DSP21k-SF Toolkit 7.4

User's Guide

Interface Libraries and Utilities for BittWare DSP Boards
with SharcFIN DSP-PCI Bridge

BittWare, Inc.
31B S. Main St.
Concord, NH 03301 USA
603.226.0404

If you have comments or suggestions about this manual or find any errors in it, please contact us via e-mail at techpubs@bittware.com.

For technical support, contact us using any of the following methods:
Phone: 603.226.0404
FAX: 603.226.6667
E-mail: support@bittware.com

BittWare also maintains the following Internet sites:

http://www.bittware.com	Contains product information, technical notes, support files available for download, and answers to frequently asked questions (FAQ).
ftp://ftp.bittware.com	Contains technical notes and support files. Log in as "anonymous," and use your e-mail address for the password.

DSP21k-SF Toolkit 7.4 User's Guide

Software Revision 7.4
32-bit version for Windows® 98, NT, 2000, XP, and Linux

Printed in the USA
July 8, 2005 Edition *UG-DTSF74-00-1*

Document History

Document Number	Document Revision	Date	Changes	Section
UG-DTSF74-00-1	0	07/08/2005	Initial release	

This page intentionally left blank.

Contents

This page intentionally left blank.

Tables

Figures

Chapter 1
Introduction

DSP21k-SF Toolkit release 7.4 is a set of libraries and utilities that enable you to develop DSP applications more quickly and easily. This release of the Toolkit is compatible only with BittWare's DSP system boards that feature the Sharc®FIN™ ASIC.

Note *The documentation for the DSP21k-SF Toolkit consists of two documents: the DSP21k-SF Toolkit 7.4 User's Guide and the Host Interface Library Reference Manual.*

After reading this user's guide, you should be able to

- Install the DSP21k-SF Toolkit 7.4 software in your system
- Install, uninstall, and set the properties for your BittWare hardware
- Use the DSP21k-SF Toolkit utilities to interact with your BittWare hardware

1.1 Overview of the DSP21k-SF Toolkit

The DSP21k-SF Toolkit consists of five main components: the Host Interface Library, Diag21k, the DSP Board Automated Diagnostic, and the BittWare Configuration Manager.

1.1.1 Host Interface Library

The primary component of the DSP21k-SF Toolkit is the Host Interface Library (HIL). The HIL is a library of C-callable functions for Windows programs that allow you to download and start programs on the DSP board, read from and write to the DSP board's memory, and control other board functions. The HIL provides a common set of functions for all of the supported boards and automatically performs the required interface operations. Nearly all of the utilities in the DSP21k-SF Toolkit are based on the HIL. For more information about the HIL, refer to the *Host Interface Library Reference Manual.*

1.1.2 Diag21k

Diag21k is a character-based diagnostic utility that you start from the command prompt. Diag21k lets you interactively download DSP programs, start and stop their operation, and access DSP memory. It also provides assembler level debug capabilities, including breakpoints. For more information about Diag21k, refer to Chapter 5 of this user's guide.

1.1.3 DSP Board Automated Diagnostic

The DSP Board Automated Diagnostic (DspBad) is a command line operated utility that you can use to test your DSP board for proper operation. It verifies the ability to communicate with the DSP board from the PC, tests the memory and any special features of the DSP board, and confirms the DSP's ability to load and run a program. For more information about DspBad, refer to Chapter 4 of this user's guide.

1.1.4 BittWare Configuration Manager

The BittWare Configuration Manager is the main configuration tool for BittWare's SharcFIN compatible DSP boards. Use the BittWare Configuration Manager to change device numbers and to determine how DspBad and Diag21k will open boards, either from the information burned into the board's EEPROM or with an EEPROM configuration file. The BittWare Configuration Manager also allows you to get and set board and processor properties. For more information about the BittWare Configuration Manager, refer to Chapter 3 of this user's guide.

1.1.5 BitLoader

BitLoader is a loader utility for BittWare boards that are configured with Xilinx FPGAs and EEPROMs. BitLoader can use Xilinx bit files (*.bit) to load new code into the FPGA; Xilinx compressed SVF files (*.xsvf) to burn new code into the EEPROM; or a DSP bootload-able image file (*.ldr) to load code into the flash. BitLoader can also re-load FPGA code previously stored in the EEPROM. BitLoader allows BittWare and users to send code updates electronically and eliminates any need for cables or extra hardware.

1.1.6 DspGraph

DspGraph is a soft real-time DSP memory graphing utility for Windows. DspGraph accepts a DSP address or host memory buffer address to read memory from. Graphs can be paused and printed, data sets can be written to a file, and peaks can be estimated in real-time with the mouse pointer. Advanced parameters such as sample widths, data widths, floating point, signed and unsigned formats, stride, and DMA transfers give DspGraph the flexibility to apply to a wide range of DSP applications.

1.1.7 BittWorks Server

The BittWorks Server program is a TCP/IP server for any BittWare Remote Client Toolkit application or the BittWare Remote Target. Host Interface Library function calls are sent over TCP/IP to the BittWorks Server which sends the calls to the HIL residing on the server machine which in turn communicates with the BittWare hardware. Programs built against the Client HIL only need to be provided the IP address of the server machine in order to establish communication. Once communication is established, the TCP/IP link is virtually transparent to the user and the program acts almost as if the hardware resides in the client machine. This can be especially useful for embedded systems or any system without a graphical user interface. The BittWare Remote Client Toolkit programs are available for Windows operating systems.

1.2 About This User's Guide

This document describes how to install and use release 7.4 of the DSP21k-SF Toolkit and is part of a set of two manuals for the DSP21k-SF Toolkit: the *DSP21k-SF Toolkit User's Guide* and the *Host Interface Library Reference Manual*. Release 7.4 of the Toolkit is the 32-bit version for Windows® 98, NT, 2000, XP, and Linux and is compatible with BittWare's DSP boards that support the SharcFIN ASIC. Refer to the release note and the file entitled `README.TXT` for updated information that may not have been available when this document was printed.

1.2.1 Conventions in This Document

Below is a list of conventions we have used throughout this user's guide.

- Throughout this manual, the terms BittWare hardware, device, and board are used interchangeably and refer to the BittWare DSP board.
- The terms processor and DSP are used interchangeably.
- File names and directories appear in the "`courier`" font.
- Functions and commands that the user enters (for Diag21k, DspHost or the Host Interface Library) appear in bold "**`courier`**" font.
- Executable programs that are compiled for the Analog Devices ADSP-21xxx processors have a .dxe extension instead of a .exe extension to avoid confusing them with executable files for the PC.

1.2.2 Chapter Overviews

Chapter 2: Installing the DSP21k-SF Toolkit

Chapter 2 gives instructions for installing the DSP21k-SF Toolkit in Windows and Linux.

Chapter 3: Using the BittWare Configuration Manager

Chapter 3 describes how to use the BittWare Configuration Manager for both Windows and Linux systems, including instructions on how to view installed devices, modify a device number, read and set device priorities, and use configuration files.

Chapter 4: Using DspBad

Chapter 4 describes how to use the DSP Board Automated Diagnostic (DspBad) to test your DSP hardware to make sure it is operating properly. The chapter gives an overview of each command.

Chapter 5: Using Diag21k

Chapter 5 describes how to use the interactive diagnostic program Diag21k to interact with the DSP on your BittWare hardware. This chapter gives an overview and, in most instances, an example of each command.

Chapter 6: Using BitLoader

Chapter 6 describes how to use BitLoader to load code into an FPGA, FPGA EEPROM, or flash on a BittWare device. The chapter details the interfaces of both the text and graphical versions of BitLoader.

Chapter 7: Using DspGraph

Chapter 7 describes how to use DspGraph to help view DSP memory. The chapter gives an overview of all the graph settings and options.

Chapter 8: Using the BittWorks Server

Chapter 8 describes how to use the BittWorks Server and its requirements. It gives an overview of the commands available within the BittWorks Server program.

1.3 Related Documentation and Products

1.3.1 Documents

We assume that you are already familiar with the SHARC™ or TigerSHARC™ families of digital signal processors. Please refer to the following documents for more information:

- ADSP-21xxx SHARC or ADSP-TSxxx TigerSHARC documentation– Analog Devices
- *Host Interface Library Reference Manual* – BittWare, Inc.
- The user's guide(s) for your BittWare DSP hardware – BittWare, Inc.
- Project Navigator suite for FPGA development – Xilinx

1.3.2 Products

Analog Devices' VisualDSP++®

Analog Devices' VisualDSP++ includes an integrated development environment (IDE) and debugger that delivers efficient project management so programmers can move easily between editing, building, and debugging. The IDE provides access to a C/C++ compiler, assembler, linker, loader, and splitter. The VisualDSP++ debugger enables you to set breakpoints, single step through code, and perform many other debugging operations.

BittWare VisualDSP Target

BittWare's VisualDSP Target allows you to debug your DSP application right on your BittWare board without a hardware emulator. A plug-in to Analog Devices' VisualDSP++ IDE, the Target allows the VisualDSP++ debugger to communicate directly with BittWare's DSP boards.

BittWare Porting Kit

BittWare's DSP21k-SF Porting Kit gives you the freedom to use your BittWare board on the operating system of your choice. It allows you to easily adapt the DSP21k-SF Toolkit to fit your system, so you can develop DSP applications for your BittWare board.

BittWare Remote Client Toolkit

BittWare’s Remote Client Toolkit, allows for remote access of BittWare hardware anywhere in the world providing the host computer is accessible by TCP/IP. BittWare Remote Client Toolkit provides client versions of all of BittWare's standard tools. . The Remote Client Toolkit is the perfect tool for the DSP developer who does not always have access to the hardware and its host computer directly. Those familiar with BittWare's standard tools will find that using Remote Client Toolkit is almost as if the hardware was in the client machine.

BittWare VisualDSP Remote Target

BittWare's VisualDSP Remote Target allows you to remotely debug your DSP application right on your BittWare board without a hardware emulator. If the host computer provides TCP/IP access, you can use the Remote Target under Analog Devices' VisualDSP ++ IDE to debug code remotely. This can be especially useful for when the host computer is running an operating system on which the VisualDSP++ IDE cannot run.

Chapter 2
Installing the DSP21k-SF Toolkit

This chapter provides instructions for installing release 7.4 of the DSP21k-SF Toolkit in either a Windows or Linux environment. The steps below are an overview of the installation procedure.

1. Install the DSP21k-SF Toolkit libraries and utilities (see this user's guide, section 2.1 for Windows-based systems or 2.2 for Linux-based systems).
2. Install the DSP boards (see the specific hardware user's guide, and section 2.3 of this user's guide).
3. Run the BittWare Configuration Manager to ensure that the boards are configured properly (see Chapter 3 of this user's guide).
4. Test the installed hardware (see Chapter 4 and Chapter 5 of this user's guide).

2.1 Installing the DSP21k-SF Toolkit Libraries and Utilities for Windows

This section describes how to install the DSP21k-SF Toolkit 7.4 libraries and utilities for Windows. The installation program creates the following directories under the target directory (`c:\dsp21ksf` by default).

`\bin`	program dlls and executables
`\docs`	documents, help files, release notes
`\drivers`	driver files for Windows 98/NT/2000/XP (also copied into appropriate system directories)
`\etc`	miscellaneous DSP executables and loader files
`\examples`	VisualDSP and host example projects
`\inc`	library include files
`\lib`	library files

1. Before installing any new BittWare hardware, run the setup program. If the setup program does not start automatically after inserting the CDROM into the CDROM drive, click on *Start | Run...* and type d:\Setup.exe (or replace d: with the letter of your CDROM drive). Then, click *OK*.

Note *If you already have an older version of DSP21k Toolkit installed, we advise you to first uninstall it.*

2. When the setup program finishes, it will prompt you to reboot your computer. If this is an upgrade, and you already have BittWare hardware in your machine, choose **Yes** at the prompt to reboot your computer. If you are installing a new piece of hardware, select **No** at the prompt and shut down your computer.
3. Follow the installation instructions that accompany your hardware.
4. When you next boot your computer, your system may prompt you for drivers for a PCI Standard PCI-to-PCI bridge or a DEC 21154 PCI-to-PCI bridge. If so, follow the directions below:

- Check the *Let Windows Search* option and click *Next*
- If the Windows CAB files exist on your machine (usually in `c:\windows\options`), point Windows to search in that directory. Otherwise, point Windows to search the Windows CDROM and click *Next*.
- Click on *Finish* to finish installing the bridge.

5. Your hardware should be automatically installed, and you can now start running the tools.

2.2 Installing the DSP21k-SF Toolkit Libraries and Utilities for Linux

This section describes how to install the DSP21k-SF Toolkit 7.4 libraries and utilities for Linux.

2.2.1 System Requirements for the DSP21k-SF Toolkit for Linux

The DSP21k-SF Toolkit 7.4 will work with the 2.4 and 2.6 series of Linux kernel and was tested using versions of RedHat Linux, RedHat Enterprise Linux, Fedora Core 2 and Core 3, and YellowDog Linux.

The Toolkit is distributed as an RPM file. If your system does not support RPM, contact Bittware for alternative installation instructions.

The kernel drivers included with the DSP21k-SF Toolkit are compiled during installation to ensure kernel-version compatibility. Therefore, you will need a compiler on your target system before attempting to install the Toolkit. Compilation is performed by RPM, and the compiler is not needed after a successful installation.

The installation and initialization scripts assume that you are using a System V compatible initialization process. To see if your system supports this process, check for the following directories on your system:

```
/etc/rc.d/init.d
/etc/rc.d/rc0.d
/etc/rc.d/rc1.d
/etc/rc.d/rc2.d
/etc/rc.d/rc3.d
/etc/rc.d/rc4.d
/etc/rc.d/rc5.d
/etc/rc.d/rc6.d
```

If these directories are present, the scripts should work correctly.

2.2.2 Installing the DSP21k-SF Toolkit for Linux

Once you have confirmed that your system meets the DSP21k-SF Linux requirements, install the Toolkit.

To install the DSP21k-SF Toolkit for Linux,

1. Install the SharcFIN-based BittWare card in your target system and turn on the power.
2. Become the super user (root).
3. Go to the directory containing the dsp21ksf RPM file.
4. Install the RPM: `rpm -Uvh dsp21ksf-7.40-x.rpm`

Note *The name of the RPM file will be different if you have a more recent version of the DSP21k-SF Toolkit. This command will upgrade any earlier versions of the Toolkit that may have been previously installed on your system.*

2.2.3 Defining the Linux Environment Variable

Before you can use the DSP21k-SF Toolkit, you must define the DSP21KSF environment variable.

To define the DSP21KSF environment variable:

If you are using the bash shell, you can add the following line to the `.bashrc` file in your home directory:

```
export DSP21KSF=/usr/local/bittware
```

You may also want to add `/usr/local/bittware/bin` to your path by adding the following line to the `.bashrc` file in your home directory (or by typing it manually): `export PATH=$PATH:/usr/local/bittware/bin`.

2.2.4 Testing the Linux Drivers

To determine whether the drivers loaded correctly, use `lsmod`:

```
# /sbin/lsmod
  Module                     Size   Used by
  bwfin_module             4444    0 (unused)
  windrvr6 60256   0 [bwfin_module]
```

If both `windrvr6` and `bwfin_module` are present, you should be able to run all of the DSP21k-SF Toolkit utilities.

2.3 Installing New BittWare Hardware

To install your new BittWare hardware,

1. Install the DSP21k-SF Toolkit.
2. Shut down your computer.
3. Follow the installation instructions that accompany your hardware.
4. Boot your computer.

2.3.1 Uninstalling a Device

To uninstall a BittWare device,

1. Shut down the device and your computer.
2. Remove the device from your computer.

2.3.2 Reading and Setting the Properties of Your Bittware Hardware

The BittWare Configuration Manager allows you to set the properties of and view the hardware configuration information for your installed BittWare hardware. Refer to Chapter 3 for details on using the BittWare Configuration Manager.

Chapter 3

Using the Configuration Manager

The BittWare Configuration Manager is the main configuration tool for BittWare's SharcFIN compatible DSP boards. Using either the BittWare Configuration Manager for Windows or Linux, you can do the following:

- View installed devices
- Modify device numbers
- Read and set device properties
- Use configuration files

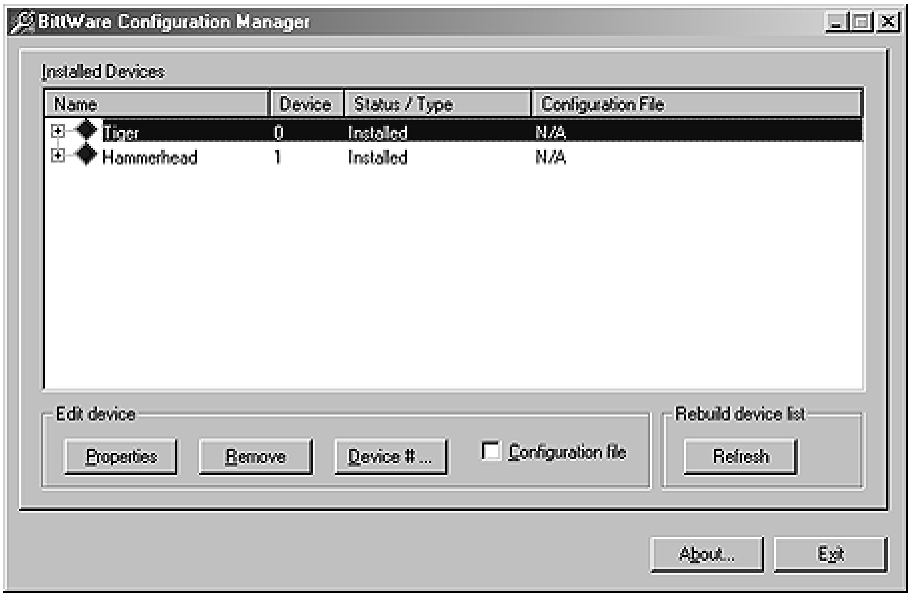
Using the BittWare Configuration Manager for Windows, you can also do the following:

- Open and access device processors
- Start Diag21k sessions

3.1 Using the BittWare Configuration Manager with Windows

The BittWare Configuration Manager allows you to get and set board and processor properties. Two versions of the BittWare Configuration Manager are available: one for Windows-based systems and a second for Linux-based systems. This section describes how to use the Configuration Manager in a Windows-based environment.

