, -1 on failure. |
| **`int write(int fd, char *buf, int n)`** | Writes **`n`** bytes or longwords to the path corresponding to **`fd`**, from **`buf`**. On success, the number of bytes or longwords written is returned. On failure, -1 is returned. |
| **`snio_shutdown()`** | Shutdown communications with host. This must be called on termination, unless **`snstdio_shutdown()`** is used instead. It is recommended that **`atexit()`** be used, so that **`snio_shutdown`** is called on exit. |
| **`STATUS snio_init (char *node)`** | Initalizes **`snio`**. This must be called before any system level I/O functions are used, unless **`snio_snstdio`** is used instead. If **`node`** is **`NULL`**, the target will wait to receive the host's node name from the host (this is the normal operating mode for the PC). **`OK`** is returned on success; **`ERROR`** is returned on failure. |
| **`system(char *string)`** | Executes **`string`** command through a shell. |
| **`int unlink(char *path)`** | Deletes file **`path`** from the directory. 0 is returned on success, -1 on failure. |

## 6.3 Host Operation (continued)

### 6.3.3 SNSTDIO Library

The following is a listing of the **snstdio** library. Descriptions of these functions can be found in any C manual (Kernighan & Ritchie). The prototypes and macros for this library are in **snstdio.h**. It should also be noted that **snio.h** is #included in **snstdio.h**.

**NOTE:** When calling any SwiftNet I/O function, ever source file must contain the following declaration:

```
#include <snstdio.h>
```

**SwiftNet I/O Functions**

| | | | | |
|---|---|---|---|---|
| clearerr() | fclose() | fdopen() | feof() | ferror() |
| fflush() | fgetc() | fgets() | fileno() | fopen() |
| fprintf() | fputc() | fputs() | fread() | freopen() |
| fscanf() | fseek() | ftell() | fwrite() | getc() |
| getchar() | gets() | getw() | printf() | putc() |
| putchar() | puts() | putw() | rewind() | scanf() |
| setbuf() | setbuffer() | setlinebuf() | setvbuf() | sprintf() |
| sscanf() | ungetc() | | | |

## 6.3 Host Operation (continued)

### 6.3.3 SNSTDIO Library (continued)

The **`fopen()`** function contained in the snstdio library differs from the K&R standard as follows:

```
fopen(name,mode)
```

Where:

| | |
|---|---|
| **`name`** | name of the file to open. |
| **`mode`** | this argument has been expanded to allow **`fopen()`** to open files for either 8 or 32 bit data. The possible combinations are as follows: |
| **`r`** | read 8 bit data |
| **`w`** | write 8 bit data |
| **`a`** | append 8 bit data |
| **`rb`** | read 32 bit data |
| **`wb`** | write 32 bit data |
| **`ab`** | append 32 bit data |

## 6.3 Host Operation (continued)

### 6.3.4 SNIO/SwiftNet Error Message - No Space left on Device

There are two conditions where the error message **`[pnkd_init] semget: No space left on device`** appears; if the semaphores provided by the operating system needs to be increased, or if SNIO is terminated incorrectly and the operating system doesn't return semaphore and shared memory resources.

#### 6.3.4.1 Adding Semaphores to the Operating System

For Solaris 2.5.x, the kernel does not have to be rebuilt since SunOS 2.x kernel modules are automatically reloaded when needed, making rebuilding the kernel unnecessary.

Below is a listing of parameters for the kernel modules:

| Parameter | Default | Description |
|---|---|---|
| seminfo_semmap | 10 | Number of entries in the semaphore map |
| seminfo_semmni | 10 | Number of semaphore identifiers |
| seminfo_semmns | 60 | Number of semaphores in the system |
| seminfo_semmnu | 30 | Number of undo structures in the system |
| seminfo_semmsl | 25 | Maximum number of semaphores, per id |
| seminfo_semopm | 10 | Maximum number of operations, per semaphore call |
| seminfo_semume | 10 | Maximum number of undo entries, per process |
| seminfo_semvmx | 32767 | Semaphore maximum value |
| seminfo_semaem | 16384 | Adjust on exit maximum value |

To change the values, enter a line in the /etc/system file using the following syntax: **`set semsys:seminfo_varible=value`**

## 6.3 Host Operation (continued)

### 6.3.4 SNIO/SwiftNet Error Message - No Space left on Device (continued)

#### 6.3.4.1 Adding Semaphores to the Operating System (continued)

For example, to increase seminfo_senmap from the default of 10 to 20, add the following line to the **/etc/system** file:

```
set semsys:seminfo_semmap=20
```

After rebooting, the kernel parses the **/etc/system** file during autoconfiguration and overrides the default value for the parameters specified in this file.

#### 6.3.4.2 SNIO Terminated Incorrectly - Recovering Resources

SNIO should be terminated properly using **CTRL-C**, or by calling the exit routine in DSP code. However, in cases where **SNIO** isn't terminated correctly, and the operating system doesn't release the semaphores and shared resources, there are two methods to recover them.

1) Reboot the workstation. All resources are returned to the system after restarting.

2) Without rebooting, run the UNIX OS **ipcs** command, then use the **ipcrm** command to return the resources back to the OS. ipcs displays all the semaphores and memory resources being used.

*Example:*

```
ipcs

IPC status from burns as of Wed Jan 21 09:09:21 1998
T     ID     KEY        MODE       OWNER    GROUP
Message Queues:
Shared Memory:
m    2600 0x00000000 --rw-------      bill     staff
Semaphores:
s   1560 0x00000000 --ra-------     bill    staff
s   1561 0x00000000 --ra-------     bill    staff
```

To remove these resources:

```
ipcrm -m 2600
ipcrm -s 1560
ipcrm -s 1561
```

***This page is intentionally blank***

# *Chapter 7: SwiftNet Operation*

## 7.1 SwiftNet Operation Details

SwiftNet makes use of node controllers which manage communications between the target DSPs and the host. Node controllers are typically MC680X0, SPARC, or PowerPC based processor boards which reside in the card cage and connect to the host through an Ethernet link. A host may also be a node controller as in the case of an embedded VME card cage in the Sun, or a Sun with an SBus-to-VMEbus adapter.

The SwiftNet drivers contain target resident code which controls inter-device communications. When a host process sends a SwiftNet signal to the DSP, the node controller delivers it to the DSP by using shared memory and interrupts. The drivers uses a pseudo interrupt vector table to supplement a standard interrupt vector table because typically only one interrupt is available from the host to the DSP. User supplied ISRs as well as built-in SwiftNet ISRs are loaded into this table. Figure 2 shows the implementation of a typical application using the SwiftNet API.

