

VISUALDSP++[®] 3.5

User's Guide

for 32-Bit Processors

Revision 1.0, March 2004

Part Number
82-000035-08

Analog Devices, Inc.
One Technology Way
Norwood, Mass. 02062-9106



Copyright Information

© 2004 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

Analog Devices, Inc. reserves the right to change this product without prior notice. Information furnished by Analog Devices is believed to be accurate and reliable. However, no responsibility is assumed by Analog Devices for its use; nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under the patent rights of Analog Devices, Inc.

Trademark and Service Mark Notice

The Analog Devices logo, the CROSSCORE logo, TigerSHARC, SHARC, EZ-ICE, and VisualDSP++ are registered trademarks of Analog Devices, Inc.

Apex-ICE, EZ-KIT Lite, and Summit-ICE are trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

CONTENTS

PREFACE

Purpose of This Manual	xxi
Intended Audience	xxi
Manual Contents	xxii
What's New in This Manual	xxiii
Technical or Customer Support	xxiii
Supported Processors	xxiv
Product Information	xxiv
MyAnalog.com	xxv
Processor Product Information	xxv
Related Documents	xxvi
Online Technical Documentation	xxvii
From VisualDSP++	xxviii
From Windows	xxviii
From the Web	xxix
Printed Manuals	xxix
VisualDSP++ Documentation Set	xxix
Hardware Tools Manuals	xxix
Processor Manuals	xxix

CONTENTS

Data Sheets	xxx
Notation Conventions	xxx

INTRODUCTION TO VISUALDSP++

VisualDSP++ Features	1-2
Integrated Development and Debugging	1-2
Code Development Tools	1-2
Source File Editing Features	1-3
Project Management Features	1-4
Debugging Features	1-5
VDK Features	1-6
VisualDSP++ 3.5 Features	1-7
License Management	1-11
Licensing Options	1-11
License Status	1-12
Temporary Licenses	1-12
Valid Versus Expired Licenses	1-12
Client Licenses	1-13
License Installation	1-13
Installing a License Shipped With EZ-KIT Lite	1-13
Installing a Single-User License	1-14
Installing a Server License	1-15
Installing a Client License	1-15
Software Registration	1-16
Validation Codes	1-16

Product Upgrades	1-16
Product Serial Numbers	1-17
Project Development	1-17
Overview of Programming With VisualDSP++	1-17
DSP Project Development Stages	1-20
Simulation	1-20
Evaluation	1-21
Emulation	1-21
Targets	1-21
Simulation Targets	1-21
EZ-KIT Lite Targets	1-22
Emulation Targets	1-22
Platforms	1-22
Hardware Simulation	1-23
Debugging Overview	1-23
VisualDSP++ Kernel	1-25
Program Development Steps	1-25
Step 1: Create a Project	1-26
Step 2: Configure Project Options	1-26
Step 3: Add and Edit Project Source Files	1-26
Adding Files to Your Project	1-26
Creating Files to Add to Your Project	1-27
Editing Files	1-27
Managing Project Dependencies	1-27

CONTENTS

Step 4: Define Project Build Options	1-27
Configuration	1-28
Project-Wide File and Tool Options	1-28
Individual File and Tool Options	1-28
Step 5: Build a Debug Version of the Project	1-29
Step 6: Create a Debug Session and Load the Executable ...	1-29
Step 7: Run and Debug the Program	1-29
Step 8: Build a Release Version of the Project	1-29
Code Development Tools	1-30
Compiler	1-31
C++ Run-Time Libraries	1-32
Assembler	1-33
Linker	1-34
Expert Linker	1-37
Expert Linker Window	1-39
Memory Map Pane Right-Click Menu	1-40
Stack and Heap Usage	1-42
Archiver	1-44
Splitter	1-44
Loader	1-45
VCSE	1-47
VCSE Components	1-47
VCSE Component Model Specification	1-48
VCSE Component Model	1-48

VCSE Tools	1-49
Use of VCSE Components With VisualDSP++	1-49
VCSE User Interface	1-50
Tool Chain Integration	1-50
Wizards	1-51
Component Manager	1-51
Structure of VCSE	1-52
Interface Definition Language (IDL) and Compiler	1-54
DSP Projects	1-57
Project Options	1-58
Project Groups	1-59
Source Code Control (SCC)	1-61
Makefiles	1-62
Rules	1-63
Output Window	1-63
Example Makefile	1-64
Project Configurations	1-67
Customized Project Configurations	1-68
Project Build	1-68
Build Options	1-69
File Building	1-70
Post-Build Options	1-70
Command Syntax	1-71
Project Dependencies	1-71