Figure 3–1 The BittWare Configuration Manager



Name	Device	Status / Type	Configuration File
Tiger	0	Installed	N/A
Hammerhead	1	Installed	N/A

3.1.1 Viewing Installed Devices in Windows

Each time your system starts, the BittWare Configuration Manager takes a picture of the devices that exist in your system. If the devices shown in the BittWare Configuration Manager's list do not match the devices in your system, you may need to force it to rebuild its list of devices. To force the BittWare Configuration Manager to rebuild its list of devices, run `bwcfg.exe -build` or click on the **Refresh** button.

When you start the BittWare Configuration Manager, it displays a list of all the devices it detects. If you have moved a board from one slot to another or have removed it from your system, the device will appear in the list with a red "X" through it. The BittWare Configuration Manager keeps track of where the device physically existed in your computer until you click **Remove** to remove it from the Configuration Manager list.

3.1.2 Modifying a Device Number in Windows

To modify a device number in the list,

1. Select the board for which you would like to change the device number from the "Installed Devices" list.
2. Select the **Device # ...** button.
3. Choose a new device number from the drop-down list.
4. Click **OK**.

Your device now has a new device number. You can use this number to open your device with the tools or with a Host Interface Library open function.

Figure 3–2 Selecting a Device Number in the BittWare Configuration Manager

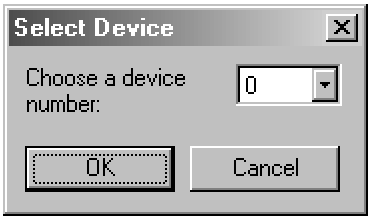
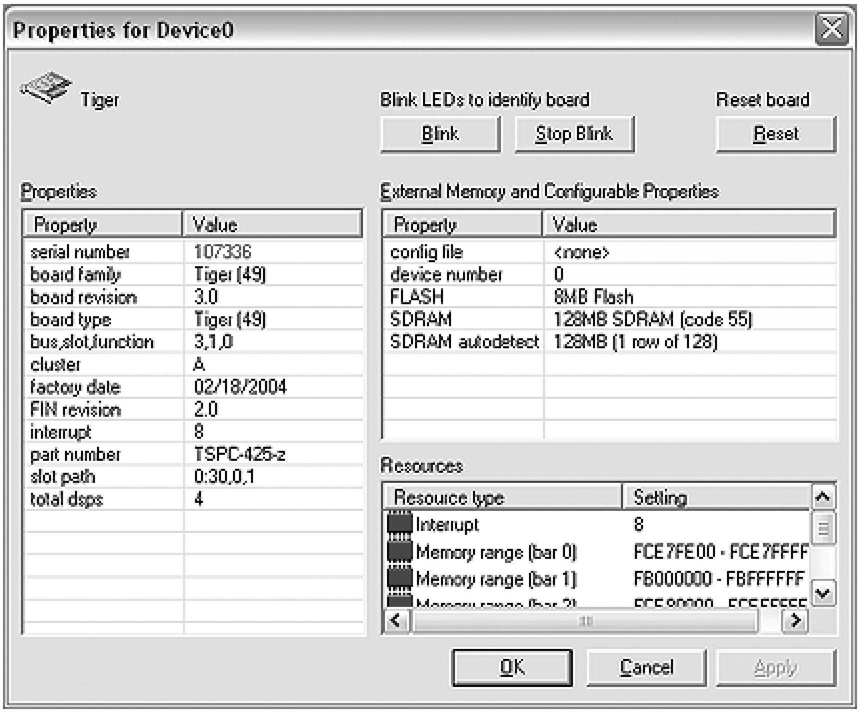


Figure 3–3 The Properties Dialog Box



Properties

Property	Value
serial number	107336
board family	Tiger (49)
board revision	3.0
board type	Tiger (49)
bus,slot,function	3,1,0
cluster	A
factory date	02/18/2004
FIN revision	2.0
interrupt	8
part number	TSPC-425-z
slot path	0:30,0,1
total dsps	4

External Memory and Configurable Properties

Property	Value
config file	<none>
device number	0
FLASH	8MB Flash
SDRAM	128MB SDRAM (code 55)
SDRAM autodetect	128MB (1 row of 128)

Resources

Resource type	Setting
Interrupt	8
Memory range (bar 0)	FCE7FE00 - FCE7FFFF
Memory range (bar 1)	FB000000 - FBFFFFFF

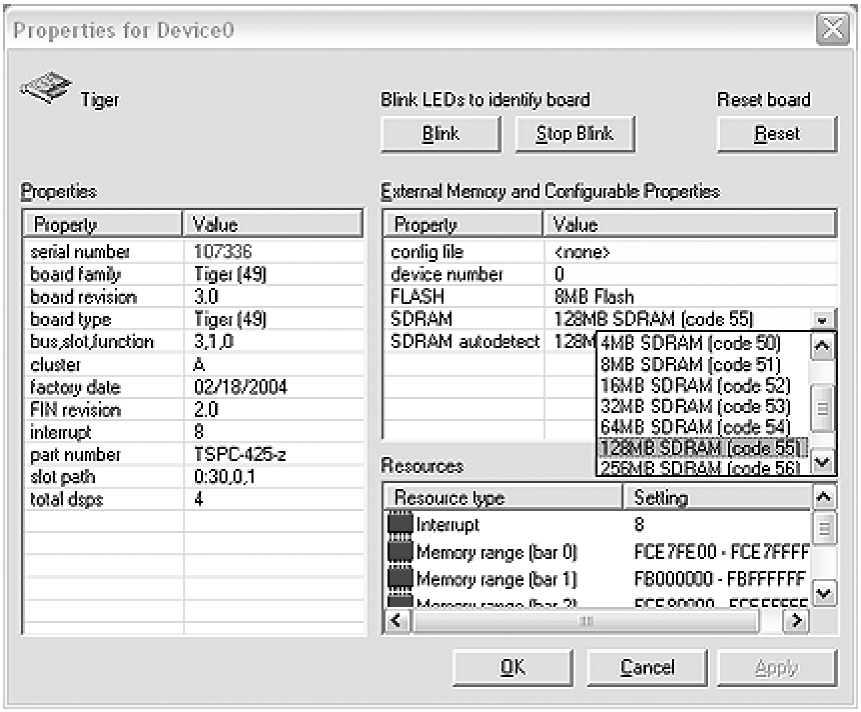
3.1.3 Reading and Setting Device Properties in Windows

To read and set a board's properties,

1. Select the board for which you would like to view or modify the properties from the "Installed Devices" list.
2. Click the **Properties** button.
3. The "Properties for Device *x*" dialog box will appear (see Figure 3–3). If your device is installed properly and is compatible, you will be able to view all of its resources and board information, as well as modify the highlighted **External Memory and Configurable Properties** fields.

Note *In the Configurable Board Information, the MSIZE value should always be greater than or equal to the recommended value. This value is used to set the SYSCON register when you call the HIL function dsp21k_cfg_proc. The SDRAM size should always be less than or equal to the "Autodetect" value. This value is used to reset SharcFIN registers when you call the HIL function dsp21k_reset_bd. The WAIT register value is used to set the WAIT register when you call the HIL function dsp21k_cfg_proc.*

Figure 3–4 The Properties Dialog Box, Modifying a property



Property	Value
serial number	107336
board family	Tiger (49)
board revision	3.0
board type	Tiger (49)
bus,slot,function	3,1,0
cluster	A
factory date	02/18/2004
FIN revision	2.0
interrupt	8
part number	TSPC-425-z
slot path	0:30,0,1
total dsps	4

Property	Value
config file	<none>
device number	0
FLASH	8MB Flash
SDRAM	128MB SDRAM (code 55)
SDRAM autodetect	128M

4MB SDRAM (code 50)
8MB SDRAM (code 51)
16MB SDRAM (code 52)
32MB SDRAM (code 53)
64MB SDRAM (code 54)
128MB SDRAM (code 55)
256MB SDRAM (code 56)

Resource type	Setting
Interrupt	8
Memory range (bar 0)	FCE7FE00 - FCE7FFFF
Memory range (bar 1)	FB000000 - FBFFFFFF

To modify the External Memory and Configurable Properties fields,

1. Click on the field you wish to modify.
2. Click on the button that appears in the value column (see Figure 3–4).
3. If a list of values appears, select the new value. If modifying a Wait Register field, modify the value and click **OK** when you are finished.
4. The new value will appear in the value column. The change will not take effect until either the **OK** or **Apply** button is selected.

Note *Changes to device number and configuration file fields are immediate.*

3.1.4 Using Configuration Files in Windows

Previous versions of the DSP21k Toolkit used environment variables to get information about a device. Now, each SharcFIN device on your BittWare DSP board has an EEPROM on it that is written with this information before being shipped. Every time the HIL functions *dsp21k_open, dsp21k_open_by_id, or dsp21k_open_all* are called, the Toolkit reads information and uses it to open your hardware and get its properties. The BittWare Configuration Manager lets you modify the MSIZE value, SDRAM size, and the WAIT register.

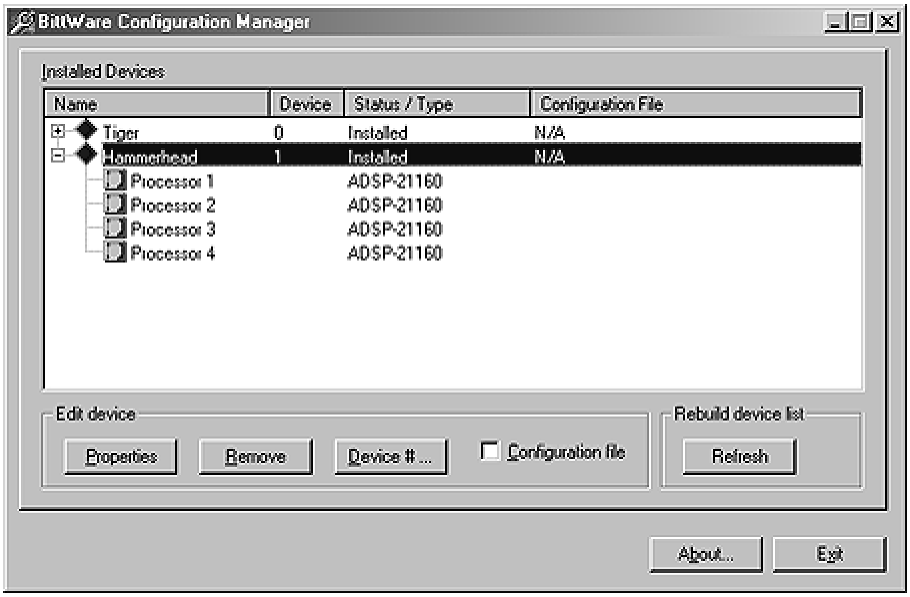
By default, Diag21k, DspBad, and the "Properties" dialog box of the BittWare Configuration Manager get their information from the EEPROM. You can override this default behavior by clicking the **Configuration File** option box for any device. The Configuration Manager will search for a `*.cfg` file instead of using the information in the EEPROM. This EEPROM configuration file contains all the information necessary to communicate with a device.

If you choose this option, the Host Interface Library and all the Toolkit utilities will use the `*.cfg` file you choose for the EEPROM configuration information to open the board. The "Properties" dialog box will also get and set information in the configuration file instead of the EEPROM. A `template.cfg` file is supplied in the `\etc` directory under `c:\dsp21ksf` (or your target directory). An EEPROM configuration file for your device is also maintained in BittWare's archives.

3.1.5 Opening and Accessing a Board's Processor in Windows

Each board with processors that can be opened will have a small box with a plus sign [+] next to it (see Figure 3–5). Click on the plus sign to open all of the processors on that board. The BittWare Configuration Manager will list all of the open processors under the board's name (see Figure 3–5). If the Configuration Manager experiences trouble opening the processors, it will not display them.

Figure 3–5 Opening a Board's Processor



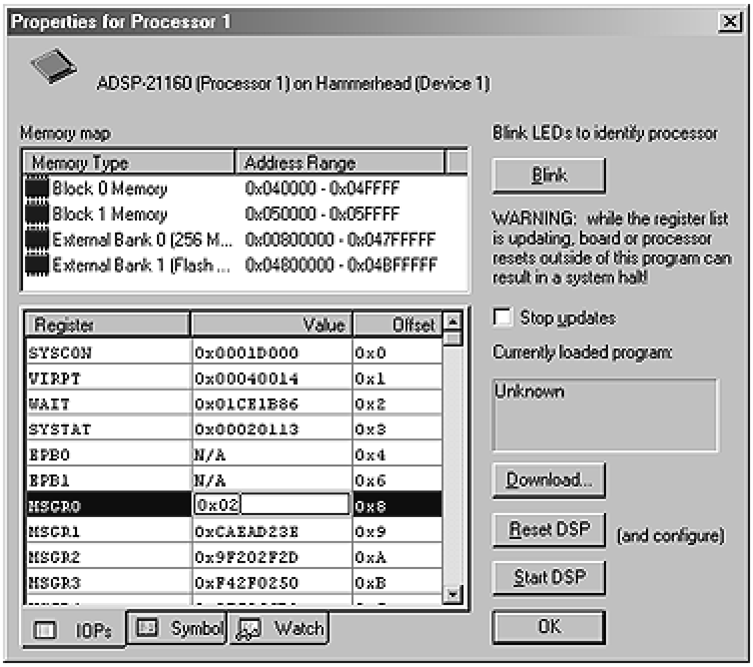
Name	Device	Status / Type	Configuration File
Tiger	0	Installed	N/A
Hammerhead	1	Installed	N/A
Processor 1		ADSP-21160	
Processor 2		ADSP-21160	
Processor 3		ADSP-21160	
Processor 4		ADSP-21160	

To access a processor,

1. Click on one of the processors listed under the device.
2. Click the **Properties** button. The "Properties for Processor x" dialog box will open (See Figure 3–6).

Warning *If you have not yet reset the board upon which this processor resides, the BittWare Configuration Manager will warn you that accessing a processor before resetting the board can cause your computer to crash. Always click Yes unless you have already reset the board since turning the computer on.*

Figure 3–6 The Processor Properties Dialog Box


Properties for Processor 1

ADSP-21160 (Processor 1) on Hammerhead (Device 1)

Memory map

Memory Type	Address Range
Block 0 Memory	0x040000 - 0x04FFFF
Block 1 Memory	0x050000 - 0x05FFFF
External Bank 0 (256 M...	0x00800000 - 0x047FFFFF
External Bank 1 (Flash ...	0x04800000 - 0x04BFFFFF

Register	Value	Offset
SYSCON	0x0001D000	0x0
VIRPT	0x00040014	0x1
WAIT	0x01CE1B86	0x2
SYSTAT	0x00020113	0x3
EPB0	N/A	0x4
EPB1	N/A	0x6
MSGR0	0x02	0x8
MSGR1	0xCAEAD23E	0x9
MSGR2	0x9F202F2D	0xA
MSGR3	0xF42F0250	0xB

Blink LEDs to identify processor

WARNING: while the register list is updating, board or processor resets outside of this program can result in a system halt!

Stop updates

Currently loaded program:

Unknown

(and configure)

The “Properties for Processor x” dialog box shows the following information:

- **Memory map:** The memory map shows the ranges of program memory, data memory and the external memory banks.
- **IOP Tab:** The IOP registers list is selected by default and shows all of the IOP registers for the processor. If you select an IOP register (all except a few such as SYSCON), you can modify that register’s contents by typing in a new value.
- **Symbols Tab:** To view the Symbol list, select the **Symbol** tab. With no program loaded, the Symbol list will remain empty. As soon as you load a program onto the processor, all of the global symbols in the program will be loaded into the Symbol list. If you select a symbol, you can modify that symbol’s contents by typing in a new value. To load the blink program, click on the **Blink** button. To download any other program, click on the **Download...** button.

- **Watch Tab: To view** the Watch list, select the **Watch** tab. You can add symbols or addresses to watch by clicking anywhere in the Watch list. A cursor will appear inside a box in the Address/Symbol column. Type the symbol name or the hexadecimal address you would like to watch. For valid addresses or symbols, you can then modify that symbol's or address's contents by clicking inside the Value box.

3.1.6 Starting Diag21k Sessions in Windows

There are two ways to launch a Diag21k session: with all the processors on a given device or with a single processor on the device.