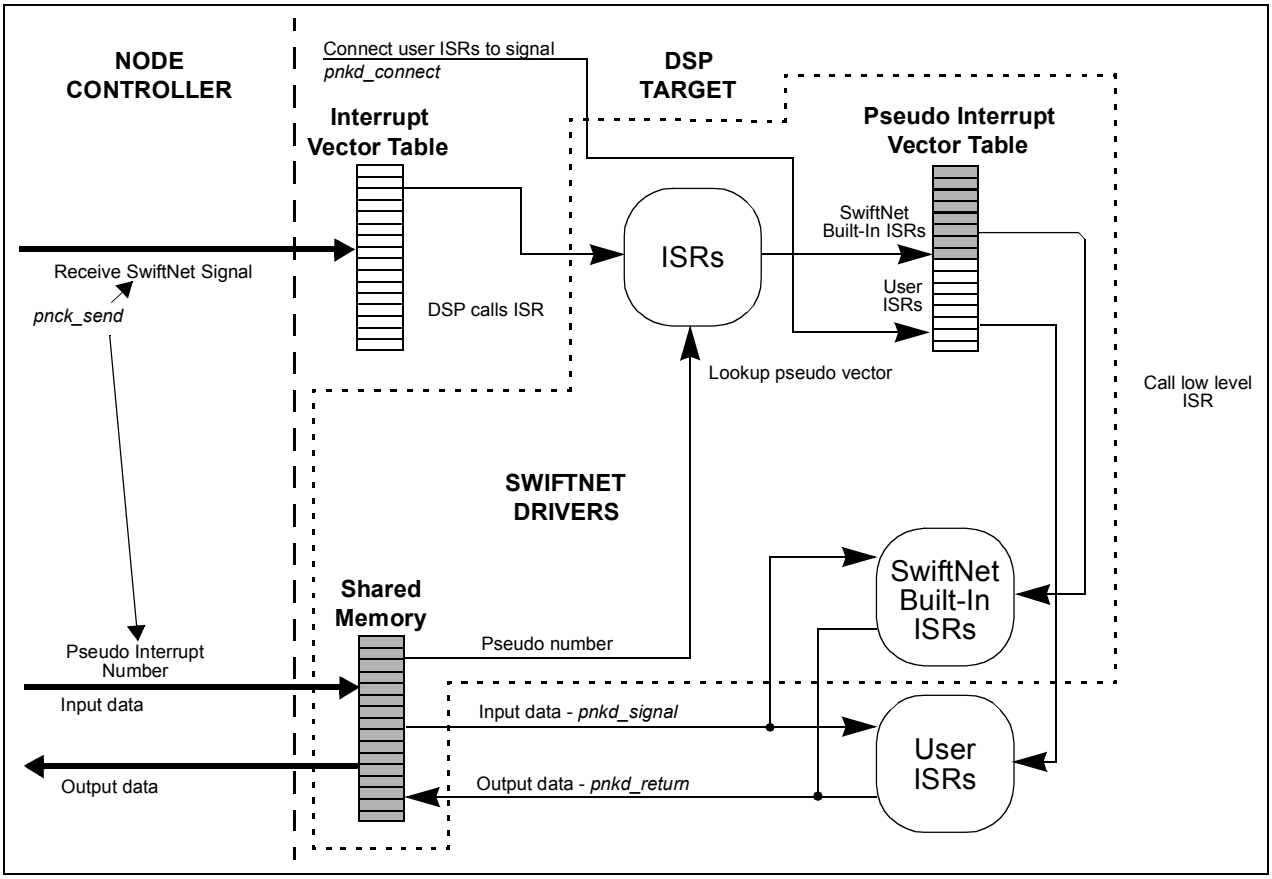



**Figure 7-1: Typical Application Using the SwiftNet API**

## 7.1 SwiftNet Operation Details (continued)

PNCFG is used to configure the target board parameters and load the drivers, initialize the interrupts vector table and initialize the pseudo interrupt vector table with built-in SwiftNet ISRs. **`PNKD_connect`** can be used to load any user supplied ISRs into the pseudo interrupt vector table. A SwiftNet signal is sent from the host using **`PNKC_send`** and is received by the node controller. The node controller passes the signal to the DSP and an ISR is dispatched. The DSP then looks in shared memory for the number of the pseudo interrupt vector which is used to look up an ISR in the pseudo interrupt vector table. The ISR can input and output data by using the API supplied functions **`PNKD_signal`** and **`PNKD_return`** respectively.

The SwiftNet protocol is symmetrical in that it can use the same SwiftNet signals in both host to target and target to host directions. Where the node controller and the SwiftNet kernel accept SwiftNet signals on the target side, the process running on the host is a virtual target which behaves like an actual DSP target. The DSP can send a signal to the host by first interrupting the node controller. The node controller then passes a SwiftNet signal to the virtual target, which uses a software signal that invokes a signal handler. Figure 2 shows how this is implemented when running the **`reverb.c`** demo program.

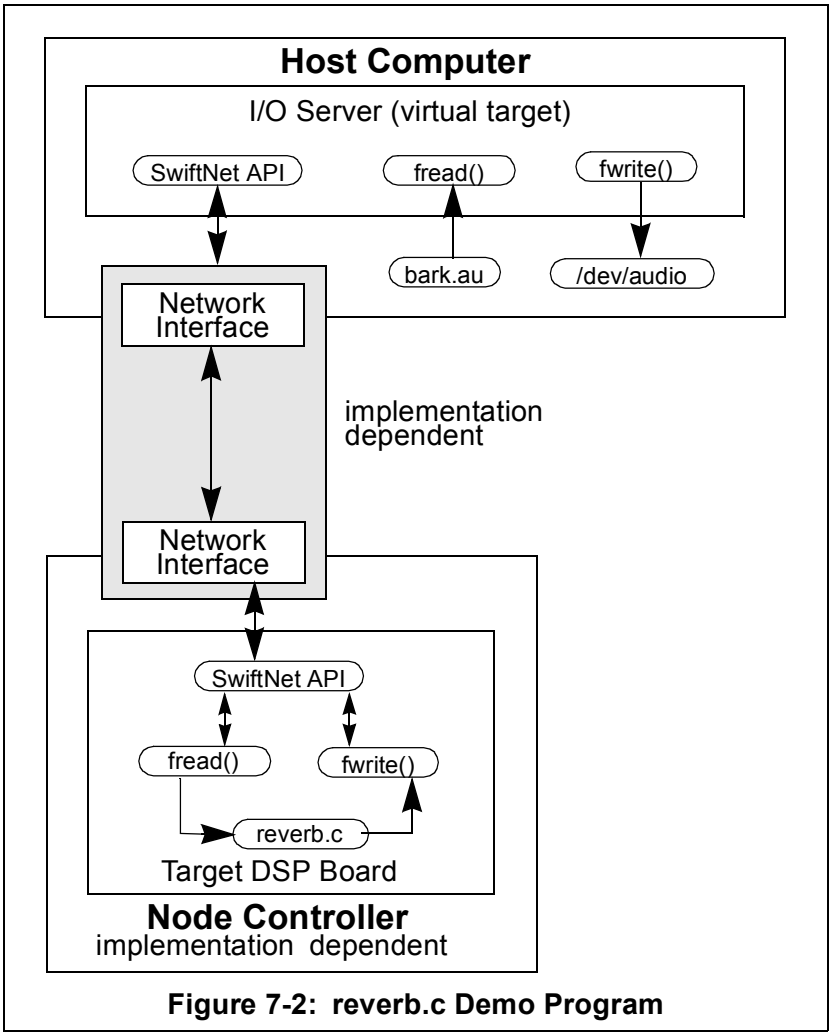



**Figure 7-2: reverb.c Demo Program**

## 7.1 SwiftNet Operation Details (continued)

The **reverb.c** program uses C standard I/O functions to read an audio file (**bark.au**) from the Sun's hard disk. The audio sample is then processed using the DSP to create a reverb effect. The processed audio sample is then written to the sun's audio device **/dev/audio**. At the DSP side, the SwiftNet API is used for standard I/O functions such as **fread()** and **fwrite()**. At the host side a virtual target is created using the API to allow the DSP access to the hosts resources, specifically the file system and the audio device.

SwiftTools, Pentek's software development environment and debugging tools, uses the API to communicate with the DSP target in a similar manner. The fundamental difference is that SwiftTools is not a virtual target. It can initiate a data transfer to or from the DSP target but does not need to respond to target initiated signals.