CONTENTS

Project Rules	1-72
VisualDSP++ Help System	1-73

ENVIRONMENT

Parts of the User Interface	2-1
Title Bar	2-3
Additional Information in Title Bars	2-4
Title Bar Right-Click Menus	2-4
Control Menu	2-5
Program Icons	2-5
Editor Windows	2-5
Debugging Windows	2-6
Menu Bar	2-6
Command Information	2-7
Toolbars and User Tools	2-7
Built-In Toolbars	2-8
Toolbar Customization	2-9
Toolbars: Docked Versus Floating	2-9
Toolbar Button Appearance	2-10
Toolbar Shape	2-12
Toolbar Rules	2-12
User Tools	2-13
Status Bar	2-13
VisualDSP++ Windows	2-15
Project Window	2-15

Project View	2-16
Project Dependencies	2-17
Project Nodes	2-18
Project Page Right-Click Menus	2-19
Project Group Icon Right-Click Menu	2-19
Project Icon Right-Click Menu	2-20
Folder Icon Right-Click Menu	2-21
File Icon Right-Click Menu	2-21
Project Folders	2-22
Project Files	2-23
Project Window Icons for Source Code Control (SCC)	2-24
File Associations	2-25
Automatic File Placement	2-26
File Placement Rules	2-26
Example	2-27
Kernel Page	2-27
Editor Windows	2-29
Right-Click Menu	2-30
Editor Tab Mode	2-31
Output Window	2-32
Output Window Tabs	2-32
Build Page	2-33
Console Page	2-33
Output Window Error Messages	2-34

CONTENTS

Error Message Severity Hierarchy	2-35
Syntax of Help for Error Messages	2-35
How to Promote, Demote, and Suppress Error Messages	2-37
Log File	2-41
Output Window Customization	2-42
Right-Click Menu	2-43
Script Command Output	2-44
Window Operations	2-46
Window Manipulation	2-46
Right-Click Menu Options	2-46
Scroll Bars and Resize Pull-Tab	2-47
Windows: Docked vs. Floating	2-47
Example of a Docked Window	2-48
Examples of Floating Windows	2-49
Window Position Rules	2-50
Standard Windows Buttons	2-51
Debugging Windows	2-52
Editor Windows	2-54
Syntax Coloring	2-54
Right-Click Menu	2-55
Editor Window Symbols	2-56
Bookmarks	2-56
Context-Sensitive Expression Evaluation	2-56
Viewing an Expression	2-57

Highlighting an Expression	2-57
Source Mode Versus Mixed Mode	2-57
Source Mode	2-57
Mixed Mode	2-58
Disassembly Windows	2-59
Other Disassembly Window Features	2-61
Right-Click Menu	2-62
Disassembly Window Symbols	2-63
Expressions Window	2-64
Right-Click Menu	2-65
Expressions in an Expression Window	2-66
Trace Windows	2-67
Locals Window	2-69
Statistical/Linear Profiling Results Window	2-70
Window Components	2-71
Left Pane	2-71
Right Pane	2-72
Status Bar	2-72
Right-Click Menu	2-72
Window Operations	2-74
Changing the Window View	2-74
Displaying a Source File	2-74
Displaying Functions in Libraries	2-75
Working With Ranges	2-75

CONTENTS

Switching Display Modes	2-76
Filtering PC Samples With No Debug Information	2-78
Call Stack Window	2-79
Memory Windows	2-79
Memory Number Formats	2-80
Right-Click Menu	2-82
Expression Tracking in a Memory Window	2-84
Memory Window Display Customization	2-86
Background Telemetry Channel (BTC) Window	2-86
BTC Definitions in Your Program	2-87
How to Enable BTC on ADSP-2126x Processors	2-88
BTC Priority	2-89
Examples	2-90
Right-Click Menu	2-92
Memory Map Windows	2-93
Register Windows	2-94
Stack Windows	2-97
Custom Registers Window	2-97
Multiprocessor Window	2-98
Multiprocessor Window Pages	2-99
Status Page	2-99
Groups Page	2-100
Multiprocessor Groups	2-100
Focus	2-101