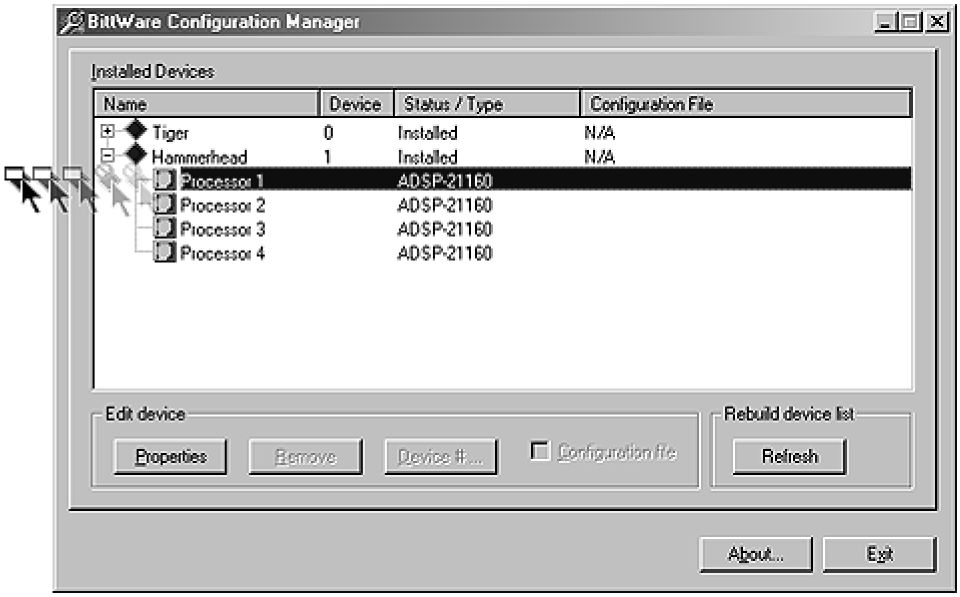
To start a Diag21k session with all the processors on a device,

1. Click on a device name.
2. Drag and drop it out of the list area.

To start a Diag21k session with a single processor,

1. Open the processor list by clicking the plus sign [+] next to the device name (See Figure 3–7).
2. Click on a processor name.
3. Drag and drop it out of the list area.

Figure 3–7 Starting a Diag21k Session for a Single Processor



Name	Device	Status / Type	Configuration File
Tiger	0	Installed	N/A
Hammerhead	1	Installed	N/A
Processor 1		ADSP-21160	
Processor 2		ADSP-21160	
Processor 3		ADSP-21160	
Processor 4		ADSP-21160	

3.2 Using the BittWare Configuration Manager with Linux

The BittWare Configuration Manager for Linux (bwcfgm.exe) is a text-based utility for configuring your BittWare hardware. Using the BittWare Configuration Manager for Linux, you can:

- View installed devices
- Modify device numbers
- Read and set device properties
- Set device configuration files

3.2.1 Viewing Installed Devices in Linux

Each time your computer starts, the BittWare Configuration Manager for Linux takes a picture of the devices that exist in your system. You can force the Configuration Manager to rebuild its list of devices by running `bwcfgm.exe -build`. When you start the BittWare Configuration Manager, it displays a list of all active devices. The BittWare Configuration Manager keeps track of where a device physically exists in your computer until you choose to remove it.

3.2.2 Modifying Device Numbers in Linux

To modify a device number in the list,

1. Enter the board for which you would like to change the device number.
2. Enter the number of the “Change Device Number” menu selection
3. The BittWare Configuration Manager will display a list of device numbers already assigned that you cannot enter. Enter any number between 0 and 255 that is not in the list of assigned device numbers.

Your device now has a new device number. You can use this number to open your device with the tools or with a Host Interface Library open function.

3.2.3 Reading and Setting Device Properties in Linux

To read and set a board's properties,

1. Enter the board you would like to view or modify the properties for.
2. Enter the number of the operation you would like performed.

Note *In the Configurable Board Information, the MSIZE value should always be greater than or equal to the recommended value. This value is used to set the SYSCON register when you call the HIL function dsp21k_cfg_proc. The SDRAM size should always be less than or equal to the "Autodetect" value. This value is used to reset SharcFIN registers when you call the HIL function dsp21k_reset_bd. The WAIT register value is used to set the WAIT register when you call the HIL function dsp21k_cfg_proc.*

3.2.4 Using Configur-ation Files in Linux

Previous versions of the DSP21k Toolkit used environment variables to get information about a device. Now, each SharcFIN device has an EEPROM on it that is written with this information before being shipped. Every time the HIL functions *dsp21k_open, dsp21k_open_by_id, or dsp21k_open_all* are called, the Toolkit reads this information and uses it to open your hardware and get its properties. The BittWare Configuration Manager lets you modify the MSIZE value, SDRAM size, and the WAIT register.

By default, Diag21k, DspBad, and the BittWare Configuration Manager get their information from the EEPROM. To override this default behavior, select the "Use Cfg File" option for any device. The Configuration Manager will allow you to enter a `*.cfg` file instead of using the information in the EEPROM. This EEPROM configuration file contains all the information necessary to communicate with a device.

If you choose this option, the Host Interface Library and all the Toolkit utilities will use the `*.cfg` file you choose for the EEPROM configuration information to open the board. The BittWare Configuration Manager will also get and set information in the configuration file instead of the EEPROM. To override the default behavior in your host programs, call the HIL function *dsp21k_open_by_file* directly. A `template.cfg` file is supplied in the `/etc` directory under `/usr/local/dsp21ksf`. An EEPROM configuration file for your device is also maintained in BittWare's archives.

This page intentionally left blank.

Chapter 4

Using DspBad

This chapter describes how to use the DSP Board Automated Diagnostic utility (DspBad). DspBad is a simple utility that you can use to perform memory tests from both the host and DSP. You can also use it to test DSP program loading and operation. This chapter describes all command line switches for DspBad and their standard usage. If you are having difficulty with your DSP Board, BittWare Technical Support may ask you to run DspBad and provide the results to them. This chapter discusses the following topics:

- All command line switches for DspBad (4.1)
- Examples of standard usage of DspBad (4.2)

4.1 Starting DspBad

DspBad accepts command-line switches that control various start-up options. The general syntax for DspBad is:

```
dspbad [switches]
```

The command line switches are case-sensitive and are preceded with "-". below describes each switch.

Table 4–1 DspBad Command-Line Switches

Switch	Description
-b<N>	Use board number **`<N>`** (overrides -d and -i switches). The board number is the device number * 10 + id number. For example, **`-b22`** is the same as **`-d2 -i2`**
-d<N>	Use device number **`<N>`** (defaults to first found device)
-h	Show usage
-i<N>	Use processor with ID number **`<N>`** (defaults to id number 1)
-n<N>	Run tests **`<N>`** times
-p	Pause before DSP self test
-s	Silent tests
-t<A>	Test types **`<A>`** is one of the following:* p - PC tests n - iNternal tests d - DSP tests x - eXternal tests i - Interrupt tests f - Flash test r - pRimes test
-v	Show version information

* Multiple -t switches are allowed. The default is all except Flash.

4.2 Standard Usage

Generally, after installing and configuring your DSP board, you will reboot the system and then run DspBad to verify that the board was properly installed and set up. The simplest and most complete method to perform this test is to run DspBad with one or more "**`-b<N>`**" parameters. The following are some standard examples of DspBad command lines.

- Run default tests on Processor 1, Device 0 ten times:

  ```
  dspbad -b1 -n10
  ```

- Run interrupt, primes, and internal memory tests on Processor 1, Device 2:

  ```
  dspbad -ti -tr -tn -b21
  ```

Chapter 5

Using Diag21k

This chapter describes how to use the interactive diagnostic utility Diag21k. Diag21k is a diagnostic utility that you can use to download and run DSP programs, perform memory tests, and look at specific memory locations in a variety of formats. This chapter

- Describes all command line switches for Diag21k (5.1)
- Describes Diag21k's command line editor (5.2)
- Describes Diag21k's scripting capabilities (5.3)
- Describes Diag21k's built-in debugger (5.6)
- Describes the stdio mechanism used in Diag21k (5.5)
- Provides a summary of Diag21k commands (5.6)
- Describes and gives an example of each Diag21k command (5.7)

5.1 Starting Diag21k

Diag21k accepts command-line switches that control various start-up options. The general syntax for Diag21k is:

```
diag21k [switches]
```

The command-line switches are case-sensitive and are preceded with "-" or "/". Table 5–1 below describes each switch.

Table 5–1 Diag21k Command-Line Switches

Switch	Description
-b<N>	Instructs Diag21k to open board number **`<N>`** at start-up (this switch overrides the **`-d`** and **`-i`** switches). The board number is the device number * 10 + id number (for example, **`-b22`** is the same as **`-d2 -i2`**). To use Diag21k with more than one board, specify multiple **`-b`** switches. Diag21k supports up to 64 processors or as many as your command shell can handle in the command line.
-d<N>	Use device number **`<N>`** (defaults to first found device). Multiple **`-d`** switches are allowed.
-f<filename>	Diag21k will use cfg_file **`<filename>`** to open the board. This switch overrides the **`-b`** and **`-d`** switches
-h	Show usage
-i<N>	Instructs Diag21k to use processor(s) with id number **`<N>`** only
-p<filename>	Diag21k will load the DSP program **`<filename>`** after the board is opened. If you specify multiple -b switches, Diag21k will download the program to the board described by the first -b switch.
-P<prompt>	Specify prompt as **`<prompt>`**. Diag21k will use the string prompt from the command line instead of the standard "diag21k" prompt. The active board number and ">" will be appended to the end of the prompt.
-v *or* -V	Diag21k displays the version and copyright banner, then exits.
-x<filename>	Execute script **`<filename>`**
-?	Show usage

5.2 Diag21k Command Line

The Diag21k command line editor supports the following keys and special characters:

Table 5–2 Command Line Keys and Special Characters

Home	Go to beginning of current line
End	Go to end of current line
Right arrow	Move cursor right one character
Left arrow	Move cursor left one character
--	Comment - ignores rest of line
;	Separates multiple commands in a single line
Esc	Exit currently running script file or infinite loop
F5	Start debugging, run program
F10	Run to next address
F11	Step program
Shift + F5	Halt program
Backspace	Delete character to the left of the cursor
Delete	Delete character that the cursor is on
Enter	Enter current command line for processing

Note *Some command-line keys such as the function keys may be overridden by the operating system.*

5.3 Diag21k Scripting

5.3.1 Overview

Creating scripts in Diag21k is a way to automate repetitive tasks and to test host to DSP communication schemes before implementing them in a host program. Diag21k scripts are just a plain text file containing Diag21k commands, one per line. For instance, the following simple script file resets the board and configures the processor,

```
-- reset board and configure processor
br
pc
```

5.3.2 Using Constants

Diag21k stores a list of constants that can be used on the command line. When the entered command is parsed, the constants are replaced by their values. A list of all the constants and their values in Diag21k can be viewed with the Show Constants command, 'sc'. Although the names of the constants are constant, the values are dynamic and depend on the current processor, among other things. In the example below, the constant SHARED_MEM_BUF contains the address of the last allocated physical memory buffer:

```
-- allocate host physical memory buffer
lpm 0x10000
-- write address to dsp variable
mw s _host_phys_addr SHARED_MEM_BUF
```

5.3.3 Using Variables

Diag21k has the ability to store user information in the form of variables. Like constants, as an entered command is parsed, variables are replaced by their stored values. Diag21k maintains an internal list of user variables that can be viewed using the Set command, 'set'. The variables are listed above the Diag21k settings and all are prefixed with a '$' symbol. The following examples show variable creation:

```
-- create or set a variable from an immediate value
set $i 10
set $address 0x50020
set $str bli21160.dxe
```

The operations in the following table can be used when either creating a variable or setting an existing variable to a new value.

Operator	Operation	Operands
+	addition	two numeric values
-	subtraction	two numeric values
*	multiplication	two numeric values*
/	division	two numeric values*
%	modulus	two integer values
&	logical AND	two integer values
|	logical OR	two integer values
^	logical XOR	two integer values
.	string concatenation	two strings

* Integer operations greater than 32-bits are not supported.

```
-- create or set a variable from the result of a command,
   operation, or combination of both
set $count (mr s count)
set $n 540 * 31
set $j $i + (mr s index)
```

Addition (+), subtraction (-), multiplication (*), division (/), modulus (%), logical AND (&), logical OR (|), logical XOR(^), and string concatenation (.), are all supported variable operations. Operations must have operators of the correct format or Diag21k will display an error. An operator can be either an immediate value, another variable, or the result of a command in parenthesis as in the examples on the following page. A command in parenthesis should be a command that returns a value such as a single memory or IOP read command. In the above example, parentheses are used to get the result of the command in parentheses. The first command above reads the contents of the DSP variable _count in memory and sets the $count variable to that value.

Variables can be used as parameters in almost all Diag21k commands. For example,

```
-- read $count locations starting at address $addr
mr s $addr $count

-- read mailbox register
rr $mailbox0 4
```

The most advantageous place to use variables is in if and while statements.

5.3.4 Using If and While Statements

If and While statements provide looping and control in Diag21k script files. If and While statements test an immediate value or the result of a command to choose which command(s) to run or if to continue the loop. Use a semicolon within the curly braces to separate multiple commands. The following examples show how to use If and While statements:

```
-- check DSP's error count, if $count variable is 0
if $count {echo Test did not complete!} {echo Test is
   complete; mr s _error_count}

-- run test if $i is 10, otherwise, upload a new $i
if $i == 10 {fx run_test.cmd} {set $i (mr s _i)}

-- if DSP's error is negative, exit Diag21k
if (mr s _error) < 0 {echo Error!; x}

-- wait until count is at least 100 before continuing
while (mr s _count) < 100 {sp 1}

-- run test until run_test.cmd sets $run_test to 0
while $run_test {fx run_test.cmd; set $times_run
   $times_run + 1}
```

Note *Press and hold the escape key to exit an infinite loop.*

5.3.5 Using Goto Label Statements

Labels can be placed inside script files and when used in combination with the goto command, can alter the flow of control in the script execution. The following is an example of using the Goto statement:

```
rem run run_test.cmd unless $done has been set to 1
if $done == 1 {goto DONE}
fx run_test.cmd
:DONE
echo Done running tests.
```

5.4 Diag21k Debugger

Diag21k contains a single-processor assembly level code debugger for SHARC processors. The Diag21k debugger can step, run, halt, set and remove breakpoints, and read and write internal registers. Since the debugger is built directly into Diag21k, most of Diag21k's capabilities are still available while debugging, including scripting, logging, stdio, board register access, memory access, and IOP access.

5.4.1 Starting the Debugger

To start the debugger, use the **`debug`** command. A program can be already loaded, or a program can be loaded after using the **`debug`** command. The debugger will respond by showing the Program Counter and the surrounding memory locations. The debugger can then be stopped at any time by using the **`debug`** command again, by resetting the board (**`br`**), or by exiting Diag21k (**`x`**). Help for debugger commands can be displayed by using the debug help command (**`dh`**).

```
diag21k[1]>fl prm21160.dxe
          "c:\dsp21ksf\etc\prm21160.dxe" loaded
diag21k[1]>debug
         Debugger initialized.
diag21k[1]>
  40002=0000:0000:0000   nop;
  40003=0000:0000:0000   nop;
[___lib_RSTI]
  40004=0000:0000:0000   nop;
->40005=063e:0004:009f<- jump ___lib_start;
  40006=0000:0000:0000   nop;
  40007=0000:0000:0000   nop;
diag21k[1]>
```

5.4.2 Displaying the Current Debug Location

At any time during a debug session, the **`dd`** command is used to display the current program counter and surrounding memory locations. Give the **`dd`** command a count to tell it to display a certain number of surrounding memory locations. The command **`dd 20`** will display 20 lines, with the Program Counter in the middle of the display. The current Program Counter is denoted by arrowheads pointing to the address (**`->`**)and assembly instruction (**`<-`**).

5.4.3 Debugger Operations

Resetting and Restarting

The **`reset`** command is used to perform a processor reset. The Program Counter will be set to the reset vector. To perform a board reset, use the **`br`** command as usual. When a **`br`** command is issued the board will be reset and the debugger session will end. To restart the program use the **`restart`** command. The restart command is the same as the **`reset`** command followed by a **`run`** command.

Single-stepping

Single-stepping through code is accomplished by using the **`step`** command or by pressing **`F11`**. You will see the Program Counter advance to the next location or to a location that is the result of a branch instruction.

Stepping over

The Diag21k debugger can not perform the usual step over function. Instead, when **`F10`** is pressed, Diag21k runs to the next valid instruction. This can be useful for stepping over branch instructions, however this instruction can cause undesired consequences: the debugger will wait until the Program Counter reaches the instruction after the return instruction, but the DSP may never get there.

Running

The **`run`** command releases control of the DSP and sets it free-running until either a **`halt`** command is issued or a breakpoint is reached. While the DSP is running, it is at full speed, the same as it would be had a processor start (**`ps`**) command been issued. If the loaded program contains stdio functions, the stdio will be handled by the Diag21k debugger. If the **`run`** command is given an address as a parameter, a temporary breakpoint will be set at that location and the DSP will run until it reaches that location.

```
diag21k[1]>run main
        Debugger running
diag21k[1]>
  401fd=7c04:d731:0cd2   if le , r14<->s0;
  401fe=887c:3dc2:09b5   if not bm r11=rot r5 by 0x709,
  r11=dm(i4,m1);
  401ff=1279:f9db:7364   stkyy=pm(0xf9db7364);
[_main]
[seg_pmco]
->40200=1607:ffff:fff0<- modify (i7,0xfffffff0);
  40201=ad0f:ffff:ffef   dm(0xffffffef,i6)=r15;
  40202=0f02:0000:0000   r2=0;
diag21k[1]>
```

Halting

The **`halt`** command attempts to interrupt the DSP by using the vectored interrupt (VIRPT). After halting the DSP, use the **`dd`** command to display the current Program Counter and surrounding memory locations. If the VIRPT is disabled, the debugger will not be able to halt and will display a message that it cannot halt. To allow halting, make sure the DSP program does not mask off the VIRPT. To disable halting in certain functions, the VIRPT can be masked upon entry and unmasked upon exit to the function.