SwiftNet uses a small portion of the global memory to load the SwiftNet kernel. Specific interrupt signals on the C30/C40/'C6x are also used by SwiftNet. Below is a list of global memory addresses and C30/C40/'C6x interrupt signals which are reserved for the kernel for each Pentek DSP board:

**Table 7-1: Memory Addresses and Interrupt Signals used by SwiftNet**

| Model | Memory | Signal used for VME Interrupt | Signal Used for MIX Interrupt |
|---|---|---|---|
| 4283 | 0x0020 0100-0x0020 1000 | INT0 | INT1, INT2, INT3 |
| 424x | 0x0000 0040-0x0000 0F40 | N/A | INT0 |
| 4280 | 0x0010 0000-0x0010 05FF<br>0x0020 0000-0x0020 1FFF<br>0x0080 9FC1-0x0080 9FFF | INT1 | N/A |
| 4284 | 0xC000 0000-0xC000 0BFF<br>0x80000000-0x80001fff | IIOF1 | IIOF2 |
| 425x | 0x4000 0000-0x4000 0BFF<br>0x8000 0000-0x8000 1FFF | N/A | IIOF1 |
| 4269<br>4270 | 0x4000 0000-0x4000 0BFF<br>0x8000 0000-0x8000 0FFF<br>0x8008 0000-0x8008 0FFF<br>0x8010 0000-0x8010 0FFF<br>0x8018 0000-0x8018 0FFF | IIOF1 | N/A |
| 4285 | 0x4000 0000-0x4000 0BFF<br>0xA000 0000-0xA000 7FFF<br>0x8000 0000-0x8000 0001 | IIOFZ | N/A |
| 4288 | 0x0002 0000-0x0002 0C4F<br>0x0003 0000-0x0003 0CFFF<br>0x00C0 0000-0x00C0 391F | VIRPTI | N/A |
| 4290 | 0x0080 0000-0x0080 C3FF (Global Bus)<br>0x02FF A000-0x02FF FFFF (Local Add.) | EXT_INT6 | N/A |
| 65xx Series | 0x4000 0000-0x4000 0BFF<br>0x8000 0000-0x8000 1FFF | IIOF1 | N/A |

## 7.1 SwiftNet Operation Details (continued)

Please refer to the TI TMS320C30, TMS320C40, and TMS320C62x user's guide for further information on the 'C30, 'C40, and 'C6x signals.

Linker command file are provided for each boards architecture in the files:

**swiftnet-*x.x*/*42xx*/lib/swiftnet.cmd**

where: ***x.x*** is the SwiftNet revision and ***42xx*** is the board type.

Please refer to these files for further information.

### 7.1.1 Installing an Interrupt Handler for the 'C6x Processors

In cases where you want to use SwiftNet and simultaneously service 'C6x interrupts, you need to install a new handler. Since SwiftNet has its own interrupt vector table and sets the Interrupt Service Table Pointer (ISTP), just changing the ISTP would disable SwiftNet's ability to handle the 'C6x mailbox interrupts.

To install a new handler, use the **isr_jump_table** array with the interrupt you want to use. The 'C6x provides 12 interrupts, **INT4** through **INT15**. Refer to the 'C6x documentation for a complete description of each interrupt.

The example below illustrates installing a handler for **INT7**.

```
extern unsigned int isr_jump_table[];
isr_jump_table[7]=handler;
```

# *Appendix A: Demonstration Programs*

## A.1 SwiftNet API Examples

There are four SwiftNet API examples in this section, APITARGET.C, APIHOST.C, APITARGETX.C, and APIHOSTX.C.

### A.1.1 APITARGET.C / APIHOST.C

APITARGET.C: code - page 1, APIHOST.C: code - page 3

These two API examples do not use the extended SwiftNet API features. The program takes keyboard input from the host, sends it to the Swiftnet target, which in turn sends back the information to the host, which displays the keyboard input all using the SwiftNet communication channels.

### A.1.2 APITARGETX.C / APIHOSTX.C

APITARGETX.C: code - page 6, APIHOSTX.C: code - page 10

These two SwiftNet API examples use the extended SwiftNet API features, and perform the same task as the APITARGET.C and APIHOST.C, except instead of using keyboard input, these examples use a data file as the input source.

## A.2 Program Descriptions

The SwiftNet Host Software is shipped with four demo programs; dim.c, dim69.c, dim70.c, and reverb.c, which can be used to confirm an operating connection between the host and the DSP target. These programs are contained in the /swiftnet-x.x/examples directory and can be compiled using SwiftTools by setting the SWIFTTOOLS_PROJ to the examples directory:

```
setenv SWIFTTOOLS_PROJ $SWIFTNET_HOME/examples
```

### A.2.1 DIM.C - DIM.C: code - page 14

The program dim.c (contained in the subdirectory led-demo) drives the front panel LED with a sinusoidally pulse-width modulated signal, causing the brightness of the display to gradually increase and decrease. This program is an endless-loop routine, and executes until you intervene.

### A.2.2 DIM69.C & DIM 70.C - DIM69.C: code - page 17, DIM70.C: code - page 19

The programs dim69.c and dim70.c are written to run on the Pentek 4269 and 4270 VME boards respectively. When downloaded and started in all processors on the board, this program drives the front panel LEDs associated with each of the processors, with a sinusoidally pulse-width modulated signal. The program should be loaded and started on processor 0 last. Again, this is an endless-loop routine, and you must terminateit yourself.

### A.2.3 REVERB.C - REVERB.C: code - page 22

REVERB.C was written for Sun SPARC workstations, but it can easily be modified for Windows 95 and Windows NT.

The program reverb.c utilizes Sun's audio device. The program reads an audio file from the Sun's hard drive, processes it using the DSP to add reverb, and plays it back through the audio device. This program operates on an any of the Pentek DSP boards.