Right-Click Menu	2-101
Pipeline Viewer Window	2-102
Right-Click Menu	2-103
Pipeline Viewer Properties Dialog Box	2-104
Pipeline Viewer Window Event Icons	2-105
Pipeline Instruction Event Details	2-106
Cache Viewer	2-107
Configuration Page	2-110
Detailed View Page	2-111
History Page	2-112
Performance Page	2-114
Histogram Page	2-115
Address View Page	2-116
VDK Status Window	2-117
VDK State History Window	2-119
Thread Status and Event Colors	2-120
Window Operations	2-121
Right-Click Menu	2-121
Target Load Window	2-122
Plot Windows	2-123
Plot Window Features	2-124
Status Bar	2-124
Tool Bar	2-125
Right-Click Menu	2-127

CONTENTS

Plot Window Statistics	2-129
Plot Configuration	2-130
Plot Window Presentation	2-131
Plot Presentation Options	2-133
Image Viewer	2-134
Right-Click Menu	2-136
Image Configuration Dialog Box	2-137
Gamma Correction Dialog Box	2-137
Export Image Dialog Box	2-138

DEBUGGING

Debug Sessions	3-2
Debug Session Management	3-3
Simulation Versus Emulation	3-3
Breakpoints	3-3
Watchpoints	3-4
Multiprocessor (MP) Debugging	3-4
Setting Up a Multiprocessor Debug Session	3-4
Debugging a Multiprocessor System	3-5
Focus and Pinning	3-5
Window Title Bar Information	3-6
Additional Focus Indication	3-7
Code Analysis Tools	3-7
Statistical Profiles and Linear Profiles	3-7
Simulation	3-8

Emulation	3-8
Traces	3-9
DSP Memory Plots	3-10
Program Execution Operations	3-11
Selecting a New Debug Session at Startup	3-11
Loading the DSP Executable Program	3-12
Using Program Execution Commands	3-12
Restarting the Program	3-13
Performing a Restart During Simulation	3-13
Performing a Restart during Emulation	3-14
Using Breakpoints	3-14
Using Unconditional and Conditional Breakpoints	3-15
Using Watchpoints	3-15
Using Hardware Breakpoints	3-16
Latency	3-16
Restrictions	3-17
Simulation Tools	3-17
Interrupts	3-17
Input/Output Simulation (Data Streams)	3-17
Image Viewer	3-19
Plots	3-19
Plot Types	3-19
Line Plots	3-21
X-Y Plots	3-22

CONTENTS

Constellation Plots	3-23
Eye Diagrams	3-24
Waterfall Plots	3-25
Spectrogram Plots	3-27
Flash Programmer	3-28
Flash Devices	3-28
Flash Programmer Functions	3-28
Flash Driver	3-29
Flash Programmer Window	3-30

REFERENCE INFORMATION

Glossary	A-2
File Types	A-24
Keyboard Shortcuts	A-27
Working With Files	A-27
Moving Within a File	A-28
Cutting, Copying, Pasting, Moving Text	A-29
Selecting Text Within a File	A-29
Working With Bookmarks in an Editor Window	A-30
Building Projects	A-31
Using Keyboard Shortcuts for Program Execution	A-31
Working With Breakpoints	A-32
Obtaining Online Help	A-32
Miscellaneous	A-32
IDDE Command-Line Parameters	A-33

Extensive Scripting	A-34
Toolbar Buttons	A-38
Text Operations	A-43
Regular Expressions Versus Normal Searches	A-43
Specific Special Characters	A-44
Special Rules for Sequences	A-45
Repetition and Combination Characters	A-45
Match Rules	A-46
Tagged Expressions in Replace Operations	A-46
Comment Start and Stop Strings	A-47
Online Help Features and Operations	A-48
Using the Help Window	A-48
Invoking Online Help	A-49
Viewing Context-Sensitive Help	A-50
Viewing Menu, Toolbar, or Window Help	A-51
Viewing Dialog Box Button or Field Help	A-51
Viewing Window Help	A-52
Using Help Window Navigation Buttons	A-52
Copying Example Code From Help	A-53
Printing Help	A-54
Bookmarking Frequently Used Help Topics	A-54
Placing a Bookmark at a Topic	A-55
Opening a Bookmarked Topic	A-55
Navigating in Online Help	A-55

CONTENTS

Using the Search Features	A-57
Help System Search Rules	A-57
Rules for Full Text Searches	A-57
Rules for Advanced Searches	A-58
Full Text Searches	A-58
Advanced Search Techniques	A-60
Using Wildcard Expressions	A-60
Using Boolean Operators	A-61
Using Nested Expressions	A-62
Viewing Online Manuals	A-62
Printing Online Documents	A-63
Using the About VisualDSP++ Dialog Box	A-64