Breakpoints

Breakpoints can be set or removed by using the **`break`** command. Use **`break set`** with an address parameter to set a breakpoint. Use **`break remove`** with an address paremeter to remove a specific breakpoint. The **`break remall`** command will remove all breakpoints. Existing breakpoints are denoted with an asterisk (*) when code is displayed using the **`dd`** command or the memory read (**`mr`**) command in disassembly format.

```
diag21k[1]>break set main
        Breakpoint added at 0x040200
diag21k[1]>mr p 0x401ff 5
  [000401ff] = 1279:f9db:7364 stkyy=pm(0xf9db7364);
[_main]
[seg_pmco]
 *[00040200] = 1607:ffff:fff0 modify (i7,0xfffffff0);
  [00040201] = ad0f:ffff:ffef dm(0xffffffef,i6)=r15;
  [00040202] = 0f02:0000:0000 r2=0;
  [00040203] = 1102:0005:0001 dm(_num_reps)=r2;
diag21k[1]>
```

Reading and Writing Registers

Use the **`uh`** command to display all of the possible internal registers that can be accessed in the Diag21k debugger. For the most part, all of the internal registers that can be accessed in VisualDSP++, can also be accessed in Diag21k. Access to internal stacks is not available, however. To read a register, use the **`ur`** command and the name of the register. To write to a register, use the **`uw`** command, the name of the register, and the hex value to write.

5.5 Using Stdio in Diag21k

Diag21k supports the stdio mechanism in DSP programs built against the default I/O library, `libio.dlb` in VisualDSP++. This mechanism is fully documented in the VisualDSP++ C and C++ Run-Time Libraries Guide.

For regular processor start operations, Diag21k checks the **`stdio_enable`** setting (see the Diag21k **`set`** command for more information). If the stdio_enable is non-zero, Diag21k will clear the __lone_SHARC semaphore to tell the DSP that Diag21k is going to handle stdio. Diag21k then polls the the __Godot semaphore until it becomes non-zero. When this happens, Diag21k reads the _PrimIOCB memory location and performs the appropriate I/O operation. When complete, Diag21k writes back to the _PrimIOCB memory location to tell the DSP what has been performed and clears the __Godot semaphore to allow the DSP to continue.

During debugging, Diag21k sets a breakpoint at the __primIO function. When this breakpoint is hit, Diag21k reads the _PrimIOCB memory location and performs the appropriate I/O operation. When complete, Diag21k writes back to the _PrimIOCB memory location to tell the DSP what has been performed and issues a run command internally so the user is unaware of the process.

Command line parameters can be sent to programs with the processor start command. Diag21k will download the parameters to the __argv_string variable. The __argv_string must be an initialized variable. For example, to download a maximum string length of 255,

const char __argv_string[255] = "";

If the __argv_string variable does not exist or is not large enough to accept all of the command line arguments, then the processor start command will display an error. If no arguments are passed to the processor start command, Diag21k will not download anything to the __argv_string variable.

Note *Users of our DspHost program should switch to Diag21k, as DspHost is no longer included in the toolkit. Switching to Diag21k is quick: remove the DspHost libraries from the VisualDSP++ project, remove the paths in the include statements and make sure there is not a path to DspHost header files in the project option's preprocessor includes. Rebuild the DSP application and load and start it in Diag21k. Contact BittWare support if any difficulties are encountered in the process.*

5.6 Command Summary

This section summarizes all of the Diag21k commands (see Table 5–3), and section 5.7 describes them in more detail. Table 5–4 on page 5-61 lists Diag21k commands that have corresponding Host Interface Library functions.

Table 5–3 Summary of Diag21k Commands

Command	Description
–	Comment (ignore remainder of line)
?	Display command list (help)
bch	Broadcast Command Help
bciw	Broadcast IOP Register Write
bcmw	Broadcast Memory Write
bcpc	Broadcast Processor Configure
bcpr	Broadcast Processor Reset
bcps	Broadcast Processor Start
bi	Board Info: display information about active board
br	Board Reset: hard-reset entire DSP board
break	Modify Breakpoints
cd	Change Directory
ds	DSP Select: specify active DSP
echo	Print a message to the screen
dd	Debug Display
debug	Toggle Debugging Mode
dh	Debug Help
dn	DSP New: open a new DSP
dx	DSP close
fl	File Load: download DSP executable (COFF or ELF)

(Sheet 1 of 3)

Table 5–3 Summary of Diag21k Commands (Continued)

Command	Description
fpm	Free Physical Memory buffer
fx	File Execute: run a command file
goto	Script Goto label statement
halt	Halt Debugger
ic	Interrupt Count
if	Script if statement
ih	IOP Help: display IOP register descriptions
ii	Install (/uninstall) interrupts
ir	IOP Read: display IOP register value
iw	IOP Write: set IOP register value
log	Start, stop, continue, or view a log file
lpm	Lock (allocate) Physical Memory buffer
mbd	Memory Buffer Dump
mbl	Memory Buffer Load
mbh	Memory Buffer Help
mbr	Memory Buffer Read
mbv	Memory Buffer View
mbw	Memory Buffer Write
mc	Memory Compare: read and compare address buffers
md	Memory Dump: read DSP memory and write output to file
mh	Memory Help: display memory command help screen
ml	Memory Load: read data from file and write to DSP memory
mm	Memory Map
mr	Memory Read
mt	Memory Test
mw	Memory Write
os	Operating System: open a command shell

(Sheet 2 of 3)

Table 5–3 Summary of Diag21k Commands (Continued)

Command	Description
pc	Processor Configure
pr	Processor Reset
ps	Processor Start
q	Quit Diag21k; reset all active boards
reset	Reset Debugger
restart	Restart Debugger
rr	Read Board Register
run	Run Debugger
rw	Write Board Register
rem	Comment (ignore remainder of line)
sc	Show Constants
set	View or modify board settings and Diag21k variables
sh	Script Help
sl	Symbol Load: load symbol table for a COFF executable file
sp	Script Pause: pause script execution for a time
ss	Show Symbols: show information for loaded symbol(s)
step	Step Debugger
sw	Script Wait: pause script execution until variable equals a value
uh	Universal Register Help
ur	Universal Register Read
uw	Universal Register Write
while	Script While statement
x	Exit: quit Diag21k, leave processor(s) running

(Sheet 3 of 3)

Table 5–4 Diag21k Commands with Corresponding HIL Functions

Diag21k Command*	Base or Major HIL Function
br	dsp21k_reset_bd
dn	dsp21k_open_by_id
bciw	dsp21k_bc_wiop
bcmw	dsp21k_bc_dl_32s
bcpc	dsp21k_bc_cfg_proc
bcpr	dsp21k_bc_reset_proc
bcps	dsp21k_bc_start
fl	dsp21k_load_exe
fpm	dspk21_free_phys_memory
ii	dsp21k_int_enable/dsp21k_int_disable
ir	dsp21k_riop
iw	dsp21k_wiop
lpm	dsp21k_alloc_phys_memory
md	dsp21k_ul_xxxx
ml	dsp21k_dl_xxxx
mr	dsp21k_ul_xxxx
mt	dsp21k_ul/dl_xxxx
mw	dsp21k_dl_xxxx
pc	dsp21k_cfg_proc
pr	dsp21k_reset_proc
ps	dsp21k_start
q	dsp21k_reset_proc, then dsp21k_close

(Sheet 1 of 2)

Table 5–4 Diag21k Commands with Corresponding HIL Functions (Continued)

Diag21k Command[*]	Base or Major HIL Function
rr	dsp21k_rd_bdreg
rw	dsp21k_wr_bdreg
set	dsp21k_rd_bd_setting
sl	dsp21k_load_symbols
ss	dsp21k_get_addr
sw	dsp21k_ul_xxxx
x, dx	dsp21k_close

(Sheet 2 of 2)

* All other Diag21k commands do not correlate closely to any specific HIL function.

5.7 Command Descriptions

Table 5–5 lists the conventions used in this guide to describe the arguments to the commands, and Table 5–6 on page 5-64 lists the debugger notations.

Table 5–5 Diag21k Command Argument Conventions

Notation	Description
[ ]	The argument inside the brackets is optional.
[c]	Continuous Mode: appending “c” to the command will cause it to repeat continuously until you hit a key. For example, memory read (mrc) will poll a memory location until you hit a key. The following are valid continuous commands: irc, iwc, mrc, mwc, mtc.
[p]	Paged Mode: appending “p” to the command will limit the command’s output to a single screen. Pressing the <esc> key will terminate the command; pressing any other key will allow the next screen to be output. Only the “mrp” command is valid.
<bank>*	A single letter or number indicating which memory segment should be affected by the command: b = byte, 1 = 16-bit, 2 = 32-bit, s = 32-bit, 3 = 48-bit, p = 48-bit, 4 = 64-bit; test commands only: i = internal, e = external, a = all.
<fmt>	A single letter indicating the data size and format: c=char, f=float (single-precision), h=hex, i=integer (16-bit), l=long (32-bit), s=string
<test>	A single letter indicating a memory test that should be performed: s=sequential, r=random, c=checkerboard, b=bit set/clear, a=all

* Use bank settings to specify the width of the memory being accessed. The address must correspond to the bank width. For example, if you are reading a short word address, you should use 1, the 16-bit modifier.

Table 5–6 Diag21k Debugger Notations

Notation	Description
*	When debugging, an asterisk in front of an address denotes a breakpoint. If the current PC is at a breakpoint, it will be shown by an arrow overlapped by an asterisk.
->	An arrow indicates the position of the program counter (PC)

Note *This version of Diag21k primarily uses the <bank> specifier to specify the memory width (32 or 48 bits) at the target location. Generally, only the "p," "s," and "l" <bank> specifiers are significant; they are the only ones you should use. When the "p" <bank> specifier is used, memory at the target address is treated as 48-bit wide program memory. When the "s" <bank> specifier is used, memory at the target address is treated as 32-bit wide data memory. And when the "l" <bank> specifier is used, the symbol table entry for the associated text label is accessed, and the width (48- or 32-bit) from the symbol table is used to access memory.*

? - Command Help

Syntax **`? [<cmd>]`**

Description The Command Help command displays a list of Diag21k commands with descriptions. To get help about a specific command and its syntax, specify an optional **`<cmd>`** parameter.

Example **`diag21k[1]>? mr`**

```
     mr[c|p] bank[fmt] addr(hex) [count(dec)]       Read mem-
       ory.

diag21k[1]>?
```

```
+------------------------------------------------------------------------------+
| INFORMATION AND HELP COMMANDS       MEMORY AND IOP ACCESS COMMANDS           |
|  ? <cmd> specific command help       bi         selected board info          |
|  bch     broadcast command help      ir         read IOP register            |
|  ih      IOP register help           iw         write IOP register           |
|  mh      memory command help         md         memory dump                  |
|  sh      script help                 ml         memory load                  |
|  dh      debugger help               mr         memory read                  |
|  mbh     memory buffer cmd help      mt         memory test                  |
|  mm      memory map for processor    mw         memory write                 |
|                                                                              |
| PROCESSOR CONTROL COMMANDS          BOARD CONTROL COMMANDS                   |
|  ds      select active DSP#          br      board reset                     |
|  dn      open a new DSP              ii      install interrupts              |
|  dx      close a DSP                 ic      interrupt count                 |
|  pc      processor configure         rr      read board register             |
|  pr      processor reset             rw      write board register            |
|  ps      processor start                                                     |
|                                     SHELL COMMANDS                           |
| PROGRAM CONTROL COMMANDS             os         OS SHELL command             |
|  fl      load ELF file               cd         change working directory     |
|  sl      load ELF file symbols       q          quit, reset all DSPs         |
|  ss      show symbol(s)              x          exit                         |
+------------------------------------------------------------------------------+
```

bch - Broadcast Command Help

Syntax **`bch`**

Description The Broadcast Command Help command displays information about the broadcast commands.

Example **`diag21k[1]>bch`**

```
+-----------------------------------------------------------------------------+
| BROADCAST HELP                                                              |
|-----------------------------------------------------------------------------|
| These commands perform broadcast writes to all the DSPs in the cluster.    |
|                                                                             |
|------COMMANDS------------------------SYNTAX---------------------------------|
|                                                                             |
|Broadcast Help - show this screen     bch                                    |
|  Memory Write - data memory write    bcmw[c] s[fmt] addr val [count [delta]]|
|     IOP Write - iop register write   bciw[c] <regname | regaddr> value      |
|         Reset - processor reset      bcpr                                   |
|     Configure - processor configure  bcpc                                   |
|         Start - processor start      bcps                                   |
+-----------------------------------------------------------------------------+
```

bciw - Broadcast IOP Register Write

Syntax **`bciw[c] <regname | regaddr> value`**

Description The Broadcast IOP Register Write command works like the IOP Register Write command except that it writes to each processor's registers in the current cluster at once. Not all boards support broadcast commands. The command will display an error if the board does not support it. See the IOP Register Write command for more information on the command parameters and syntax.

Example **`diag21k[1]>bciw msgr0 0x87654321`**

```
diag21k[1]>ir msgr0
  MSGR0    (0x08) = 0x87654321 = 020731241441 = -
  2023406815
diag21k[1]>ds 2
Current DSP:       #2, processor 2 on Hammerhead (device
  0)

diag21k[2]>ir msgr0
  MSGR0    (0x08) = 0x87654321 = 020731241441 = -
  2023406815
```

bcmw - Broadcast Memory Write

Syntax **`bcmw[c] s[fmt] addr value [count [delta]]`**

Description The Broadcast Memory Write command works like the Memory Write command except that it writes to all of the processors in the current cluster at once and can only perform 32-bit writes. Not all boards support broadcast commands. The command will display an error if the board does not support it. See the Memory Write command for more information on the command parameters and syntax.

Example **`diag21k[1]>bcmw s 0x40000 0x12345678`**

```
diag21k[1]>mr s 0x40000
  [00040000] = 0x12345678
diag21k[1]>ds 2
Current DSP:            #2, processor 2 on Hammerhead
  (device 0)

diag21k[1]>mr s 0x40000   [00040000] = 0x12345678
```

bcpc - Broadcast Processor Configure

Syntax **`bcpc`**

Description The Broadcast Processor Configure command works like the Processor Configure command except that it configures all of the processor in the current cluster at once. Not all boards support broadcast commands. The command will display an error if the board does not support it. See the Processor Configure command for more information on the command parameters and syntax.

Example

```
diag21k[1]>bcpc
        Processors configured.
```

bcpr - Broadcast Processor Reset

Syntax `bcpr`

Description The Broadcast Processor Reset command works like the Processor Reset command except that it resets all of the processor in the current cluster at once. Not all boards support broadcast commands. The command will display an error if the board does not support it. . See the Processor Reset command for more information on the command parameters and syntax.

Example

```
diag21k[1]>bcpr
  Processors reset.
```

bcps - Broadcast Processor Start

Syntax **`bcps`**

Description The Broadcast Processor Start command works like the Processor Start command except that it acts on all of the processors in the current cluster. The result is all the processors in the cluster being released from reset at once. Not all boards support broadcast commands. The command will display an error if the board does not support it. See the Processor Start command for more information on the command parameters and syntax.

Example

```
diag21k[1]>bcps
        Processors started.
```

bi - Board Information

Syntax **`bi`**

Description The Board Information command displays information about the active board.[1]

Example **`diag21k[1]>bi`**

```
+-----------------------------------------------------------------------------+
| Board/Processor Information for DSP #1   (Not Started)                      |
|-----------------------------------------------------------------------------|
|   Board Type: (38) Hammerhead                      DSP Type: (7) ADSP-21160 |
|   Multi-proc ID: 1                         Interrupt Number: 10             |
|-----------------------------------------------------------------------------|
|   BAR0: 0x10800000  Size: 0x00000200     BAR3: 0x10800200  Size: 0x00000100 |
|   BAR1: 0x10400000  Size: 0x00400000     BAR4: 0x0e000000  Size: 0x01000000 |
|   BAR2: 0x0c000000  Size: 0x02000000     BAR5:             Size: 0x0        |
|-----------------------------------------------------------------------------|
| Int. Mem: 4 Mbit       IMDW0: 32-bit data                   IMDW1: 32-bit data |
|   MMS WS: 0    Ext Bank Size:  65536 KW (MSIZE = 13) DRAM PgSz:   256 W     |
| Bank 0: Start = 0x00800000  Width = 32 bits  Depth = 65536 KW  WS/WM = 1/2  |
| Bank 1: Start = 0x04800000  Width =  8 bits  Depth =  2048 KW  WS/WM = 7/0  |
| Bank 2: Start = 0x08800000  WS/WM = 1 / 2                                   |
| Bank 3: Start = 0x0c800000  WS/WM = 7 / 0                                   |
| Unbnkd: Start = 0x10800000  WS/WM = 7 / 0                                   |
|-----------------------------------------------------------------------------|
|  Program loaded: (none)                                                     |
|  Labels: *not defined*                                                      |
+-----------------------------------------------------------------------------+
```

Note *The displayed information will vary depending on the board and interface type.*

1. Some information is read from the board, while most comes from internally held information about the board. Wherever possible, information that is available in IOP registers is read from the register for display. Therefore, the sequence “br, bi” may produce different information than “br, pc, bi” does. (The latter is the preferred and more accurate command sequence.)

br - Board Reset

Syntax **`br`**

Description The Board Reset command performs a hardware reset on the board that the active processor is on. It asserts the board-level hardware reset signal on the carrier board, causing all processors on the board to reset. After a Board Reset command, the processors on the board need to be configured before accessing memory. The suggested method of configuring processors depends on the type of processor, according to Table 5–7

Table 5–7 Board Reset command by processor type

Processor type	Diag21k command
ADSP-21xxx	**`pc`** (processor configure), or **`bcpc`** (broadcast processor configure) if supported
ADSP-TS101	**`bcpc`** (broadcast processor configure)
ADSP-TS201	**`bcpc;bcpr`**; (broadcast processor configure; broadcast processor reset)

Example.

```
diag21k[1]>br
      Board reset
```

break - Modify Breakpoints

Syntax **`break <set address | rem address | remall>`**

Description The Modify Breakpoints command allows the insertion and removal of all breakpoints. Symbols can be used in place of the address parameter.