### A.2.4 IPFIFO.C - IPFIFO.C: code - page 26

IPFIFO.C is designed for the Model 4290. It is an example, using SwiftNet, of using the interprocessor FIFOs and interrupts on the 4290. Processor 0 writes a count between 0 and 3 to the IP FIFO. After the almost full threshold is reached, an interrupt is received by processor 1, which reads the data from the IP FIFO, and flashes the corresponding LED.

```
/*****************************************************************************
 * FILE NAME: APITARGET.C
 *
 * VERSION DATA:  $Archive: $
 *                $Revision: $
 *                $Modtime: $
 * DESCRIPTION:
 *
 * Target end of SwiftNet API example
 *
 *****************************************************************************/

/* $History: $
*/

#include "snio.h"
#include "pnkdev.h"
#include "pnkdapi.h"
#include "pnkcapi.h"

#define     APISAMP_SIG 1
#define HOSTNAME  "flanders"        /* Name of machine where host end is
                                       running */
#define DEVICENAME      "apisamp"   /* Virtual device name registered by
                                       host */

#define TICK_ULEN       1000            /* 1 mS ticks                      */
#define TIMEOUT_TICKS   10000           /* Number of ticks, TICK_ULEN long */


u_int_32 _data[256];
u_int_32 _data_len;

c_int01()
{
  PNK_SIGNAL_PTR sig;
  PNK_RETURN_TYPE ret = {0,0,NULL};
  u_int_32   i;

  sig = pnkd_signal();  /* Retreive signal */

  /* Copy data to global variable.  sig is no longer valid after a
     pnkd_return() */

  memcpy(_data,sig->data.data_val,sig->data.data_len);
  _data_len = sig->data.data_len;

  pnkd_return(&ret);    /* Complete signal transaction, send empty return
                           data */
}


main()
{
  PNK_DEVICE_TYPE     dev;
  PNKCAPI_TYPE          pnkc;
  PNK_SIGNAL_TYPE     sig;
  PNK_RETURN_TYPE     ret;

  char      string[256];

  if (pnkc_init() != OK)      /* Initialize client API */
    {
      plog("Error initializing client interface!");
```

```
      exit(1);
    }

  if (pnkd_init() != OK)      /* Initialize device API */
    {
      plog("Error initializing device interface!");
      exit(1);
    }

  /* Connect signal handler, c_int01 to Signal # APISAMP_SIG */

  if (pnkd_connect(APISAMP_SIG, (FUNCPTR)c_int01, NULL) != OK)
    {
      plog("unable to connect signal handler");
      exit(1);
    }

  /* Open up a connection to virtual device existing on host */

  if (pnkc_open(&pnkc, HOSTNAME, DEVICENAME) != OK)
    {
      plog("error opening connection to host");
      exit(1);
    }

  plog("target running");

  while(1)
    {
      pnkd_pause();     /* Wait for SwiftNet signal */

      sig.num = APISAMP_SIG;

      sig.data.data_val = (u_int_32 *)_data;

      sig.data.data_len = _data_len;

      sig.timeout_ticks = TIMEOUT_TICKS;
      sig.tick_ulen = TICK_ULEN;

      /* Send received data back to host */

      pnkc_send(&pnkc,0,&sig,&ret);
    }
}
```

```
/*****************************************************************************
 * FILE NAME: APIHOST.C
 *
 * VERSION DATA:  $Archive: $
 *                $Revision: $
 *                $Modtime: $
 * DESCRIPTION:
 *
 * Host end of SwiftNet API example
 *
 *****************************************************************************/

/* $History: $
*/
#include <stdio.h>

#include "pnkdev.h"
#include "pnkdapi.h"
#include "pnkcapi.h"

#define     APISAMP_SIG 1           /* See pnkdev.h for range of available
                                       USER signals */

#define NODENAME  "sr-200"    /* Name of node where BOARDNAME
                                 resides */

#define BOARDNAME "b1"        /* Name of board running target end
                                 code */

#define TICK_ULEN       1000           /* 1 mS ticks                    */
#define TIMEOUT_TICKS   10000          /* Total wait time =
                                          TICK_ULEN * TIMEOUT_TICKS uS */


char _data[256];


handler()
{
  PNK_SIGNAL_PTR sig;
  PNK_RETURN_TYPE ret = {0,0,NULL};
  u_int_32   i;

  sig = pnkd_signal();  /* Retreive signal */

  /* Copy signal to global variable. sig not valid after pnkd_return */

  memcpy(_data,sig->data.data_val,sig->data.data_len * 4);

  pnkd_return(&ret);    /* Complete signal transaction */
}


main()
{
  PNK_DEVICE_TYPE     dev;
  PNK_DEVHDR_PTR      devHdrPtr;
  PNKCAPI_TYPE          pnkc;
  PNK_SIGNAL_TYPE     sig;
  PNK_RETURN_TYPE     ret;

  char      string[256];

  if (pnkc_init() != OK)      /* Initialize client API */
```

```
  {
    printf("Error initializing client interface!\n");
    exit(1);
  }

if (pnkd_init(1024) != OK)  /* Initialize device API, maximum signal size
                               is 1024 long words */
  {
    printf("Error initializing device interface!\n");
    exit(1);
  }

dev.arch = "apisamp";
dev.parent = "";
dev.name = "apisamp";
dev.params.params_len = 0;
dev.params.params_val = 0;

/* Create virtual device (A communications end point) */

if (pnkd_createDev(&dev) != OK)
  {
    printf("Error creating device\n");
    exit(1);
  }

/* Connect signal handler, hander to signal numbe APISAMP_SIG */

if (pnkd_connect(APISAMP_SIG, (FUNCPTR)handler, NULL) != OK)
  {
    printf("unable to connect signal handler\n");
    exit(1);
  }

/* Open a connection to real device NODENAME:BOARDNAME */

if (pnkc_open(&pnkc, NODENAME, BOARDNAME) != OK)
  {
    printf("error opening connection to target\n");
  }

printf("Host running. type 'RETURN' on a blank line to quit\n");

while(1)
  {
    fgets(string,255,stdin);

    if (string[0] == '\n')
    exit(0);


    /* Send read string to target */

    sig.num = APISAMP_SIG;

    sig.data.data_val = (u_int_32 *)string;
    sig.data.data_len = strlen(string)/4 + 1;

    sig.args.args_val = NULL;
    sig.args.args_len = 0;

    sig.timeout_ticks = TIMEOUT_TICKS;
    sig.tick_ulen = TICK_ULEN;
```

```
      if ((strlen(string)%4) !=0)
      sig.data.data_len + 1;

      pnkc_send(&pnkc,0,&sig,&ret);

      /* Wait for target to return string */

      pnkd_pause();

      /* Display received string */

      fputs(_data,stdout);
    }
}
```

```
/*****************************************************************************
 * FILE NAME: APITARGETX.C
 *
 * VERSION DATA:  $Archive: $
 *                $Revision: $
 *                $Modtime: $
 * DESCRIPTION:
 *
 * Target end of SwiftNet API example, using extended size signals
 *
 *****************************************************************************/

/* $History: $
*/

#include "snio.h"
#include "pnkdev.h"
#include "pnkdapi.h"
#include "pnkcapi.h"

#define     APISAMP_SIG 1           /* Signal to send data */
#define SHMALLOC_SIG    2           /* Signal to allocate shared memory */
#define SHMFREE_SIG     3           /* Signal to free shared memory */
#define HANDSHAKE_SIG   4           /* Handshake signal */
#define TERMINATE_SIG   5           /* Signal to terminate transfers */

#define HOSTNAME  "flanders"        /* Name of machine where host end is
                                       running */
#define DEVICENAME      "apisamp"   /* Virtual device name registered by
                                       host */

#define TICK_ULEN       1000            /* 1 mS ticks                      */
#define TIMEOUT_TICKS   10000           /* Number of ticks, TICK_ULEN long */


u_int_32 *_data;
u_int_32 _data_len;
u_int_32 _data_sig = FALSE;
u_int_32 _terminate = FALSE;

/*****************************************************************************
 FUNCTION NAME:    c_int02 - shmalloc interrupt handler
 ENTRY CONDITIONS: A SwiftNet signal # SHMALLOC_SIG was received
                   args_val[0] contains the size in long words to allocate
                   args_val[1] contains the section number
 EXIT CONDITIONS:  Shared memory is allocated using the shmalloc() function
                   call, and the pointer to the memory is returned in ret.val
 *****************************************************************************/

c_int02()

{
    static PNK_RETURN_TYPE ret = { 0L, 0L, NULL };
    PNK_SIGNAL_PTR sig;

    sig = pnkd_signal();

    ret.val = shmalloc(sig->args.args_val[0],sig->args.args_val[1]);