SIMULATION OF TIGERSHARC PROCESSORS

ADSP-TS101 Processors	B-1
Simulator Timing Analysis Overview	B-2
Pipeline Stages	B-2
Stalls	B-3
Stalls Due to IALU Dependency	B-4
Stalls Due to Compute Block Dependency	B-5
Aborts	B-6
Aborts Due to an Unpredicted Change of Flow	B-6
Abort Due to Mispredicted Change of Flow	B-7
Branch Target Buffer Hits	B-8
Pipeline Viewer and Disassembly Window Operations	B-8

Current Program Counter Value	B-9
Stepping	B-11
Simulator Options	B-12
ADSP-TS20x Processors	B-13
Simulator Timing Analysis Overview	B-13
Pipeline Stages	B-14
Stalls	B-15
Stalls Due to IALU Dependency	B-15
Stalls Due to Compute Block Dependency	B-16
Stalls Due to a Cache Miss	B-17
Aborts	B-17
Aborts Due to an Unpredicted Change of Flow	B-18
Abort Due to Mispredicted Change of Flow	B-19
Branch Target Buffer Hits	B-20
Pipeline Viewer and Disassembly Window Operations	B-20
Current Program Counter Value	B-21
Stepping	B-23
Simulator Options	B-23

SIMULATION OF SHARC PROCESSORS

Anomaly Options	C-2
ADSP-21x6x Anomalies	C-2
Shadow Write FIFO Anomaly (ADSP-2116x Only)	C-2
SIMD Read from Internal Memory With Shadow Write FIFO Hit Anomaly (ADSP-2116x Only)	C-3

CONTENTS

Event Options C-4

 FP Denorm C-4

 Short Word Anomaly C-4

 Access to ADSP-21065L Short Word Internal Memory

 9th Column at Even Addresses C-7

How to Record a Simulator Anomaly or Event C-8

Select Processor ID Options C-10

Simulator Options (ADSP-21161Only) C-10

Load Sim Loader Options C-11

SPI Simulation in Slave Mode C-12

INDEX

PREFACE

Thank you for purchasing VisualDSP++[®], the development software for Analog Devices processors.

Purpose of This Manual

The *VisualDSP++ 3.5 User's Guide for 32-Bit Processors* describes the features, components, and functions of VisualDSP++. Use this guide as a reference for developing programs for SHARC[®] and TigerSHARC[®] processors.

The User's Guide does not include detailed procedures for building and debugging projects. For how-to information, refer to the VisualDSP++ online Help and the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors*.

Intended Audience

This manual is primarily intended for digital signal processing (DSP) programmers who are familiar with Analog Devices processors, but are unfamiliar with the VisualDSP++ environment. The manual assumes that the audience has a working knowledge of the target processor's architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors should supplement this manual with other texts (such as the Hardware Reference and Instruction Set Reference manuals that describe the target's architecture and instruction set).

Manual Contents

The manual's contents are summarized as follows.

- Chapter 1, “Introduction,” describes VisualDSP++ features, license management, project development, code development tools, VCSE, and DSP projects; also provides a brief introduction to the VisualDSP++ Help system.
- Chapter 2, “Environment,” describes the VisualDSP++ user interface, windows, environment customization, window operations, and the debugging windows.
- Chapter 3, “Debugging,” describes debug sessions, code analysis tools, program execution operations, simulation tools, Image Viewer, plots, and Flash Programmer.
- Appendix A, “Reference Information,” provides a glossary and information about file types, keyboard shortcuts, command-line parameters, scripting, toolbar buttons, and text operations; also provides details about online Help features and operations.
- Appendix B, “Simulation of TigerSHARC Processors,” describes simulator instruction timing analysis, pipeline stages, the Pipeline Viewer, stalls, aborts, the current program counter value, stepping, and the **Select Loader Program** command on the **Simulator** sub-menu under **Settings**.
- Appendix C, “Simulation of SHARC Processors,” describes the simulator options available on the **Anomalies**, **Events**, **Simulator**, **Load Sim Loader**, and **Select Processor ID** submenus under **Settings**; also explains how to record simulator anomalies and events, and describes SPI simulation in slave mode.