Example

```
diag21k[1]>break set ___modsi3
            Breakpoint added at 0x04036e
diag21k[1]>break rem ___modsi3
            Breakpoint at 0x04036e removed
diag21k[1]>break remall
            All breakpoints removed
diag21k[1]>
```

cd - Change Directory

Syntax **`cd [directory]`**

Description The Change Directory command changes Diag21k's current working directory. If directory is not specified, the current working directory is displayed. Diag21k's current working directory is the directory that is looked in first for opening files for reading or writing.

Example **`diag21k[0]>cd c:\dsp21ksf\exaples\ts101\primests101 \debug`**

```
 c:\dsp21ksf\examples\ts101\primests101\debug
diag21k[0]>cd
 c:\dsp21ksf\examples\ts101\primests101\debug
```

dd - Debug Display

Syntax **`dd [number]`**

Description The Debug Display command displays **`number`** lines of code around and including the current program counter (PC). Without the parameter number, Debug Display shows six lines of code by default.

Example

```
diag21k[1]>dd 3

       4026c=013e:0000:a032 comp(r3,r2);
     ->4026d=0722:0000:0003 if ge jump (pc, 0x3);
       4026e=013e:0002:9330 r3=r3+1;

diag21k[1]>
```

debug - Debug Toggle

Syntax `debug`

Description The Debug command toggles the debugging mode on/off. If the debugger is toggled on, it will attempt to halt at the current program counter (PC) position. Use the reset command to reset the PC to the reset vector.

Example

```
diag21k[1]>debug

     Debugger initialized.

diag21k[1]>

       40002=0000:0000:0000 nop;
       40003=0000:0000:0000 nop;
       40004=0000:0000:0000 nop;
     ->40005=063e:0004:0085 jump 0x40085;
       40006=0000:0000:0000 nop;
       40007=0000:0000:0000 nop;

diag21k[1]>
```

Note *This command requires the current processor to be configured. Use the Processor Configure command after each board or processor reset .*

dh - Debug Help

Syntax **`dh`**

Description The Debug Help command displays the Debugger commands and syntax.

Example **`diag21k[1]>dh`**

```
+-COMMANDS---------------------------------------SYNTAX------------------------+
|                                                                              |
|    Debug Help - show this screen               dh                            |
|         Debug - toggle debugging mode on/off   debug              (F5)       |
|           Run - run program or run to address  run [address]      (F5)       |
|          Step - step program one location      step               (F11)      |
|         Reset - reset debugger                 reset                         |
|       Restart - restart debugger (reset+run)   restart                       |
|          Halt - halt debugger                  halt               (SHIFT-F5) |
| Debug Display - display current PC and number  dd [number]                   |
|                 of surrounding addresses                                     |
|   Breakpoints - Add, Remove, or Remove all     break <set|rem|remall>        |
|     UReg Help - help on universal registers    uh                            |
|     UReg Read - read universal register        ur <register>                 |
|    UReg Write - write universal register       uw <register> <value(hex)>    |
|                                                                              |
|- EXAMPLES: ------------------------------------------------------------------|
|                                                                              |
|       Run: diag21k[1]>run _main                                              |
|   Display: diag21k[1]>dd 8                                                   |
|    Breaks: diag21k[1]>break set _exit                                        |
| Registers: diag21k[1]>uw r0 0x123456789a                                     |
+------------------------------------------------------------------------------+
```

dn - DSP New: Open a new DSP

Syntax **`dn <dspnum>`**

Description The DSP New command tries to open a DSP defined by the given DSP number. The command will then display a list of the open DSPs. The active DSP number will not change.

Example

```
diag21k[1]>dn 11
     Available DSP numbers: 1 2 3 4 11
```

ds - DSP Select

Syntax **`ds [dspnum]`**

Description The DSP Select command makes **`dspnum`** the active board. Diag21k will interpret and apply all subsequent commands in the context of the DSP defined by the given DSP number. The **`dspnum`** parameter must be one of the DSPs that was opened. If you do not specify a **`dspnum`** parameter, a list of opened DSPs is displayed. If you specify a DSP that is not open, the active DSP number will not change.

Note *The active DSP number is always displayed in the command prompt.*

Example

```
diag21k[1]>ds 2
     Current DSP is 2
diag21k[2]>
```

dx - DSP Close

Syntax **`dx <dspnum>`**

Description The DSP Close command will close the DSP defined by the given DSP number if the DSP given is open and is not the active DSP.

Example

```
diag21k[1]>dx 11
      DSP #11 closed.
```

echo - Print message to screen

Syntax **`echo <message string>`**

Description The Echo command displays <message string> to the current output device. This command is most useful in command files to describe execution steps. It can also be used to show the contents of constants and variables.

Example **`diag21k[1]>echo Reset and load Processor B`**

```
Reset and load Processor B

use echo to display constant
diag21k[1]>lpm 0x10000
        Locked shared memory buffer
        Physical address: 0x0000e000, Size: 0x00010000
        Use fpm to free memory
diag21k[1]>echo SHARED_MEM_BUF
0x0000e000

use echo to display variable
diag21k[1]>set $file bli21160.dxe
diag21k[1]>echo $file
bli21160.dxe
```

fl - File Load

Syntax **`fl filename[.dxe]`**

Description The File Load command resets and configures the active processor and then downloads a DSP executable file (ELF) that the Analog Devices linker generates. If you do not specify a file extension, Diag21k will append .dxe. This command will not start the program; you must use the **`ps`** (Processor Start) command to start the processor.

Diag21k will store the addresses of global symbols that are stored in the ELF file; you can use the **`ga`** (Get Address) command or the **`l`** bank modifier in memory access commands to access the symbols. You can view the available global symbols by using the **`ss`** (show symbols) command. Internal C language or debugger symbols are left out of the list by default. These symbols can be shown by using the **`show_c_symbols`** setting.

By default, this command resets the processor before loading it. If the **`reset_on_load`** setting is off, this command will not reset the processor. See the **`set`** command for more information on board settings.

Example

```
diag21k[1]>fl prmts101
      "C:\dsp21ksf\etc\prmts101.dxe" loaded
```

fpm - Free Physical Memory

Syntax **`fpm <physical address>`**

Description The Free Physical Memory command frees the physical memory that was allocated previously with the **`lpm`** command.

Example **`diag21k[1]>fpm 0x03add0000`**

```
      Freed shared memory buffer at address: 0x03add000
```

fx - File eXecute

Syntax

```
fx filename[.CMD]
```

Description

The File Execute command redirects input from a text file that contains a list of commands for Diag21k to execute. The commands in the file use the same syntax as commands that are entered at the keyboard. If you do not specify a file extension, Diag21k will append .CMD.

Example

```
diag21k[1]>fx memtest.cmd
```

goto - Goto

Syntax **`goto <LABEL>`**

Description The goto LABEL Statement finds the specified label in the current script file and execution is transferred to the next line after the label.

Example

```
if $done == 1 {goto DONE} {goto NOTDONE}
:NOTDONE
echo not done yet
goto EXIT
:DONE
echo done!
:EXIT
```

halt - Halt Debugger

Syntax `halt`

Description The Halt Debugger command uses the VIRPT to interrupt the currently executing program code.

Note *The VIRPT interrupt must not be disabled in order for the halt command to work properly.*

Example

```
diag21k[1]>halt
     Debugger halted
diag21k[1]>
```

ic - Interrupt Count

Syntax `ic`

Description The Interrupt Count command displays the number of interrupts that occurred since the interrupts were installed using the Install Interrupts (**`ii`**) command.

Example

```
diag21k[1]>ic
     Last interrupt received: #412
```

if - Script If

Syntax **`if expr {TRUE command} {FALSE command}`**

Description The Script If command tests the expression, **`expr`**. If the result is true, the first command or set of commands in curly braces {} is executed. If the result is false, the second command or set of commands in curly braces {} is executed. See section 5.3 for further information on using if statements and expressions in Diag21k.

Example

```
diag21k[1]>if (mr s primes[19]) == 0 {echo primes has not
    run}{mr s primes 20}
     primes has not run
```

ih - IOP Help

Syntax **`ih [<regname | regaddr>]`**

Description The IOP Help command displays IOP register descriptions and addresses for the current processor. You can specify the IOP register by its name or its address. Typing **`ih`** without an argument lists all IOP registers.

Example

```
diag21k[1]>ih syscon
     "SYSCON"    (0x00)  System configuration register
diag21k[1]>ih 0xe0
     "STCTL0"    (0xe0)  Serial Port 0 Transmit Control Reg-
      ister
```

ii - Install Interrupts

Syntax

```
ii [cmd_file]
```

Description The Install Interrupts command will install an interrupt service routine for the device that the current processor is on if one is not already installed. If the device already has an interrupt service routine, this command will uninstall it. If the parameter **`cmd_file`** exists, the commands in the file will be executed when an interrupt occurs. The **`cmd_file`** parameter should be in the format of a Diag21k script file (.cmd). See the File Execute (**`fx`**) command for more information.

Example

```
diag21k[1]>ii
     interrupts installed
```

ir - IOP Read

Syntax **`ir[c] <regname | regaddr> [count]`**

Description The IOP Read command reads and displays the value of an IOP register. You can specify the IOP register by its name or its address. Wildcard characters (*, ?) can be used in the name to find registers that match a pattern. If you type **`irc`**, Diag21k will read the same IOP register continuously until you press a key. The optional **`count`** parameter will cause Diag21k to read subsequent IOP register addresses.

Example **`diag21k[1]>ir syscon 4`**

```
SYSCON  (0x00) = 0x00006c00 = 000000066000 = 27648
VIRPT   (0x01) = 0x00020014 = 000000400024 = 131092
WAIT    (0x02) = 0x165d1826 = 002627214046 = 375199782
SYSTAT  (0x03) = 0x00000113 = 000000000423 = 275
```

`diag21k[1]>ir 0 4`

```
SYSCON  (0x00) = 0x00006c00 = 000000066000 = 27648
VIRPT   (0x01) = 0x00020014 = 000000400024 = 131092
WAIT    (0x02) = 0x165d1826 = 002627214046 = 375199782
SYSTAT  (0x03) = 0x00000113 = 000000000423 = 275
```

`diag21k[0]>ir systat*`

```
SYSTAT (0x180486) = 0x00003a80 = 000000035200 = 14976
SYSTATC  (0x180487) = 0x00003a80 = 000000035200 = 14976
SYSTATCL (0x180487) = 0x00003a80 = 000000035200 = 14976
SYSTAT_PID = 0x00000000 = 000000000000 = 0
SYSTAT_BM  - 0x00000000 = 000000000000 = 0
SYSTAT_HM  = 0x00000001 = 000000000001 = 1
SYSTAT_MC  = 0x00000002 = 000000000002 = 2
SYSTAT_FR  = 0x00000001 = 000000000001 = 1
SYSTAT_BMOD = 0x00000001 = 000000000001 = 1
SYSTAT_MRSC = 0x00000001 = 000000000001 = 1
SYSTAT_BSLCK = 0x00000000 = 000000000000 = 0
SYSTAT_SBRRD = 0x00000000 = 000000000000 = 0
SYSTAT_AUTODMAERR = 0x00000000 = 000000000000 = 0
SYSTAT_SDRAMERR = 0x00000000 = 000000000000 = 0
SYSTAT_MPRD = 0x00000000 = 000000000000 = 0
SYSTAT_REV = 0x00000000 = 000000000000 = 0
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

iw - IOP Write

Syntax

```
iw[c] <regname | regaddr> value
```

Description The IOP Write command writes **`value`** to an IOP register. You can specify the IOP register by its name or its address. If you type **`iwc`**, Diag21k will write the same IOP register continuously until you press a key.

Example

```
diag21k[1]>iw wait 0x21ad68a5
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

log - Logging Command

Syntax **`log < start | continue | stop | view>`**

Description Using the Logging Command, you can start a log file, continue an existing log file, stop logging to the current log file, or view an existing log file. Use the **`show_logging`** setting to tell Diag21k to place timestamps in the log file. See the **`set`** command for more information.

Example

```
diag21k[1]>log start log1.txt

     LOG STARTED - log1.txt Fri May 04 15:29:09 2001


diag21k[1]>log stop

     LOG STOPPED - log1.txt Fri May 04 15:29:16 2001


diag21k[1]>log continue log1.txt

     LOG STARTED - log1.txt Fri May 04 15:30:01 2001


diag21k[1]>log view

     LOG STARTED - log1.txt Fri May 04 15:29:09 2001

     Board reset
     processor configured
     LOG STOPPED - log1.txt Fri May 04 15:29:16 2001
     LOG STARTED - log1.txt Fri May 04 15:30:01 2001

     "c:\dsp21ksf\etc\prm21160.dxe" loaded
             processor running
             DATA_SRAM [00050045] = 0x0000:000d
             LOG STOPPED - log1.txt Fri May 04 15:30:36 2001
diag21k[1]>
```

lpm - Lock Physical Memory

Syntax **`lpm [phys_addr] size`**

Description If the **`phys_addr`** parameter does not exist, the command will attempt to allocate a new host physical memory buffer of **`size`** bytes. If the **`phys_addr`** parameter exists, the Lock Physical Memory command will attempt to map **`size`** in bytes host physical memory at address **`phys_addr`**. The locked memory can be accessed by using the Memory Buffer Read (**`mbr`**) and Memory Buffer Write (**`mbw`**) commands.

Note *Use caution when giving a host physical address to map. If an incorrect address is used, operating system memory or another PCI device's memory could be accessed. Accessing unknown host physical memory can possibly corrupt the operating system or affect the hardware in some way, resulting in a crash or a hardware failure. Permanent corruption of the operating system or hardware is also possible.*

Example **`diag21k[1]>lpm 0x10000`**

```
Locked shared memory buffer
Physical address: 0x079ec000, Size: 0x00010000
Use fpm to free memory
```

mbd - Memory Buffer Dump

Syntax

```
mbd <physical address> <32-bit offset> file <count>
```

Description

The Memory Buffer Dump command reads the contents of the shared memory buffer allocated with the Lock Physical Memory (lpm) command and writes it to a file. This command will read count 32-bit addresses starting at the 32-bit offset into the buffer. If count is larger than the number of 32-bit values that can be read from the buffer, an error will not occur and only the amount of data that can be read from the buffer will be written to the file.

Example

```
diag21k[1]>mbd SHARED_MEM_BUF 0 100 phys_mem.txt
```

mbl - Memory Buffer Load

Syntax

```
mbl <physical address> <32-bit offset> file [count]
```

Description The Memory Buffer Load command reads the contents of file, and writes each line (except lines starting with a semicolon) to the shared memory buffer allocated with the Lock Physical Memory (lpm) command. If count is specified, this command will read count 32-bit values from the file and write to the buffer starting at the 32-bit offset. If count is not specified, the command will read the entire file and write to the buffer starting at the 32-bit offset. If count is too large for the buffer, an error will not occur and only the amount of data that will fit into the buffer will be written.

Example

```
diag21k[1]>mbl SHARED_MEM_BUF 0 phys_mem.txt 20
```

mbh - Memory Buffer Help

Syntax **`mbh`**

Description The Memory Buffer Help command displays detailed syntax information for the commands dealing with host physical memory buffers.

Example **`diag21k[1]>mbh`**

```
+-------------------------------------------------------------------------------+
| PHYSICAL MEMORY BUFFER COMMAND HELP                                           |
|-------------------------------------------------------------------------------|
|-COMMAND----SYNTAX-------------------------------------------------------------|
|            (addr is a Host Physical (byte) Address and offset is 32-bit)      |
| Lock       lpm   [addr] size                                                 |
| Free       fpm   addr                                                         |
| Dump       mbd   addr offset filename count                                    |
| Load       mbl   addr offset filename [count]                                  |
| Read       mbr   addr offset [count]                                           |
| Write      mbw   addr offset value [count]                                     |
| View       mbv                                                                |
|                                                                               |
|- EXAMPLES: -------------------------------------------------------------------|
| Lock 0x10000 bytes           diag21k[1]>lpm 0x10000                           |
| Read first 4 32-bit words    diag21k[1]>mbr 0x1f7000 0 4                      |
| Write 0's to entire buffer   diag21k[1]>mbw 0x1f7000 0 0 0x10000              |
+-------------------------------------------------------------------------------+
```

mbr - Memory Buffer Read

Syntax **`mbr <offset> [count]`**

Description The Memory Buffer Read command reads and displays the contents of the shared memory buffer allocated with the Lock Physical Memory (**`lpm`**) command. This command will read **`count`** 32-bit addresses starting at the 32-bit **`offset`** into the buffer.