    pnkd_return(&ret);
    _data_sig = FALSE;
}
```

```
/*************************************************************************
 FUNCTION NAME:    c_int03 - shmfree interrupt handler
 ENTRY CONDITIONS: A SwiftNet signal # SHMFREE_SIG was received
                   args_val[0] contains a pointer to the shared memory,
                   previously returned by shmalloc.
 EXIT CONDITIONS:  Shared memory is freed up using the shmfree() function
                   call, and the return value is returned in ret.val
 *************************************************************************/

c_int03()

{
    static PNK_RETURN_TYPE ret = { 0L, 0L, NULL };
    PNK_SIGNAL_PTR sig;

    sig = pnkd_signal();

    ret.val = shmfree(sig->args.args_val[0]);

    pnkd_return(&ret);
    _data_sig = FALSE;
}

/*************************************************************************
 FUNCTION NAME:    c_int04 - terminate signal interrupt handler
 ENTRY CONDITIONS: A SwiftNet signal # TERMINATE_SIG was received
 EXIT CONDITIONS:  _data_sig is set to FALSE
                   _terminate is set to TRUE
 *************************************************************************/

c_int04()
{
    static PNK_RETURN_TYPE ret = { 0L, 0L, NULL };

    pnkd_return(&ret);

    _terminate = TRUE;
    _data_sig = FALSE;
}

/*************************************************************************
 FUNCTION NAME:    c_int01 - Data interrupt handler
 ENTRY CONDITIONS: A SwiftNet signal # APISAMP_SIG was received
 EXIT CONDITIONS:  Received data is copied to _data
                   The data length is long words is placed in _data_len
                   _data_sig is set to TRUE to indicate that a data signal
                   was received.
 *************************************************************************/


c_int01()
{
  PNK_SIGNAL_PTR sig;
  PNK_RETURN_TYPE ret = {0,0,NULL};
  u_int_32   i;

  sig = pnkd_signal();  /* Retreive signal */

  /* Copy data to global variable.  sig is no longer valid after a
     pnkd_return() */

  memcpy(_data,sig->data.data_val,sig->data.data_len);
  _data_len = sig->data.data_len;
```

```
  pnkd_return(&ret);    /* Complete signal transaction, send empty return
                           data */
  _data_sig = TRUE;
}


main()
{
  PNK_DEVICE_TYPE     dev;
  PNKCAPI_TYPE        pnkc;
  PNK_SIGNAL_TYPE     sig;
  PNK_RETURN_TYPE     ret;


  char      string[256];

  shminit(0);     /* Initialize shared memory */

  _data = (u_int_32 *)shmalloc(4096,0);   /* Allocate space in shared memory
                                             for signal data */

  if (pnkc_init() != OK)      /* Initialize client API */
    {
      plog("Error initializing client interface!");
      exit(1);
    }

  if (pnkd_init() != OK)      /* Initialize device API */
    {
      plog("Error initializing device interface!");
      exit(1);
    }

  /* Connect signal handler, c_int01 to Signal # APISAMP_SIG */

  if (pnkd_connect(APISAMP_SIG, (FUNCPTR)c_int01, NULL) != OK)
    {
      plog("unable to connect APISAMP_SIG signal handler");
      exit(1);
    }

  if (pnkd_connect(SHMALLOC_SIG, (FUNCPTR)c_int02, NULL) != OK)
    {
      plog("unable to connect SHMALLOC_SIG signal handler");
      exit(1);
    }

  if (pnkd_connect(SHMFREE_SIG, (FUNCPTR)c_int03, NULL) != OK)
    {
      plog("unable to connect SHMFREE_SIG signal handler");
      exit(1);
    }

  if (pnkd_connect(TERMINATE_SIG, (FUNCPTR)c_int04, NULL) != OK)
    {
      plog("unable to connect TERMINATE_SIG signal handler");
      exit(1);
    }
```

```
 /* Open up a connection to virtual device existing on host */

  if (pnkc_open(&pnkc, HOSTNAME, DEVICENAME) != OK)
    {
      plog("error opening connection to host");
      exit(1);
    }

  plog("target running");

  sig.num = HANDSHAKE_SIG;
  sig.data.data_val = NULL;
  sig.data.data_len = 0;
  sig.args.args_len = 0;
  sig.args.args_val = NULL;

  pnkc_send(&pnkc,0,&sig,&ret); /* Send handshake signal to indicate that
                                   target is up and running */

  while(1)
    {
      while (1)
      {
        pnkd_pause();   /* Wait for SwiftNet signal */
        if (_data_sig | _terminate) break; /* If we received a data signal
                                   or a terminate signal, some
                                   additional processing needs
                                   to be done */
      }

      if (_terminate) break;  /* Terminate signal was received break out
                                 of loop */

      sig.num = APISAMP_SIG;

      sig.data.data_val = (u_int_32 *)_data;

      sig.data.data_len = _data_len;

      sig.timeout_ticks = TIMEOUT_TICKS;
      sig.tick_ulen = TICK_ULEN;

      /* Send received data back to host */

      pnkc_sendEx(&pnkc,0,&sig,&ret);
    }

  shmfree(_data); /* Free up previously allocated shared memory */
}
```

```
/*****************************************************************************
 * FILE NAME: apihostx.c
 *
 * VERSION DATA:  $Archive: $
 *                $Revision: $
 *                $Modtime: $
 * DESCRIPTION:
 *
 * Host end of SwiftNet API example
 *
 *****************************************************************************/

/* $History: $
*/

#include <stdio.h>

#include "pnkdev.h"
#include "pnkdapi.h"
#include "pnkcapi.h"

/* See pnkdev.h for a range of user definable signals */

#define     APISAMP_SIG 1     /* Data Signal */
#define SHMALLOC_SIG    2     /* Shared memory allocation signal */
#define SHMFREE_SIG     3     /* Shared memory free signal */
#define HANDSHAKE_SIG   4     /* Handshake signal */
#define TERMINATE_SIG   5     /* Terminate signal */

#define NODENAME  "sr-200"    /* Name of node where BOARDNAME
                                 resides */

#define BOARDNAME "b1"        /* Name of board running target end
                                 code */

#define PROCNUM         0     /* Processor number */


#define TICK_ULEN       1000           /* 1 mS ticks                    */
#define TIMEOUT_TICKS   10000          /* Total wait time =
                                          TICK_ULEN * TIMEOUT_TICKS uS */

#define     SENDFILE    "host.c"    /* File to send */
#define XFERSIZE  4096              /* Maximum transfer size */


u_int_32 _data[XFERSIZE];
u_int_32 _data_len;
u_int_32 _ok = FALSE;


/*****************************************************************************
 FUNCTION NAME:    handler() - Data handler
 ENTRY CONDITIONS: A SwiftNet signal # APISAMP_SIG was received
 EXIT CONDITIONS:  The received data is placed in _data
                   The received data length is placed in _data_len
 *****************************************************************************/

handler()
{
  PNK_SIGNAL_PTR sig;
  PNK_RETURN_TYPE ret = {0,0,NULL};
  u_int_32   i;
```

```
  sig = pnkd_signal();  /* Retreive signal */

  /* Copy signal to global variable. sig not valid after pnkd_return */

  memcpy(_data,sig->data.data_val,sig->data.data_len * 4);

  _data_len = sig->data.data_len;

  pnkd_return(&ret);    /* Complete signal transaction */
}

/*************************************************************************
 FUNCTION NAME:   handshake() - Handshake signal handler
 ENTRY CONDITIONS: A SwiftNet signal # HANDSHAKE_SIG was received
 EXIT CONDITIONS:  _ok is set to TRUE
 *************************************************************************/

handshake()
{
  PNK_RETURN_TYPE ret = {0,0,NULL};
  pnkd_return(&ret);
  _ok = TRUE;
}


main()
{
  PNK_DEVICE_TYPE     dev;
  PNK_DEVHDR_PTR      devHdrPtr;
  PNKCAPI_TYPE        pnkc;
  PNK_SIGNAL_TYPE     sig,tmpsig;
  PNK_RETURN_TYPE     ret,tmpret;
  FILE                *fp;
  u_int_32        size;
  u_int_32        args[2];
  u_int_32        data[XFERSIZE];
  u_int_32        dpr_data;

  if (pnkc_init() != OK)      /* Initialize client API */
    {
      printf("Error initializing client interface!\n");
      exit(1);
    }

  if (pnkd_init(XFERSIZE) != OK)   /* Initialize device API, maximum signal size
                                      is XFERSIZE long words */
    {
      printf("Error initializing device interface!\n");
      exit(1);
    }

  dev.arch = "apisamp";