What's New in This Manual

The *VisualDSP++ 3.5 User's Guide for 32-Bit Processors* supports the new SHARC processors (ADSP-21261, ADSP-21262, ADSP-21266, ADSP-21267, ADSP-21363, ADSP-21364, and ADSP-21365) and the new TigerSHARC processors (ADSP-TS202 and ADSP-TS203). This edition also documents new features and enhancements. For information about all new and updated VisualDSP++ features, see the *VisualDSP++ 3.5 Product Release Bulletin for 32-Bit Processors*.

Technical or Customer Support

You can reach Analog Devices, Inc. Customer Support in these ways:

- Visit the Embedded Processing and DSP products Web site at:
<http://www.analog.com/processors/technicalSupport>
- E-mail tools questions to:
dsptools.support@analog.com
- E-mail processor questions to:
dsp.support@analog.com
- Phone questions to: **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor.
- Send questions by mail to:

Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
USA

Supported Processors

The names “*SHARC*” and “*TigerSHARC*” refer to the family of Analog Devices 32-bit, digital signal processors.

VisualDSP++ currently supports the following SHARC processors.

ADSP-21020	ADSP-21261
ADSP-21060	ADSP-21262
ADSP-21061	ADSP-21266
ADSP-21062	ADSP-21267
ADSP-21065L	ADSP-21363
ADSP-21160	ADSP-21364
ADSP-21161	ADSP-21365

VisualDSP++ currently supports the following TigerSHARC processors.

ADSP-TS101	ADSP-TS202
ADSP-TS201	ADSP-TS203

Product Information

You can obtain product information from the Analog Devices Web site, from the product CD-ROM, or from the printed publications (manuals).

Analog Devices is online at www.analog.com. Our Web site provides information about a broad range of products—analog integrated circuits, amplifiers, converters, and digital signal processors.

MyAnalog.com

MyAnalog.com, a free feature of the Analog Devices Web site, allows customization of a Web page to display only the latest information on products that you are interested in. You can also choose to receive weekly E-mail notifications containing updates to the Web pages that meet your interests. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Registration

Visit www.myanalog.com to sign up. Click **Register** to use MyAnalog.com. Registration takes about five minutes and serves as means to select the information that you want to receive.

If you are already a registered user, just log on. Your user name is your E-mail address.

Processor Product Information

For information about embedded processors and DSPs, visit our Web site at www.analog.com/processors. This site provides access to technical publications, data sheets, application notes, product overviews, and product announcements.

Product Information

You may also obtain additional information about Analog Devices and its products in the following ways.

- E-mail questions or requests for information to:
dsp.support@analog.com
- Fax questions or requests for information to:
1-781-461-3010 (North America)
089/76 903-557 (Europe)
- Access the FTP Web site at:
[ftp ftp.analog.com](ftp://ftp.analog.com) **or** [ftp 137.71.23.21](ftp://137.71.23.21)
<ftp://ftp.analog.com>

Related Documents

For information on product related development software, see the following publications.

VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors

VisualDSP++ 3.5 C/C++ Compiler and Library Manual for SHARC Processors

VisualDSP++ 3.5 C/C++ Compiler and Library Manual for TigerSHARC Processors

VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors

VisualDSP++ 3.5 Assembler and Preprocessor Manual for SHARC Processors

VisualDSP++ 3.5 Assembler and Preprocessor Manual for TigerSHARC Processors

VisualDSP++ 3.5 Loader Manual for 32-Bit Processors

VisualDSP++ 3.5 Product Release Bulletin for 32-Bit Processors

VisualDSP++ 3.5 Kernel (VDK) User's Guide for 32-Bit Processors

VisualDSP++ 3.5 Component Software Engineering User's Guide for 32-Bit Processors

Quick Installation Reference Card

For hardware information, refer to your processors's hardware reference, programming reference, or data sheet. All documentation is available online. Most documentation is available in printed form.

To access all processor and tools manuals and data sheets, visit the Technical Library Web site at:

<http://www.analog.com/processors/resources/technicalLibrary>

Online Technical Documentation

Online documentation comprises the VisualDSP++ Help system, software tools manuals, hardware tools manuals, processor manuals, Dinkum Abridged C++ library and Flexible License Manager (FlexLM) network license manager software documentation. You can easily search across the entire VisualDSP++ documentation set for any topic of interest. For easy printing, supplementary .PDF files of most manuals are also provided.

Each documentation file type is described in the following table.