Example **`diag21k[1]>mbr 0 10`**

```
[0x0746A000] =     1,     0x00000001
[0x0746A004] =     2,     0x00000002
[0x0746A008] =     3,     0x00000003
[0x0746A00C] =     4,     0x00000004
[0x0746A010] =     5,     0x00000005
[0x0746A014] =     6,     0x00000006
[0x0746A018] =     7,     0x00000007
[0x0746A01C] =     8,     0x00000008
[0x0746A020] =     9,     0x00000009
[0x0746A024] =    10,     0x0000000A
```

mbv - Memory Buffer View

Syntax **`mbv`**

Description The Memory Buffer View command displays a list of the host physical memory buffers allocated using the **`lpm`** command.

Example **`diag21k[0]>mbv`**

```
     2 host physical memory buffers:
           Physical address: 0x02a85000, Size: 0x00010000
           Physical address: 0x0200e000, Size: 0x00010000
```

mbw - Memory Buffer Write

Syntax `mbw <offset> <value> [count]`

Description The Memory Buffer Write command writes **`value`** into the 32-bit address **`offset`** for **`count`** 32-bit address locations.

Example

```
diag21k[1]>mbw 0 0xdeaddead 10
```

mc - Memory Compare

Syntax **`mc <bank>[<fmt>] addr1 addr2 [count]`**

Description The Memory Compare command reads **`count`** locations from addr1 into a buffer and **`count`** from addr2 into a buffer, and compares the two buffers. If the two buffers are not equal, the first 5 values that are not equal will be displayed. The address displayed for unequal values is the offset into the buffers that the value is found.

Example

```
diag21k[0]>mc s 0 0x200 100
     The memory blocks are equal.
diag21k[0]>mc s 1 0x201 200
       [0x00000063]  0x95b87b05 != 0xb3c00000
       [0x00000064]  0x8e690810 != 0xb1080021
       [0x00000065]  0x00443f11 != 0xb3c00000
       [0x00000066]  0xb2000800 != 0xb108001f
       [0x00000067]  0xa8141080 != 0xb3c00000
           The memory blocks are not equal.
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

md - Memory Dump

Syntax `md <bank>[<fmt>] addr count filename`

Description The Memory Dump command reads **`count`** locations from memory bank **`<bank>`** and address **`addr`** and then writes the data in **`<fmt>`** format to an ASCII file (**`filename`**). Table 5–1 on page 5-46 gives the syntax for **`<bank>`** and **`<fmt>`**.

Example

```
diag21k[1]>md li _primes 20 primes.dmp
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

mh - Memory Help

Syntax **`mh`**

Description The Memory Help command displays detailed syntax for the memory commands.

Example **`diag21k[1]>mh`**

```
+--------------------------------------------------------------------------------+
|-COMMAND----SYNTAX--------------------------------------------------------------|
| Mem-Dump   md      <bank>[<fmt>] addr count filename                           |
| Mem-Load   ml      <bank>[<fmt>] addr filename                                 |
| Mem-Read   mr[c|p] <bank>[<fmt>] addr [count]                                  |
| Mem-Test   mt[c]   <bank><test> [<max_err>]                                     |
| Mem-Write  mw[c]   <bank>[<fmt>] addr value [count [delta]]                     |
| Mem-Cmp    mc      <bank>[<fmt>] addr1 addr2 [count]                           |
|                                                                                |
|-MODIFIERS----------------------------------------------------------------------|
|       c = continuous mode (until key hit)                                      |
|       p = paged mode (<esc> key quits, any other key advances page)            |
|  <bank> = [b|1|2|s|3|p|4] = [Byte/16/32/DM(32)/48/PM(48)/64-bit]               |
|           [i|e|a] = [Internal/External/All] (tests only)                       |
|   <fmt> = [c|d|f|h|i|l] = [Char/Disasm/Float/Hex/Integer/Long]                 |
|  <test> = [s|r|c|b|a] = [Sequential/Random/Checkerboard/Bits/ALL]              |
|                                                                                |
|- EXAMPLES: --------------------------------------------------------------------|
|      Read: diag21k[1]>mr p 0 10                                                |
|      Dump: diag21k[1]>md 2i 0 100 dump.mem                                     |
|     Write: diag21k[1]>mw 2f 1a6 1.23 5 0.01                                    |
| SRAM Test: diag21k[1]>mt ia                                                    |
|   Compare: diag21k[1]>mc s 0x40000 0x50000 0x100                               |
+--------------------------------------------------------------------------------+
```

ml - Memory Load

Syntax **`ml <bank>[<fmt>] addr filename`**

Description The Memory Load command reads **`<fmt>`** formatted ASCII data from **`filename`** (probably created with the Memory Dump command **`-md`**) and writes it to DSP memory bank **`<bank>`** and address **`addr`**. Table 5–1 on page 5-46 gives the syntax for **`<bank>`** and **`<fmt>`**.

Example

```
diag21k[1]>ml si 1000 primes.dmp
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

mr - Memory Read

Syntax **`mr[c|p] <bank>[<fmt>] addr [count]`**

Description The Memory Read command reads **`count`** locations from memory bank **`<bnk>`** and address **`addr`** and displays the values in the **`<fmt>`** format. Table 5–1 on page 5-46 gives the syntax for **`<bank>`** and **`<fmt>`**. By appending **`c`** to the command, you can poll a single location continuously until you hit a key. If you specify a large **`count`**, you can display a single page at a time by appending **`p`** to the command. The default format for program memory disassembles assembly-level code; the default format for other memory banks is hexadecimal.

Examples Read program memory from 20080 to 20084 and display in hex.

`diag21k[1]>mr ph 20080 5`

```
[0002 0080] = 0x0f00:0000:0000
[00020081] = 0x1100:0002:8000
[0002 0082] = 0x1100:0002:8001
[0002 0083] = 0x140a:0001:8000
[0002 0084] = 0x142c:0008:0000
```

Disassemble program memory from 0x8160 to 0x8164.

`diag21k[1]>mr p 0x8160 5`

```
[_main]
08160=1607:ffff:fff6 modify (i7,0xfffffff6);
08161=ad03:ffff:fff5 dm(0xfffffff5,i6)=r3;
08162=ad05:ffff:fff6 dm(0xfffffff6,i6)=r5;
08163=ad06:ffff:fff7 dm(0xfffffff7,i6)=r6;
08164=ad07:ffff:fff8 dm(0xfffffff8,i6)=r7;
```

Read 10 integers starting at location "_primes."

```
diag21k[1]>mr si primes 10
     [00030004] =      2
     [00030005] =      3
     [00030006] =      5
     [00030007] =      7
     [00030008] =     11
     [00030009] =     13
     [0003000A] =     17
     [0003000B] =     19
     [0003000C] =     23
     [0003000D] =     29
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

mt - Memory Test

Syntax **`mt[c][cq] <bank><test>[max_err]`**

Description The Memory Test command performs specified test(s) **`<test>`** on selected memory banks **`<bank>`**. The **`mt`** command should only be used if the board has been reset since turning the computer on and the current processor has been configured. Table 5–1 on page 5-46 gives the syntax for **`<bank>`** and **`<test>`**. The parameter **`max_err`** specifies the maximum number of errors that are reported for each test (the default is 5). If you append **`c`** to the command, the memory test will repeat continuously until you press a key. You can abort the current memory test and all remaining tests by pressing the <esc> key. Pressing any other key will abort only the current memory test.

If you append **`cq`** to the command, Diag21k will perform the same action as with the **`mtc`** command, except that it will perform the test with "quiet" output. The base **`mtc`** command prints out progress and status for each test. Since the **`mtcq`** command only prints out a message that an entire pass of tests has succeeded or every error, the **`mtcq`** command is suitable for long burn-in tests during which **`mtc`** results (good and bad) scroll off the screen quickly. The **`mtcq`** results do not scroll at all unless there are errors.

Examples Test all memory banks, using all tests.

```
diag21k[1]>mt aa
```

Test internal memory banks only.

```
diag21k[1]>mt ia
```

Continuously test external memory using Checkerboard test.

```
diag21k[1]>mtc ec
```

Continuously test external memory with quiet output.

```
Diag21k[1]>mtcq aa
```

```
Memory test has run 2 times
..................***ABORTED***
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

mw - Memory Write

Syntax **`mw[c] <bank>[<fmt>] addr value [count [delta]]`**

Description The Memory Write command writes **`value`** to **`count`** locations, starting at memory bank **`<bank>`** and address **`addr`**. If you append **`c`** to the command, Diag21k will write to a single location until you hit a key. The parameter **`count`** is always decimal (default count is 1). The format for **`value`** is determined by **`<fmt>`**. Table 5–1 on page 5-46 gives the syntax for **`<bank>`** and **`<fmt>`**. If you specify delta, it will be cumulatively added to **`value`** each time it is written.

Example Fill program memory locations 20000–20009 with hex value.

```
diag21k[1]>mw p 20000 1234:5678:abcd 10
```

Write the value 1.23 to “my_float_var.”

```
diag21k[1]>mw lf_my_float_var 1.23
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

os - Operating System

Syntax **`os [command string]`**

Description The Operating System command executes an operating system command or shell. If you do not enter a command string, Diag21k opens a temporary command shell, and you must type “exit” to return to Diag21k.

Example Look for DSP21k executables in current directory.

```
diag21k[1]>os dir *.dxe
```

Open command shell to the operating system.

```
diag21k[1]>os
```

```
Type EXIT to return.
c:\dsp21k\bin>
```

pc - Processor Configure

Syntax **`pc`**

Description The Processor Configure command configures the current processor for memory accesses from the host computer. The processor is configured by writing values to certain IOP registers. The IOP registers that are written by the command are then displayed. Issue this command after the processor has been reset since a processor reset will return these registers to their default values. Issuing this command while the processor is running may have undesired effects. See the entry for dsp21k_cfg_proc in the Host Interface Library manual for more information.

Example **`diag21k[11]>pc`**

```
          processor configured
     SYSCON    (0x00) = 0x0001b000 = 000000330000 = 110592
     WAIT      (0x02) = 0x01ce1b86 = 000163415606 = 30284678
```

pr - Processor Reset

Syntax **`pr`**

Description The Processor Reset command resets the processor. On SHARC processors, this is a software reset that is generated by setting the SRST bit in the SYSCON register. After resetting the processor, you must configure the processor before accessing the DSP again.

Example

```
diag21k[1]>pr
     processor reset
```

ps - Processor Start

Syntax **`ps [DSP command line arg1 [arg2 […]]]`**

Description The Processor Start command command will first attempt to download any parameters as a string to the "__argv_string" variable. If the "__argv_string" variable does not exist or is not large enough to accept all of the command line arguments, then an error will be displayed. If no arguments are passed to the processor start command, Diag21k will not download anything to the __argv_string variable. After any command line parameters have been downloaded, the Processor Startstarts the currently loaded program; execution will begin at the reset vector.

Example

```
diag21k[1]>ps
     processor running
```

q - Quit

Syntax **`q`**

Description The Quit command allows you to leave the program after it resets all active processors.

Example **`diag21k[1]>q`**

```
exiting...resetting processor(s)
c:\dsp21k\bin>
```

reset - Reset Debugger

Syntax `reset`

Description The reset command resets the debugger. The program counter (PC) should be displayed at the reset vector.

Example

```
diag21k[1]>reset
             Debugger reset

       40002=0000:0000:0000 nop;
       40003=0000:0000:0000 nop;
       40004=0000:0000:0000 nop;
     ->40005=063e:0004:0085 jump 0x40085;
       40006=0000:0000:0000 nop;
       40007=0000:0000:0000 nop;

diag21k[1]>
```

restart - Restart Debugger

Syntax `restart`

Description The restart command restarts the debugger. This command is equivalent to first issuing a reset command, and then issuing a run command.

Example

```
diag21k[1]>restart
             Debugger reset
       40002=0000:0000:0000 nop;
       40003=0000:0000:0000 nop;
       40004=0000:0000:0000 nop;
     ->40005=063e:0004:0085 jump 0x40085;
       40006=0000:0000:0000 nop;
       40007=0000:0000:0000 nop;
             Debugger running

diag21k[1]>
```

rr - Register Read

Syntax

```
rr offset[count]
```

Description The Register Read command reads **`count`** locations starting with the 32-bit address **`offset`**. This command reads SharcFIN register space on the processor.

Example

```
diag21k[1]>rr 0x40
     Register (0x40) = 0x00000008 = 8
```

run - Run Debugger (F5)

Syntax `run [address]`

Description The Run Debugger command will run the program until one of three events occur: first, a breakpoint is hit; second, a halt command is issued; or third, the program counter matches the address.

Example

```
diag21k[1]>run main

     Debugger running

diag21k[1]>

       40258=083f:3400:0000 jump (m14,i12) (db);
       40259=717f:0b80:0000 i7=i6;
       4025a=ac16:0000:0000 i6=dm(0,i6);
     [_main]
     ->4025b=1607:ffff:fffc modify (i7,0xfffffffc);
       4025c=ad03:ffff:fffb dm(0xfffffffb,i6)=r3;
       4025d=ad06:ffff:fffc dm(0xfffffffc,i6)=r6;

diag21k[1]>
```

rw - Register Write

Syntax **`rw offset value [count]`**

Description The Register Write command writes **`value`** to **`count`** locations starting with the 32-bit address **`offset`**. This command writes to the SharcFIN register space on the processor.

Example

```
diag21k[1]>rw 0x1c 0x12345678
```

sc - Show Constants

Syntax **`sc`**

Description The Show Constants command lists all of the scripting constants available. Despite being 'constants', the list of available constants and most of the values change depending upon the current processor and the underlying device.

Example Display all scripting constants available.

`diag21k[0]>sc`

```
Constant              Value       Description
--------              -----       -----------
BAR0                  0xfce7fe00  Base Address 0 of this device
BAR1                  0xfb000000  Base Address 1 of this device
BAR2                  0xfce80000  Base Address 2 of this device
BAR3                  0xfc000000  Base Address 3 of this device
BAR4                  0xf8000000  Base Address 4 of this device
BLOCK0_SIZE          0x00000040  Size of internal memory block 0 in kwords
BLOCK0_START          0x00000000  Start address of internal memory block 0
BLOCK1_SIZE          0x00000040  Size of internal memory block 1 in kwords
BLOCK1_START          0x00080000  Start address of internal memory block 1
BLOCK2_SIZE          0x00000040  Size of internal memory block 2 in kwords
BLOCK2_START          0x00100000  Start address of internal memory block 2
BOARD_TYPE                    49  Board type of board this processor is on
DSP_TYPE                      10  Processor type of current processor
ID0                            0  Processor number of ID0 on this board
ID1                            1  Processor number of ID1 on this board
ID2                            2  Processor number of ID2 on this board
ID3                            3  Processor number of ID3 on this board
PART_NUM              TSPC-425-7  Part number of board
SERIAL_NUM                103860  Serial number
SHARED_MEM_BUF        0x00000000  Physical address of last host buffer
SHARED_MEM_BUF_SIZE   0x00000000  Size of last shared memory buffer
Tiger[0]                       0  First processor on 1st Tiger
VERSION                      486  Diag21k scripting version
XMEM2_SIZE            0x00008000  Size of external memory bank 2 in kwords
XMEM2_START           0x04000000  Start address of external memory bank 2
XMEM3_SIZE            0x00000800  Size of external memory bank 3 in kwords
XMEM3_START           0x10400000  Start address of external memory bank 3
```

set - Set Command: View or Modify Diag21k Variables and Board Settings

Syntax **`set [setting [value]]`**

Description The Set command without parameters displays the current processor's board settings and Diag21k variables (variables are preceded with a $). To delete a variable, use only the setting parameter, which is the name of the variable. To create or modify the contents of a variable or modify a board setting, use the **`setting`** parameter for the setting or variable name and the **`value`** parameter for the value you would like to set it to. The **`value`** parameter can be an immediate value, another variable or constant, or the result of a command in parentheses (). See section 5.3 for more information on using variables in Diag21k.

Example Display all variables and board settings.

```
diag21k[1]>set
```

```
   command_output = 0  Display output of commands within ()'s
       debug_addr = (default - not set)  Custom debugger location
      fast_target = 1  Tiger fast target mode
            flyby = 1  Tiger flyby mode
ignore_sht_nobits = 1  Ignore SHT_NOBITS sections when downloading programs
     int_priority = 0  Interrupt thread's OS priority
    reset_on_load = 1  Reset processor on file load
   show_c_symbols = 0  Display C line number and temp variable symbols
     show_logging = 1  Display logging time stamps in log files
     stdio_enable = 0  Enable STDIO handling in non-debug sessions
```

Note *Some settings only apply to specific board types.*

Delete a variable.

```
diag21k[1]>set $count
```

Create or modify the contents of a variable or board setting.

```
diag21k[1]>set reset_on_load 0
diag21k[1]>set $address (mr s _address)
diag21k[1]>set $my_var 0xcabba9e5
```

sh - Script Help

Syntax **`sh`**

Description The script help command displays the scripting commands and syntax.