  dev.parent = "";
  dev.name = "apisamp";
  dev.params.params_len = 0;
  dev.params.params_val = 0;

  /* Create virtual device (A communications end point) */

  if (pnkd_createDev(&dev) != OK)
    {
      printf("Error creating device\n");
      exit(1);
    }
```

```
/* Connect signal handler, hander to signal number APISAMP_SIG */

if (pnkd_connect(APISAMP_SIG, (FUNCPTR)handler, NULL) != OK)
  {
    printf("unable to connect APISAMP_SIG signal handler\n");
    exit(1);
  }

/* Connect signal handler, handshake to signal number HANDSHAKE_SIG */

if (pnkd_connect(HANDSHAKE_SIG, (FUNCPTR)handshake, NULL) != OK)
  {
    printf("unable to connect HANDSHAKE_SIG signal handler\n");
    exit(1);
  }

/* Open a connection to real device NODENAME:BOARDNAME */

if (pnkc_open(&pnkc, NODENAME, BOARDNAME) != OK)
  {
    printf("error opening connection to target\n");
  }

fp = fopen(SENDFILE,"r");


while(!_ok);    /* Wait for target to handshake */

/* allocate space in shared memory */

tmpsig.num = SHMALLOC_SIG;
tmpsig.args.args_len = 2;
tmpsig.args.args_val = args;
tmpsig.data.data_len = 0;
tmpsig.data.data_val = NULL;
tmpsig.timeout_ticks = TIMEOUT_TICKS;
tmpsig.tick_ulen = TICK_ULEN;

args[0] = XFERSIZE;   /* size */
args[1] = 0;          /* section */


if (pnkc_send(&pnkc, PROCNUM, &tmpsig, &tmpret) != OK){
  fprintf(stderr, "error sending shmalloc signal!\n");
}

dpr_data = tmpret.val;

while(1)
    {
    size = fread(data, 4, XFERSIZE, fp);

    /* Send read string to target */

    sig.num = APISAMP_SIG;

    sig.data.data_val = (u_int_32 *)data;
    sig.data.data_len = size;

    sig.args.args_val = NULL;
    sig.args.args_len = 0;

    sig.timeout_ticks = TIMEOUT_TICKS;
```

```
        sig.tick_ulen = TICK_ULEN;

        pnkc_sendEx(&pnkc,PROCNUM,&sig,&ret, NULL, dpr_data, NULL);

        /* Wait for target to return string */

        pnkd_pause();

        /* Display received string */

        fwrite(_data,4,_data_len,stdout);

        if (size < XFERSIZE)    /* Last block sent */
              break;
        }

  fclose(fp);

  tmpsig.num = SHMFREE_SIG;
  tmpsig.args.args_len = 2;
  tmpsig.args.args_val = args;
  tmpsig.data.data_len = 0;
  tmpsig.data.data_val = NULL;
  tmpsig.timeout_ticks = TIMEOUT_TICKS;
  tmpsig.tick_ulen = TICK_ULEN;

  args[0] = dpr_data;

  if (pnkc_send(&pnkc, PROCNUM, &tmpsig, &tmpret) != OK)
    fprintf(stderr, "error sending shfree signal!\n");


  /* Send terminate signal to indicate to target that we are done */


  tmpsig.num = TERMINATE_SIG;
  tmpsig.args.args_len = 0;
  tmpsig.args.args_val = NULL;
  tmpsig.data.data_len = 0;
  tmpsig.data.data_val = NULL;
  tmpsig.timeout_ticks = TIMEOUT_TICKS;
  tmpsig.tick_ulen = TICK_ULEN;


  if (pnkc_send(&pnkc, PROCNUM, &tmpsig, &tmpret) != OK)
    fprintf(stderr, "error sending terminate signal!\n");
}
```

```
/*****************************************************************************
 *
 *
 * FILE NAME: dim.c
 *
 *
 *****************************************************************************/

/*****************************************************************************/

/*#define CACHE_TEST    1*/
/*#define JTAG_TEST     1*/


/*****************************************************************************/

#include <math.h>


/*****************************************************************************/

#define PI  3.141592654

/*****************************************************************************/

int angleSteps = 200;
int fullDuty = 10000;

/*****************************************************************************/

main()

{
    double angle;
    int onDelay, offDelay;
    int cnt = 0;

#if JTAG_TEST
    asm("   ANDN 2000h,ST");
#endif

    while(1){

      for(angle = 0.0;

          angle <= 2 * PI;
          angle = angle + (2 * PI / angleSteps)){

          cnt++;
          onDelay = (fullDuty / 2.0) * (sin(angle) + 1.0);
          offDelay = fullDuty - onDelay;

          ledOn();
          wait(onDelay);
          ledOff();
          wait(offDelay);

      }
    }
}

/*****************************************************************************/

ledOn()
```

```
{

#ifdef ARCH_4283
    asm("   ldi 6,iof");                                /* turn LED on */
#endif

#ifdef ARCH_4284
    *((volatile unsigned long *)0x90000000) |= 0x40;
#endif

#ifdef ARCH_4280
    *((volatile unsigned long *)0x500000) &= ~0x200;
#endif

#ifdef ARCH_4270
    *((volatile unsigned long *)0x100020) &= ~0x5;
    *((volatile unsigned long *)0x100020) |= 0x2;
#endif

#ifdef ARCH_4285
    *((volatile unsigned long *)0x00380000) &= ~0x80;
#endif

#ifdef ARCH_4288
  unsigned long systat;

  systat = ((*(unsigned long *)0x3) >> 8) & 0x7;      /* get processor ID */
  *(unsigned long *)(0x1400008 + systat) = 0x1;       /* LED ON */
#endif

#ifdef ARCH_4290
    *((volatile unsigned int *)0x28001c) |= 0x8;
#endif

}

/*****************************************************************************/

ledOff()

{

#ifdef ARCH_4283
    asm("   ldi 2,iof");                                 /* turn LED off */
#endif

#ifdef ARCH_4284
    *((volatile unsigned long *)0x90000000) &= ~0x40;
#endif

#ifdef ARCH_4280
    *((volatile unsigned long *)0x500000) |= 0x200;
#endif

#ifdef ARCH_4270
    *((volatile unsigned long *)0x100020) &= ~0x1;
    *((volatile unsigned long *)0x100020) |= 0x6;
#endif

#ifdef ARCH_4285
    *((volatile unsigned long *)0x00380000) |= 0x80;
#endif
```

```
#ifdef ARCH_4288
  unsigned long systat;

  systat = ((*(unsigned long *)0x3) >> 8) & 0x7;      /* get processor ID */
  *(unsigned long *)(0x1400008 + systat) = 0x0;       /* LED OFF */
#endif

#ifdef ARCH_4290
    *((volatile unsigned int *)0x28001c) &= (~0x8);
#endif

}

/*****************************************************************************/

wait(count)

int count;

{
    int i;

#if CACHE_TEST
    asm("    OR 800h,ST");                          /* enable cache */
#endif

    for(i = 0; i < count; i++)
      {
          asm("   nop");
      }
}

/*****************************************************************************/
```

```
/*****************************************************************************
 *
 *
 * FILE NAME: dim69.c
 *
 *
 *****************************************************************************/



#include <math.h>

/*****************************************************************************/

#define PROC_ID         ((unsigned long *)0x340000)
#define TIMER0_CTRL     ((unsigned long *)0x100020)
#define LED_ON          0x302
#define LED_OFF         0x306

#define COM0_CTRL       ((volatile unsigned long *)0x100040)
#define COM0_INPUT      ((unsigned long *)0x100041)
#define COM0_OUTPUT     ((unsigned long *)0x100042)

#define COM3_CTRL       ((volatile unsigned long *)0x100070)
#define COM3_INPUT      ((unsigned long *)0x100071)
#define COM3_OUTPUT     ((unsigned long *)0x100072)
#define PI              3.1415927

/*****************************************************************************/

int angleSteps = 200;
int fullDuty = 10000;

/*****************************************************************************/

main()
{
      if(((*PROC_ID & 0x300000) == 0) || ((*PROC_ID & 0x300000) == 0x300000))
      {
            procAD();                                   /* processor A/D routine */
      }
      else
            {
            procBC();                                   /* processor B/C routine */
            }
}

/*****************************************************************************/

procAD()
{
      double angle;
      int onDelay, offDelay;
      while(1)
            {
                  for(angle = 0.0;
                        angle <= 2 * PI;
                        angle = angle + (2 * PI / angleSteps))
                  {
                        onDelay = (fullDuty / 2.0) * (sin(angle) + 1.0);
                        offDelay = fullDuty - onDelay;

                        *TIMER0_CTRL = LED_ON;              /* turn proc A LED on */
```

```
                        while((*COM0_CTRL & 0x1e0) != 0x0); /* wait for COM0 output
*/
                        *COM0_OUTPUT = LED_OFF;             /* turn proc B LED off *
/
                  wait(onDelay);

                  *TIMER0_CTRL = LED_OFF;                   /* turn proc A LED off *
/

                  while((*COM0_CTRL & 0x1e0) != 0x0);       /* wait for COM0 output
*/
                  *COM0_OUTPUT = LED_ON;                    /* turn proc B LED on */

                  wait(offDelay);
            }
      }
}

/*************************************************************************/

wait(count)
int count;
{
      int i;
      for(i = 0; i < count; i++);
}

/*************************************************************************/

procBC()
{
      while(1)
      {
            while((*COM3_CTRL & 0x1e00) == 0x0);    /* wait for COM3 input */
            *TIMER0_CTRL = *COM3_INPUT;             /* turn LED on or off */
      }
}

/*************************************************************************/
```

```
/*****************************************************************************
 *
 *
 * FILE NAME: dim70.c
 *
 *
 *****************************************************************************/

#include <math.h>

/*****************************************************************************/
#define PROC_ID         ((unsigned long *)0x340000)
#define TIMER0_CTRL     ((unsigned long *)0x100020)
#define LED_ON          0x302
#define LED_OFF         0x306
#define COM0_CTRL       ((volatile unsigned long *)0x100040)
#define COM0_INPUT      ((unsigned long *)0x100041)
#define COM0_OUTPUT     ((unsigned long *)0x100042)
#define COM1_CTRL       ((volatile unsigned long *)0x100050)
#define COM1_INPUT      ((unsigned long *)0x100051)
#define COM1_OUTPUT     ((unsigned long *)0x100052)
#define COM3_CTRL       ((volatile unsigned long *)0x100070)
#define COM3_INPUT      ((unsigned long *)0x100071)
#define COM3_OUTPUT     ((unsigned long *)0x100072)
#define COM4_CTRL       ((volatile unsigned long *)0x100080)
#define COM4_INPUT      ((unsigned long *)0x100081)
#define COM4_OUTPUT     ((unsigned long *)0x100082)

#define PI              3.1415927

/*****************************************************************************/

int angleSteps = 200;
int fullDuty = 10000;

/*****************************************************************************/

main()
{
   int id;
   id = (*(PROC_ID) >> 20) & 0x3;
   switch(id)
      {
            case 0:
                  procA();                                  /* processor A */
            case 1:
                  procB();                                  /* processor B */
            case 2:
                  procC();                                  /* processor C */
            case 3:
                  procD();                                  /* processor D */
      }
}

/*****************************************************************************/

procA()
{
  double angle;
  int onDelay, offDelay;
  while(1)
    {
     for(angle = 0.0;
        angle <= 2 * PI;
```

```
        angle = angle + (2 * PI / angleSteps))
          {
            onDelay = (fullDuty / 2.0) * (sin(angle) + 1.0);
            offDelay = fullDuty - onDelay;
            *TIMER0_CTRL = LED_ON;                          /* turn proc A LED on */
            while((*COM0_CTRL & 0x1e0) != 0x0);             /* wait for COM0 output */
            *COM0_OUTPUT = LED_OFF;                         /* turn proc B LED off */
             while((*COM1_CTRL & 0x1e0) != 0x0);            /* wait for COM1 output */
                  *COM1_OUTPUT = LED_ON                     /* turn proc D LED on */
                  while((*COM3_CTRL & 0x1e0) != 0x0);       /* wait for COM3 output */
                  *COM3_OUTPUT =                            /* turn proc C LED off */
                  wait(onDelay);
                  *TIMER0_CTRL = LED_OFF;                   /* turn proc A LED off */
            while((*COM0_CTRL & 0x1e0) != 0x0);             /* wait for COM0 output */
                  *COM0_OUTPUT = LED_ON;                    /* turn proc B LED on */
                  while((*COM1_CTRL & 0x1e0) != 0x0);       /* wait for COM1 output */
                  *COM1_OUTPUT = LED_OFF;                   /* turn proc D LED off */
                  while((*COM3_CTRL & 0x1e0) != 0x0);       /* wait for COM3 output */
                  *COM3_OUTPUT = LED_ON;                    /* turn proc C LED on */
                  wait(offDelay);
                  }
            }
      }

 /*************************************************************************/

wait(count)
int count;
{
      int i;
      for(i = 0; i < count; i++);
}

/*************************************************************************/

procB()
{
while(1)
      {
      while((*COM3_CTRL & 0x1e00) == 0x0)             /* wait for COM3 input */
      *TIMER0_CTRL = *COM3_INPUT;                     /* turn LED on or off */
      }
}

/*************************************************************************/

procC()
{
      while(1)
      {
            while((*COM0_CTRL & 0x1e00) == 0x0);      /* wait for COM0 input */
            *TIMER0_CTRL = *COM0_INPUT;               /* turn LED on or off */
      }
}

/*************************************************************************/
```

```
procD()
{
      while(1)
      {
            while((*COM4_CTRL & 0x1e00) == 0x0);
                                                  /* wait for COM4 input */
         *TIMER0_CTRL = *COM4_INPUT;               /* turn LED on or off */
      }
}

/*************************************************************************/
```

```
/*****************************************************************************
 *
 *
 * FILE NAME: reverb.c
 *
 *
 *****************************************************************************/

#include <stdlib.h>
#include <snstdio.h>


/*****************************************************************************/

#define HOST_NAME       NULL
#define SOUND_DIR       "/usr/demo/SOUND/sounds"
#define SAMP_RATE       8000
#define SAMP_PERIOD     ((float)(1.0 / (float)SAMP_RATE))
#define OUTPUT_THRSHLD  .01


/*****************************************************************************/

#if defined(MACH_C30) || defined(MACH_C40)
extern unsigned uLawCmp(int val);
extern int      uLawExp(unsigned val);
#endif

#ifdef MACH_C6X
unsigned char linear2ulaw(/* int */);
int ulaw2linear(/* unsigned char */);

#define uLawCmp   linear2ulaw
#define uLawExp ulaw2linear
#endif

#ifdef MACH_21060
#define uLawCmp   mu_compress
#define uLawExp mu_expand
#endif


/*****************************************************************************/

main()

{
    static char buf[BUFSIZ];
    static float rvBuf[SAMP_RATE];
    FILE *auFile, *auDev;
    float delay, gain, tmp, input, output;
    int rvLen, rvIndex, extraIter, extraSamp;
    unsigned data;


    /*
     * initialize standard I/O
     */

    if(snstdio_init(HOST_NAME) != OK){
      plog("snstdio_init() failed!");
      return(-1);
    }
    atexit((void (*)())snstdio_shutdown);
```

```
/*
 * chdir to sound directory
 */

if(chdir(SOUND_DIR) != 0){
  printf("Sound directory does not exist: %s\n", SOUND_DIR);
}

/*
 * list sound files
 */

printf("Audio files:\n\n");
system("ls *.au");

printf("\n\nYou will be prompted for names of audio files to play.\n");
printf("Press <RETURN> with no file name to exit.\n");


/*
 * continue to read files until the user exits
 */

while(1){

  /*
   * get/open audio file
   */

  printf("\n\nEnter an audio file name to load (*.au): ");
  fflush(stdout);

  gets(buf);
  if(strlen(buf) == 0){
      break;
  }

  if((auFile = fopen(buf, "r")) == NULL){
      printf("Cannot open file: %s\n", buf);
      continue;
  }

  /*
   * get delay length, calculate buffer length
   */

  printf("Enter delay length in seconds (%fs <= t <= 1.0s): ",
         SAMP_PERIOD);
  fflush(stdout); gets(buf);
  if(sscanf(buf, "%f", &delay) != 1){
      printf("Invalid delay: %s\n\n", buf);
      fclose(auFile); continue;
  }
  rvLen = delay / SAMP_PERIOD;
  if((rvLen < 1) || (rvLen > SAMP_RATE)){
      printf("Invalid delay: %s\n\n", buf);
      fclose(auFile); continue;
  }

  printf("Buffer length = %d\n", rvLen);


  /*
```

```
 * initialize buffer
 */

for(rvIndex = 0; rvIndex < rvLen; rvIndex++){
    rvBuf[rvIndex] = 0.0;
}
rvIndex = 0;