Example **`diag21k[1]>sh`**

```
+-COMMANDS--------------------------------------SYNTAX------------------------+
|  Script Help - show this screen               sh                            |
| File Execute - execute a Diag21k script file  fx [filename]                 |
| Script Pause - pause for a number of seconds  sp [seconds]                  |
|  Script Wait - wait for a memory location to  sw bank<fmt> addr value       |
|                equal the value given                                        |
|  Script Goto - goto label starting with :     goto <:LABEL>                 |
|      Logging - start, stop, continue or view  log <start|stop|continue|view>|
|                a log file                                                   |
| Show Const's - list the available constants   sc                            |
|-----------------------------------------------------------------------------|
| For the following, expr = $var | (command) | expr <comparison> expr         |
|    Script If - test expression and perform    if expr {TRUE command}       -|
|                TRUE or FALSE command            [{FALSE command}]           |
| Script While - test expression and perform    while expr {command}          |
|                command until failure                                        |
|- EXAMPLES: -----------------------------------------------------------------|
|   Execute: diag21k[1]>fx script1.cmd                                        |
|     Pause: diag21k[1]>sp 5                                                  |
|      Wait: diag21k[1]>sw s _done 1                                          |
|        If: diag21k[1]>if (mr s _done 1) {echo done=1}{echo mr s _done}      |
|     While: diag21k[1]>while $done_value == (mr s _done 1) {sp 1}            |
|   Logging: diag21k[1]>log start diag21k1.log                                |
+-----------------------------------------------------------------------------+
```

sl - Symbol Load

Syntax **`sl filename[.dxe]`**

Description The Symbol Load command loads the symbol table for `filename.dxe` into memory, allowing Diag21k to use the references that `filename.dxe` contains. This command is useful when using Diag21k to debug a program that was loaded in a different session of Diag21k or by another method such as a HIL application or an emulator.

Example **`diag21k[1]>sl blinkall.dxe`**

```
     symbols for “blinkall.dxe” loaded
```

sp - Script Pause

Syntax **`sp [count]`**

Description The Script Pause command pauses the currently executing script for **`count`** seconds, or until you hit a key.

Example

```
diag21k[1]>sp 10

     Pausing 10 seconds.

diag21k[1]>sp

     -- Press any key to continue --
```

ss - Show Symbols

Syntax **`ss [symbol1/addr1 [symbol2/addr2 [...]]]`**

Description If the Show Symbols command is used without parameters, it will display all global symbols for the currently loaded file. If parameters are given, the Show Symbols command will display information only for the given symbols or addresses of symbols. Wildcard characters (*, ?) can be used in the symbol named. Internal C language or debugger symbols are left out of the list by default. These symbols can be shown by turning on the **`show_c_symbols`** setting. See the **`set`** command for more information on board settings. Load a file with the **`fl`** (File Load) command or load just the symbols with the **`sl`** (Symbol Load) command first.

Example **`diag21k[1]>ss primes 0x5002a`**

```
Address]  Bits  Size  Symbol
[05002a]   32    20    "_floats"
[050040]   32    20    "_primes"
```

`diag21k[0]>ss *s`

```
[Address]  Bits  Size  Symbol
[080004]   32    4     "___allones"
[080000]   32    4     "___allzeros"
[000100]   32    0     "___call_ctors"
[08006d]   32    1     "___ctors"
[080038]   32    20    "_chars"
[080024]   32    20    "_dbls"
[080010]   32    20    "_floats"
[080060]   32    1     "_n_primes"
[08000d]   32    1     "_num_reps"
[08004c]   32    20    "_primes
```

step- Step Debugger (F11)

Syntax **`step`**

Description The Step Debugger command steps one assembly instruction.

Example

```
40258=083f:3400:0000 jump (m14,i12) (db);
  40259=717f:0b80:0000 i7=i6;
  4025a=ac16:0000:0000 i6=dm(0,i6);
[_main]
->4025b=1607:ffff:fffc modify (i7,0xfffffffc);
  4025c=ad03:ffff:fffb dm(0xfffffffb,i6)=r3;
  4025d=ad06:ffff:fffc dm(0xfffffffc,i6)=r6;
diag21k[1]>step
diag21k[1]>
  40259=717f:0b80:0000 i7=i6;
  4025a=ac16:0000:0000 i6=dm(0,i6);
[_main]
  4025b=1607:ffff:fffc modify (i7,0xfffffffc);
->4025c=ad03:ffff:fffb dm(0xfffffffb,i6)=r3;
  4025d=ad06:ffff:fffc dm(0xfffffffc,i6)=r6;
  4025e=0f02:ffff:ffff r2=0xffffffff;
```

`diag21k[1]>`

sw - Script Wait

Syntax **`sw <bank>[<fmt>] addr value [milliseconds]`**

Description The Script Wait command pauses the currently executing script, polling and displaying the data at **`addr`** and waiting for it to match **`value`**. The polling stops when the data at **`addr`** equals **`value`** or when you hit a key. The optimal milliseconds parameter determines the polling delay. This command is useful when a script file must wait for a variable to reach some value.

Example

```
diag21k[0]>sw sh 0x80000 1
        [00080000] = 0x00000001
```

Note *This command requires the current processor to be configured. Configure the processor(s) after each board or processor reset.*

uh - Universal Register Help

Syntax `uh`

Description The Universal Register Help command displays the entire list of registers and register bit fields that can be read from and written to using the Universal Register Read (**`ur`**) and Universal Register Write (**`uw`**) commands. A short description of each register and register bit field is also displayed.

Example

```
diag21k[1]>uh
```

ur - Universal Register Read

Syntax **`ur register`**

Description The Universal Register Read command reads and displays the value of **`register`**.

Example

```
diag21k[1]>ur r4
            R4 = 0x000002400
diag21k[1]>ur i0'
            I0' = 0x0
diag21k[1]>ur mode1
            MODE1 = 0x11800
diag21k[1]>ur irq0e
            IRQ0E = 0x0
diag21k[1]>
```

uw - Universal Register Write

Syntax **`uw register value`**

Description The Universal Register Write command writes **`register`** with **`value`**.

Example

```
diag21k[1]>uw irq0e 1
diag21k[1]>ur irq0e
            IRQ0E = 0x1
diag21k[1]>uw r0 0x123456789a
diag21k[1]>ur r0
            R0 = 0x123456789a
diag21k[1]>
```

while - Script While

Syntax `while expr {command}`

Description The Script While command tests the expression, expr. The command or set of commands in curly braces {} is executed while the expression tests true. See section 5.3 for more information on using while statements and expressions in Diag21k.

Example

```
diag21k[1]>while (mr s count) < 0x200 {fx
    write_values.cmd}
      Loop complete (546 iterations)
```

x - eXit

Syntax `x`

Description The Exit command allows you to exit the program with the DSP running.

Example

```
diag21k[1]>x
     c:\dsp21k\bin>
```

This page intentionally left blank.

Chapter 6
Using BitLoader

BitLoader is an FPGA, FPGA EEPROM, and FLASH loader utility. Using either BitLoader for Windows or Linux, you can do the following:

- Load or erase the flash
- Load or unload the FPGA
- Load or erase the FPGA EEPROM

To develop code for the FPGA, refer to your hardware documentation and/or your FPGA developer kit documentation.

6.1 Using the Graphical BitLoader (Windows only)

Two versions of BitLoader are available: a graphical version that is only for Windows-based systems and a second text-based version that can be used on either Linux or Windows. This section describes how to use the graphical version of BitLoader.

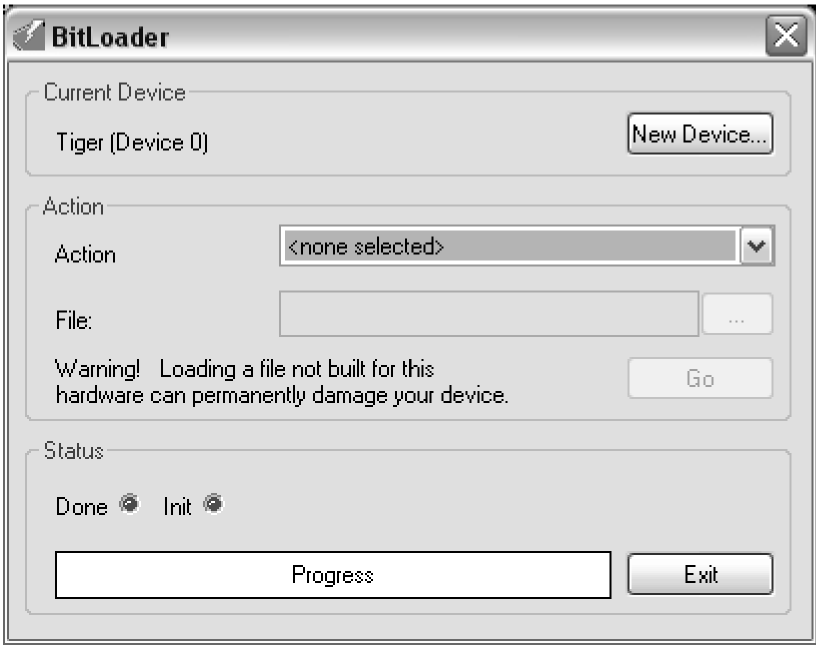
6.1.1 Running BitLoader

Before you run BitLoader to load files into the FPGA, FPGA EEPROM, or FLASH, refer to your specific hardware manual for information on setting configuration switches. Switches may need to be set correctly in order to boot the FPGA from its EEPROM or DSPs from the FLASH.

The graphical version of BitLoader is found in the dsp21ksf/bin directory in Windows. The filename is BitLoader.exe. BitLoader can also be launched via BittWare's menu in the Windows Start Menu.

BitLoader will first display the list of devices in the system. After choosing a device, the BitLoader main window will appear.

Figure 6–1 BitLoader Main Window



Tiger (Device 0)

Action <none selected>

Warning! Loading a file not built for this hardware can permanently damage your device.

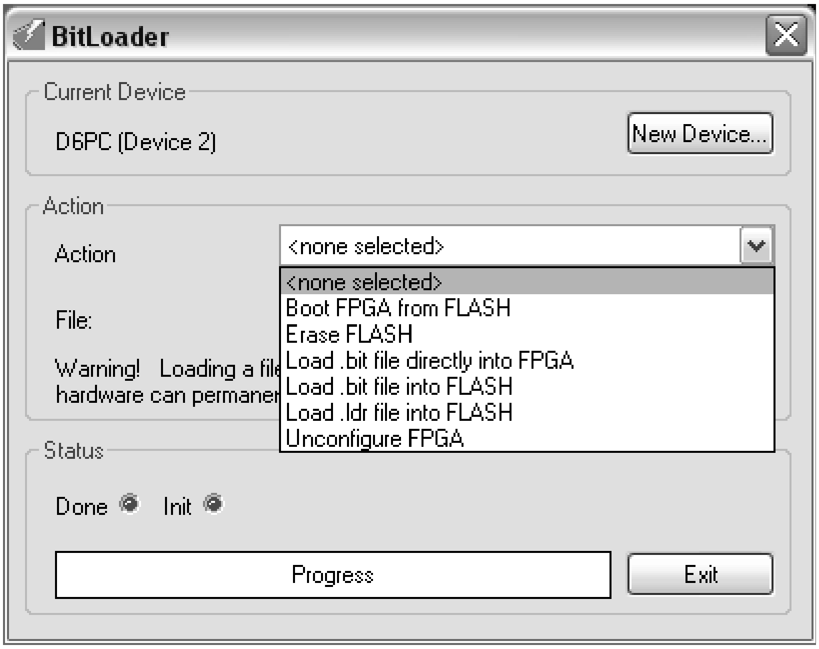
Current Device

The Current Device section of the main window shows which device that BitLoader has opened. The current device can be changed by clicking New Device.

Action

The Action section contains a drop-down list of Actions.

Figure 6–2 The Action Drop-down list



The list of actions available depends on the current device. Each device type supports a range of actions and file types that it supports. The following table describes all of the possible actions at the time this documented was published.

Table 6–1 Descriptions of BitLoader Actions

Action	Description
Boot FPGA from EEPROM/Flash	Boot the FPGA with the FPGA boot image that resides in the FPGA EEPROM or the FLASH.
Unconfigure FPGA	Clear the contents of the FPGA. It will remain cleared until the FPGA next boots.
Load .bit file directly into FPGA	Boot the FPGA with the FPGA boot image that is in the selected .bit file
Load .bit file into Flash	Write the .bit file to Flash* memory to be loaded next time the FPGA boots.
Load .xsvf file into EEPROM	Write the .xsvf file to the FPGA EEPROM to be loaded next time the FPGA boots.
Load .ldr file into Flash	Write the .ldr file into Flash* memory to be loaded next time the DSP(s) boot.
Erase Flash	Erase the entire contents of Flash* memory.

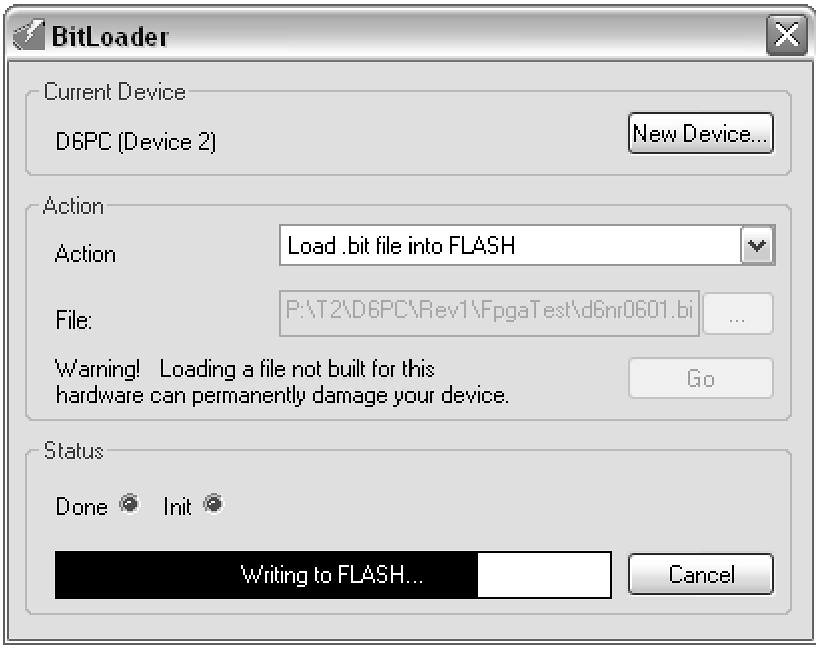
* In some cases, Flash memory is shared between the FPGA boot image and the DSP boot image(s). The FPGA boot image begins at the top of Flash and works its way down, while the DSP boot image(s) start at the bottom and work up. There is no mechanism in BitLoader to prevent the images from overwriting each other.

To perform an action on the current device,

1. Select an action from the list.
2. If the action is loading a file, enter a file path into the File entry field or click on the ... button to browse for a file to load.
3. Click Go.

Note *Some types of actions may not successfully complete on the first attempt and will be automatically retried up to two times by BitLoader.*

Figure 6–3 Status and Action Progress



BitLoader

Current Device

D6PC (Device 2)

Action

Action: Load .bit file into FLASH

File: P:\T2\D6PC\Rev1\FpgaTest\d6nr0601.bi

Warning! Loading a file not built for this hardware can permanently damage your device.

Status

Writing to FLASH...

Status

The Status section shows the green DONE and INIT status LEDs for the device, if the current device supports them. If the current device does not have an FPGA on it, the DONE and INIT status LEDs will remain grayed out.

The progress indicator shows the progress of an action that is currently running. An action can be canceled at any time by clicking on the Cancel button. If an action encounters an error condition, the action will stop and the progress indicator will display the error.

6.2 Using the Text-based BitLoader (for Windows or Linux)

The text-based version of BitLoader is available for both Windows and Linux. This section describes how to use the text-based version of BitLoader.

6.2.1 Running the text-based version of BitLoader

The text-based version of BitLoader can be run using the file bitldr.exe on windows or bitldr on linux and is found in the dsp21ksf/bin directory. Type **`bitldr`** with no parameters to view a list of the command line parameters and quick examples.

Table 6–2 matches each command line switch with an action. Further descriptions of most of the actions are described in Table 6–1 on page 138.

Table 6–2 bitldr Command-Line Switches

Switch	Action
-b <filename>	Load .bit file into FLASH or load .xsvf file into EEPROM
-d <N>	Device to program
-e	Erase Flash
-f <filename>	Load .bit file directly into FPGA
-l <filename>	Load .ldr file into Flash
-p	Boot FPGA from EEPROM/Flash
-r	Reset board(s)
-t <board type number>	Perform action on all boards of type specified
-u	Unconfigure FPGA

6.2.2 Standard Usage

The text-based version of BitLoader is generally used for quickly testing out new boot images for FPGAs or DSP processors on BittWare devices.

BitLoader is also sometimes used to write a new boot image directly to an FPGA when the operating system starts up. If this is what is desired, it is important to let the BittWare driver startup (*`bwcfg -build`* on Windows, or *`bwcfgm -build`* on any other OS) run to completion before launching bitldr.

This page intentionally left blank.

Chapter 7
Using DspGraph

DspGraph is a soft real-time DSP memory graphing utility for Windows. The following sections describe how to use DspGraph.

- Choose the type of memory to graph
- Setting the graph parameters
- Using Advanced data configurations
- Graph scaling
- Viewing a graph of memory

Note *DspGraph is compatible with Windows only.*

7.1 Choosing the Type of Memory to Graph

You can choose to graph either processor memory or a host DMA buffer that has been previously allocated in another application.

7.1.1 Graphing processor memory

To graph processor memory,

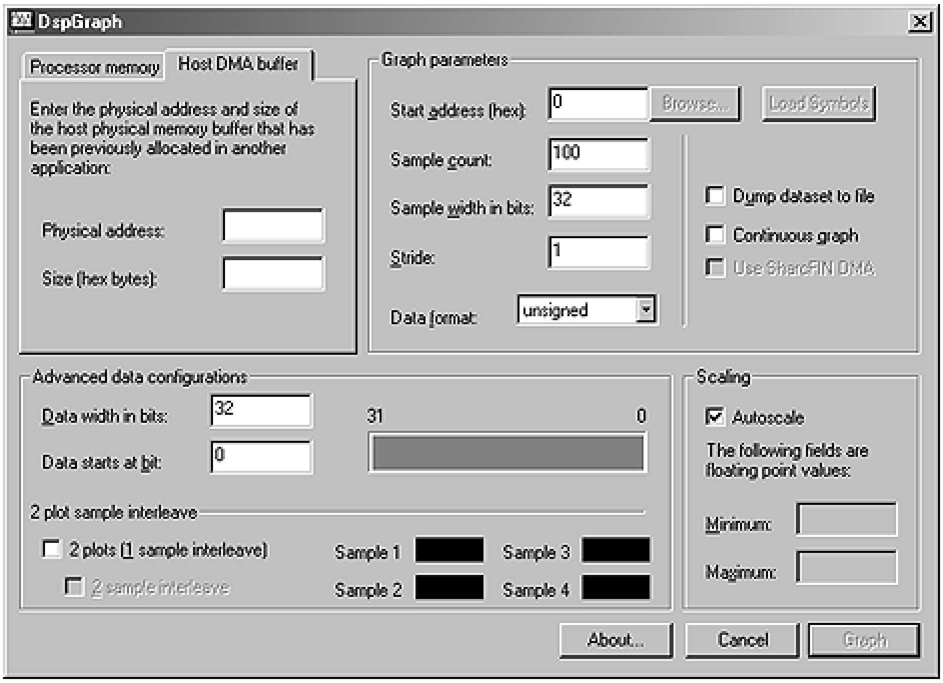
1. Select the Processor memory tab.
2. Scroll the list until you find the processor you wish to use. You must select a processor and not a device, even if you want to graph external memory.

7.1.2 Graphing a previously allocated host memory buffer

To graph a previously allocated host memory buffer,

1. Select the Host DMA buffer tab.
2. Enter the physical address of host physical memory buffer that has been previously allocated in another application.
3. Enter the size in bytes in hexadecimal format of the physical memory buffer. This size must be greater or equal to the size you want to graph as determined by the start address (offset into the buffer), sample count, sample width and stride.

Figure 7–1 Host DMA Buffer


DspGraph

Processor memory | Host DMA buffer

Enter the physical address and size of the host physical memory buffer that has been previously allocated in another application:

Physical address:

Size (hex bytes):

Graph parameters

Start address (hex): 0

Sample count: 100

Sample width in bits: 32

Stride: 1

Data format: unsigned

☐ Dump dataset to file

☐ Continuous graph

☐ Use SharcFIN DMA

Advanced data configurations

Data width in bits: 32

Data starts at bit: 0

2 plot sample interleave

☐ 2 plots (1 sample interleave)

☐ 2 sample interleave

Scaling

☑ Autoscale

The following fields are floating point values:

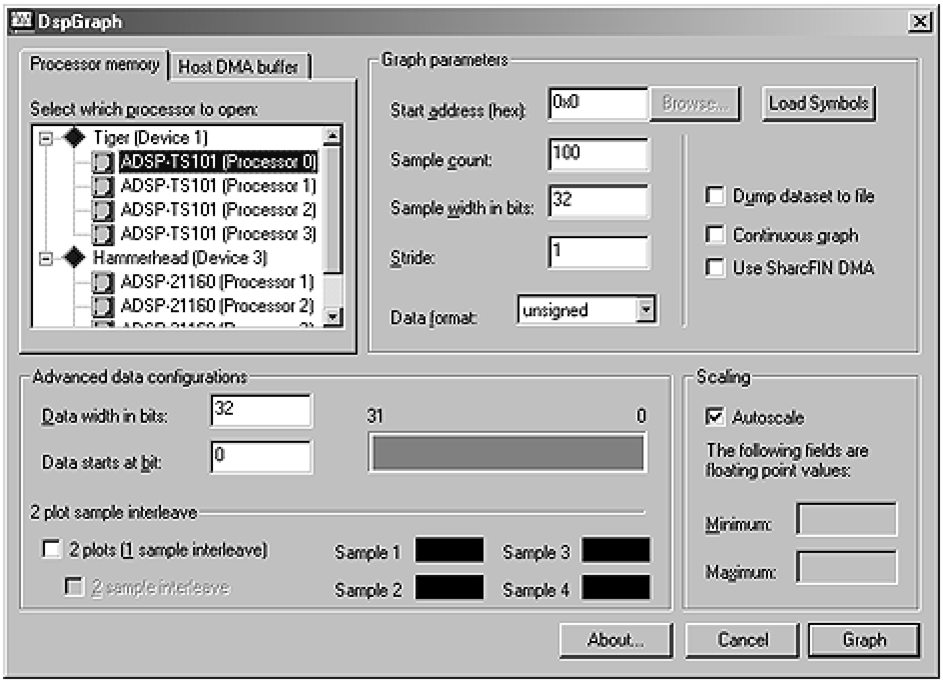
Minimum:

Maximum:

7.2 Setting the DspGraph Parameters

To set the graph parameters, follow the steps described in the sections below. Refer to Figure 7–2 below for an illustration of the main screen in DspGraph.

Figure 7–2 DSPGraph Main Screen



Select which processor to open:

- Tiger (Device 1)
  - ADSP-TS101 (Processor 0)
  - ADSP-TS101 (Processor 1)
  - ADSP-TS101 (Processor 2)
  - ADSP-TS101 (Processor 3)
- Hammerhead (Device 3)
  - ADSP-21160 (Processor 1)
  - ADSP-21160 (Processor 2)

Start address (hex): 0x0
Sample count: 100
Sample width in bits: 32
Stride: 1
Data format: unsigned

☐ Dump dataset to file
☐ Continuous graph
☐ Use SharcFIN DMA

Data width in bits: 32
Data starts at bit: 0

☐ 2 plots (1 sample interleave)
☐ 2 sample interleave

☑ Autoscale

The following fields are floating point values:

Minimum:
Maximum:

Step 1: Enter the start address

For graphing processor memory, the Start address is a DSP address or the same address you would pass to an upload or download function in the Host Interface Library. For graphing a host DMA buffer, the Start address is a 32-bit offset into the buffer. It is usually set to 0 in this case.

Step 2: Enter the sample count

The sample count is the actual number of samples to graph.

Step 3: Enter the sample width in bits

The sample width must be greater than or equal to the data width. The sample width cannot exceed 64 bits.

Step 4: Enter the stride

The stride determines how many samples are skipped. A stride of 2 will skip a sample after every sample read. A stride of 0 will continuously read the same sample.

Step 5: Choose the data format

Floating point format uploads 32-bit floating point data (data width must be 32 bits). Signed format treats the leftmost data bit as signed.

Step 6: Dump dataset to file

You can select Dump Dataset to File if you want to save the data set to a file. The data format determines how the data will be saved. For continuous graphs, only the first data set uploaded will get saved to the file.

Step 7: Perform a soft real-time graph

To perform a soft real-time graph, check Continuous Graph. You can also unpause a non-continuous graph in the graph window to make it a continuous graph. Likewise, a continuous graph can be paused. When a continuous graph is selected, DspGraph continuously uploads data sets as fast as possible.

Step 8: Selecting HIL DMA Functionality

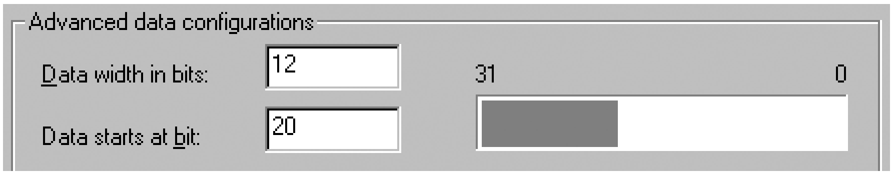
For devices that are supported by the Host Interface Library DMA functions, the Use SharcFIN DMA box can be checked. Selecting this tells DspGraph to use the HIL DMA functions to upload rather than using the normal HIL upload functions.

7.3 Using Advanced Data Configurations

7.3.1 Specifying the Sample Range

Enter the data width and data starts at bit. These fields are supplied so that if your data is located in just part of a sample, you can graph that part of the sample. For instance, if you are only interested in the last 12 bits of samples that are 32 bits wide, you would set the data width to 12 and the data starts at bit field to bit 20. The location of the data inside the sample is shown to the right of the fields. A white bar depicts the sample and a gray bar inside shows where the data is inside the sample. The sides of the gray bar can be dragged to move the data width and the data starts at bit fields.

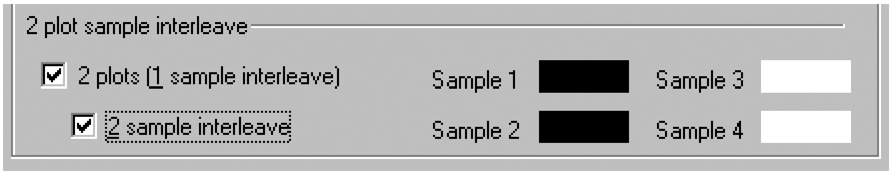
Figure 7–3 DspGraph Advanced Data Configuration


Advanced data configurations
Data width in bits: 12
Data starts at bit: 20

7.3.2 Plotting Multiple Data Sets

DspGraph also has some limited ability to plot multiple data sets. If the memory contains data from two separate channels, two plot sample interleave can be used to add a plot to the graph. The sample boxes to the right of the checkboxes show plot one samples in black and plot two samples in white. One sample interleave will send odd samples to plot one and even samples to plot two. If two sample interleave is checked, the first two samples get sent to plot one, the next two samples to plot two and so on.

Figure 7–4 Sample Interleave


2 plot sample interleave
[✓] 2 plots (1 sample interleave)
[✓] 2 sample interleave

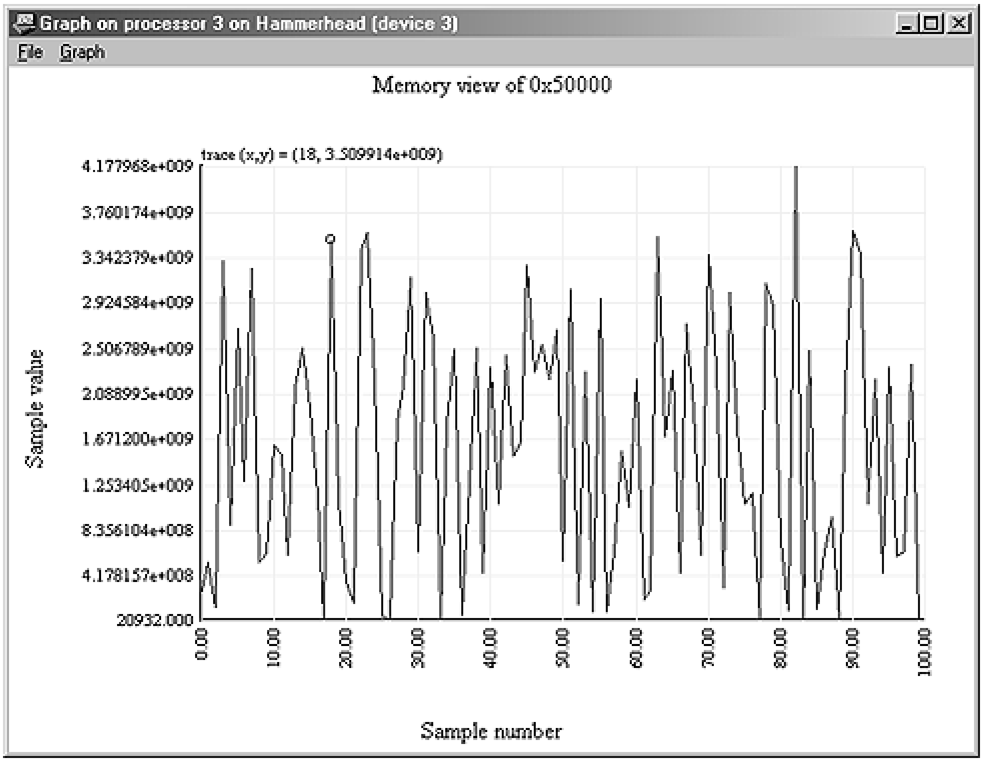
7.4 Graph Scaling

Graph scaling determines the values shown on the Y axis of the graph. If autoscale is checked, DspGraph will find the smallest and largest values in the first data set uploaded and scale the graph accordingly. To specify the minimum and maximum values for the Y axis, uncheck autoscale and enter the minimum and maximum values as floating point numbers.

7.5 Viewing a graph of memory

Click the Graph button to open a window displaying a graph of the memory. See Figure 7–5 for an example of a DspGraph graph window. In the graph window the following actions can be taken:

Figure 7–5 DspGraph Graph Window



Action 1: Print the graph

Choose File->Print from the menu. The graph will be scaled to the page.

Action 2: Pause/unpause the graph

Choose Graph->Pause to pause or unpause the graph. Pausing the graph stops DspGraph from uploading samples continuously. Unpausing the graph causes DspGraph to continuously upload more samples and display them.

Action 3: Expand/shrink the graph window

The graph window can be expanded or shrunk as desired by dragging any part of the edge of the window.

Action 4: Trace Points

Use the arrow keys to move the trace circle left and right across the plot. The values are displayed at the upper left corner of the graph.

Action 5: Estimate values

The mouse pointer can be hovered over any part of the plot to estimate the x, y values at that location. The values are displayed at the upper left corner of the graph.

Action 6: Open new graphs

By moving the graph window over or minimizing the graph window, you can use the main DspGraph screen to open more graphs. Only one graph per processor is allowed. Multiple continuous graphs will not perform as well as single continuous graphs.

Action 7: Close the graph window

To close the graph window, click the X in the upper right corner of the window.

Chapter 8
Using the BittWorks Server

The BittWorks Server program is a TCP/IP server for any BittWare Remote Client Toolkit application or the BittWare VisualDSP Remote Target or any program communicating the Client HIL Interface Protocol (CHILI Protocol). The BittWorks Server comes standard with the DSP21k-SF Toolkit because a toolkit needs to be installed on the machine the BittWorks Server runs on. The BittWorks Server will not be useful without a client program communicating with it. The following sections describe how to launch and use the BittWorks Server program.

8.1 Launching the BittWorks Server

The BittWorks Server accepts command-line switches that control various start-up options. The general syntax for the BittWorks Server is:

```
bwserver [switches]
```

The command-line switches are not case-sensitive and are preceded with "-" or "/". Table 8–1 below describes each switch.

Table 8–1 HIL Server Command-Line Switches

Switch	Description
-?	Show usage
-h	Show usage
-p port	Specify server port number (default is 4794)
-s	Run as a service (no console)
-v	Show version information

8.1.1 Running the BittWorks Server as a Service

The BittWorks Server can be run without a user interface by using the **`-s`** command-line switch. When the BittWorks Server is run with this option, the CTRL-C key combination can be used to shut it down cleanly.

8.1.2 Specifying a Port Number

To specify a port number other than the default of 4794, use the **`-p`** command-line switch. The BittWorks Server will listen to the specified port and any remote clients must connect using the same port number. For Remote Client Toolkit programs, the port number can be specified using the :<**`port`**> suffix to the network address (IP address), where <**`port`**> is replaced with the port number. For example, if the server is at address 192.168.1.50 and is launched with the **`-p 8765`** option, the remote clients can connect using the string "192.168.1.50:8765".

8.2 BittWorks Server commands

If the BittWorks Server is launched normally (without the **`-s`** command-line option), the following commands are available at the prompt:

- menu
- ports
- shutdown
- trace
- transfers

menu

The menu command displays the possible commands and short descriptions for each.

ports

The ports command displays the port number that the server is listening to and the network addresses and outgoing port numbers of each connected client. If the network address of the server is not known, run **`ifconfig`** for Linux or **`ipconfig`** for Windows to get the address assigned to the network device.

shutdown

This command shuts down the server cleanly if no clients are connected. If any connections are currently open, the BittWorks Server will display the number of connected clients and ask if a shutdown is really desired. If the server is shut down while a remote client program is connected, it will stop operating correctly or crash.

trace

Use the trace command to toggle the function trace feature. When the function trace feature is on, the BittWorks Server will display a line of text containing the name of the function that has been called by a client program. The trace feature will display documented and undocumented function names for debug and support purposes.

transfers

The transfer command displays the number of bytes sent and received since the server was started. This command can be useful to show activity on the server.

8.3 BittWorks Server Usage Notes

Multiple clients

Since the BittWorks Server only uses a single instance of the libraries that it attaches to, care must be taken to ensure that one remote client does not clobber data that another remote client is accessing.

For example, remote client A opens up processor 1. Remote client B also opens up processor 1. Remote client A then closes processor 1 using a call to dsp21k_close_all, which closes every open processor regardless of how many times it was open. When remote client B tries to access processor 1, it no longer exists and remote client B crashes.

Authentication and Security

The BittWorks Server does not use any authentication or security schemes. As such, BittWare cannot be held responsible for any breaches in security. See Disclaimer below.

Protocol

The BittWorks Server uses the Client HIL Interface Protocol (CHILI Protocol). The CHILI Protocol is a simple specification layered on top of TCP/IP for remotely accessing BittWare boards using the Host Interface Library portion of the BittWorks Server. The CHILI Protocol describes the data to be sent and received by each HIL function. Using the CHILI Protocol directly, it is possible to port the HIL to any programming language that can use TCP/IP network sockets. For more information on the CHILI Protocol, please contact BittWare Technical Support.

Disclaimer

THIS SOFTWARE IS PROVIDED BY BITTWARE, INC. (BITTWARE) “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL BITTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,

Index