/*
 * get feedback gain
 */

printf("Enter a feedback gain (0.0 <= x <= 1.0): ");
fflush(stdout);   gets(buf);
if((sscanf(buf, "%f", &gain) != 1) || (gain < 0.0) || (gain > 1.0)){
    printf("Invalid feedback gain: %s\n\n", buf);
    fclose(auFile); continue;
}

/*
 * calculate how long to continue to play after input ends
 */

tmp = gain;
extraIter = 1;
while(tmp > OUTPUT_THRSHLD){
    tmp = tmp * gain;
    extraIter++;
}

printf("Output will be generated for %f seconds after input ends.",
       (float)((float)extraIter * delay));

/*
 * open audio device
 */

printf("\n\nOpening /dev/audio...");
fflush(stdout);

if((auDev = fopen("/dev/audio", "w")) == NULL){
    printf("ERROR\nCannot open audio device for writing: /dev/audio\n");
    fclose(auFile); continue;
}

printf("done.\n");

/*
 * do it!
 */

printf("Sending audio file...");
fflush(stdout);

data = getw(auFile);
while(!feof(auFile)){
    input = uLawExp(data);
    output = input + rvBuf[rvIndex];
    rvBuf[rvIndex] = output * gain;
    rvIndex  = (rvIndex + 1) % rvLen;
    putw(uLawCmp((int)output), auDev);
    data = getw(auFile);
}
```

```
      /*
       * generate extra output to allow signal to decay
       */

      for(extraSamp = extraIter * rvLen; extraSamp > 0; extraSamp--){
          output = rvBuf[rvIndex];
          rvBuf[rvIndex] = output * gain;
          rvIndex  = (rvIndex + 1) % rvLen;
          putw(uLawCmp((int)output), auDev);
      }

      printf("done.\n");

      printf("Closing /dev/audio...");
      fflush(stdout);
      fclose(auDev);
      printf("done.\n");

      printf("Closing audio file...");
      fflush(stdout);
      fclose(auFile);
      printf("done.\n");
    }

    printf("bye!\n");
}


/*****************************************************************************/
```

```
/*****************************************************************************
 *
 * File Name:     ipfifo.c
 *
 * DESCRIPTION:
 *
 * Simple example of using the interprocessor FIFOs and interrupts
 * Processor 0 will write a count between 0 and 3 to the IP Fifo.  After
 * the almost full threshold has been reached, an interrupt will be received
 * by processor 1, which will read the data from the IP Fifo, and flash the
 * corresponding LED.
 *
 *****************************************************************************/

#define RESET     0
#define NMI 1
#define INT4      4
#define INT5      5
#define INT6      6
#define INT7      7
#define INT8      8
#define INT9      9
#define INT10     10
#define INT11     11
#define INT12     12
#define INT13     13
#define INT14     14
#define INT15     15

/* Board registers */

#define IER0  ((volatile unsigned int *)0x280000)  /*Interrupt Enable Reg 0 */
#define IER1  ((volatile unsigned int *)0x280004)  /*Interrupt Enable Reg 1 */
#define IFR0  ((volatile unsigned int *)0x280008)  /*Interrupt Flag Reg 0 */
#define IFR1  ((volatile unsigned int *)0x28000C)  /*Interrupt Flag Reg 1 */
#define ISR0  ((volatile unsigned int *)0x280010)  /*Interrupt Status Reg 0 */
#define ISR1  ((volatile unsigned int *)0x280014)   /*Interrupt Status Reg 1 */
#define BCR   ((volatile unsigned int *)0x280020)   /*Board Config. Reg */

#define MCR0  ((volatile unsigned int *)0x28001c)   /*Misc. Ctrl Reg 0 */
#define MIVR  ((volatile unsigned int *)0x280018)   /*Misc. Int/VME Ctrl */

#define FIFO      ((volatile unsigned int *)0x1000000)      /* Bi-FIFO  */
#define FIFO_MB   ((volatile unsigned int *)0x1200000)      /* Bi-FIFO Mailbox  */

/* Processor registers */

extern cregister volatile unsigned int IER; /* Interrupt Enable Register */
extern cregister volatile unsigned int ICR; /* Interrupt Clear Register */
extern cregister volatile unsigned int CSR; /* Control Status Register */


/* Interrupt vector table, declared in intlib.asm */

extern unsigned int isr_jump_table[];     /* Defined by SwiftNet */

/*****************************************************************************
 FUNCTION NAME:    handler() - Simple interrupt handler
 ENTRY CONDITIONS: A processor INT7 has been received
 EXIT CONDITIONS:  a LED is flashed coresponding to data read from the IP fifo.
                   Handler returns when there is no more data in FIFO.
 RETURNS: nothing
 CAVEATS: none known
 *****************************************************************************/
```

```
interrupt void handler()
{
    *IFR1 = 0x400;            /* Clear almost full flag*/
    ICR = 0x80;               /* Clear C6X INT7 */

    while(!((*ISR1) & 0x100))  /* Read data while the FIFO is not empty */
          flash(*FIFO,1);      /* flash led coresponding to data read */
}


main()
{
    int count;
    volatile unsigned int temp;

    /* Note, the processor codes used here, to detect the processor # only
       apply to four processor boards.  This will have to be changed if using
       a two processor board */


    if ((*BCR & 0xf) == 0xf)  /* Check to see if running on PROC #0 */
      {
        /* IER = 0x2;         /* Disable interrupts */

          *MCR0 &= 0x3fff;    /* Select CPU 0 to 1 Path */

          *MCR0 |= 0x1800;    /* Hold Fifos in RESET */
          delay(1000);
          *MCR0 &= (~0x1800); /* Release Reset */

          *FIFO = 1024 - 0x20;      /* Input Almost Full threshold */
                                    /* Number of samples away from a
                                       full FIFO */

          *FIFO = 0x10;             /* Input Almost Empty threshold */
          *FIFO = 1024 - 0x20;      /* Output Almost Full threshold */
                                    /* Number of samples away from a
                                       full FIFO */

          *FIFO = 0x10;             /* Output Almost Empty threshold */

          delay(1000);

          while(1)
            {
                for (count = 0;count < 4;count++)
                  {
                      *FIFO = count;      /* Write to FIFO */
                      flash(count,1);
                  }
            }
      }

    else if ((*BCR & 0xf) == 0xe) /* Check to see if running on PROC #1 */
      {
          isr_jump_table[INT7] = (unsigned int)handler;

          *IFR1 = 0x400;      /* Clear almost full flag*/
          ICR = 0x80;         /* Clear INT7 */
          IER |= 0x80;        /* Enable INT7 */

          CSR |= 1;           /* GIE = 1 */
```

```
          *IER1 |= 0x400;     /* Enable almost full interrupt */

          *MCR0 &= 0x3fff;    /* Clear select lines      */
          *MCR0 |= 0x4000;    /* Set CPU#1 to CPU#0 path */

          while (1)
            asm(" idle");     /* Wait for interrupt */
      }

    flash(0,255); /* Not on proc 0 or 1, or being run on a two processor
                     4290 */
}

/*************************************************************************
 FUNCTION NAME:   flash() - flash a specified LED a specified number of times
 ENTRY CONDITIONS: led = LED number to flash; count = number of times to flash
 EXIT CONDITIONS:  none
 RETURNS: nothing
 CAVEATS: none known
 *************************************************************************/

flash(int led, int count)
{
    int count2;

    if ((led > 3) || (led < 0))
      led = 0;

    for (count2 = 0; count2<count; count2++)
      {
          *MCR0 = *MCR0 | (1 << led);

          delay(500);

          *MCR0 = *MCR0 & (~(1 << led));

          delay(500);
      }
}
```

```
/*****************************************************************************
 FUNCTION NAME:   delay() - Simple loop delay
 ENTRY CONDITIONS: len = number of iterations of outside loop
 EXIT CONDITIONS:  none
 RETURNS: nothing
 CAVEATS: delay is dependent on processor clock speed
 *****************************************************************************/

delay(int len)
{
    int count,count2;

    for (count2 = 0;count2 < len;count2++)
      {
          for (count = 0;count < 100;count++)
            {
                asm("   nop");
                asm("   nop");
                asm("   nop");
                asm("   nop");
                asm("   nop");
                asm("   nop");
                asm("   nop");
                asm("   nop");
            }
      }
}
```

*This page is intentionally blank*