File	Description
.CHM	Help system files and manuals in Help format
.HTM or .HTML	Dinkum Abridged C++ library and FlexLM network license manager software documentation. Viewing and printing the .HTML files require a browser, such as Internet Explorer 4.0 (or higher).
.PDF	VisualDSP++ tools manuals in Portable Documentation Format; one .PDF file for each manual. Viewing and printing the .PDF files require a PDF reader, such as Adobe Acrobat Reader (4.0 or higher).

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by running the Tools installation. Access the online documentation from the VisualDSP++ environment, Windows[®] Explorer, or the Analog Devices Web site.

From VisualDSP++

From VisualDSP++ environment:

- Access VisualDSP++ online Help from the Help menu's **Contents**, **Search**, and **Index** commands.
- Open online Help from context-sensitive user interface items (toolbar buttons, menu commands, and windows).

From Windows

In addition to any shortcuts you may have constructed, there are many ways to open VisualDSP++ online Help or the supplementary documentation from Windows.

Help system files (.CHM) are located in the `Help` folder, and .PDF files are located in the `Docs` folder of your VisualDSP++ installation CD-ROM. The `Docs` folder also contains the Dinkum Abridged C++ library and FlexLM network license manager software documentation.

Using Windows Explorer

- Double-click the `vdsp-help.chm` file, which is the master Help system, to access all the other .CHM files.
- Double-click any file that is part of the VisualDSP++ documentation set.

Using the Windows Start Button

- Access VisualDSP++ online Help by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, and **VisualDSP++ Documentation**.
- Access the .PDF files by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, **Documentation for Printing**, and the name of the book.

From the Web

Download manuals at the following Web site:

<http://www.analog.com/processors/resources/technicalLibrary/manuals>

Select a processor family and book title. Download archive (.ZIP) files, one for each manual. Use any archive management software, such as WinZip, to decompress downloaded files.

Printed Manuals

For general questions regarding literature ordering, call the Literature Center at 1-800-ANALOGD (1-800-262-5643) and follow the prompts.

VisualDSP++ Documentation Set

To purchase VisualDSP++ manuals, call 1-603-883-2430. The manuals may be purchased only as a kit.

If you do not have an account with Analog Devices, you are referred to Analog Devices distributors. For information on our distributors, log onto <http://www.analog.com/salesdir/continent.asp>.

Hardware Tools Manuals

To purchase EZ-KIT Lite™ and In-Circuit Emulator (ICE) manuals, call 1-603-883-2430. The manuals may be ordered by title or by product number located on the back cover of each manual.

Processor Manuals

Hardware reference and instruction set reference manuals may be ordered through the Literature Center at 1-800-ANALOGD (1-800-262-5643), or downloaded from the Analog Devices Web site. Manuals may be ordered by title or by product number located on the back cover of each manual.

Notation Conventions

Data Sheets

All data sheets (preliminary and production) may be downloaded from the Analog Devices Web site. Final data sheets (Rev. 0, A, B, C and so on) can be obtained from the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**) or downloaded from the Web site.

To have a data sheet faxed to you, call **1-800-446-6212**. Follow the prompts and a list of data sheet code numbers will be faxed to you. Call the Literature Center first to find out if requested data sheets are available.

Notation Conventions

The following table identifies and describes text conventions used in this manual.

 Additional conventions, which apply only to specific chapters, may appear throughout this document. Code has been formatted to fit this manual's page width.

Example	Description
Close command (File menu)	Titles in reference sections indicate the location of an item within the VisualDSP++ environment's menu system (for example, the Close command appears on the File menu).
{this that}	Alternative items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as <i>this</i> or <i>that</i> . One or the other is required.
[this that]	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional <i>this</i> or <i>that</i> .
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of <i>this</i> .
.SECTION	Commands, directives, keywords, and feature names are in text with letter gothic font.

Example	Description
	This symbol indicates a note that provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.
	This symbol indicates a warning that advises on an inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word Warning appears instead of this symbol.

Notation Conventions

1 INTRODUCTION TO VISUALDSP++

This manual describes VisualDSP++, a flexible management system that provides a suite of tools for developing DSP applications and projects.

VisualDSP++ includes:

- Integrated Development and Debugging Environment (IDDE) with VisualDSP++ Kernel (VDK) integration
- C/C++ optimizing compiler with run-time library
- Assembler and linker
- Simulator software and example programs

This chapter contains the following topics.

- [“VisualDSP++ Features” on page 1-2](#)
- [“License Management” on page 1-11](#)
- [“Project Development” on page 1-17](#)
- [“Code Development Tools” on page 1-30](#)
- [“VCSE” on page 1-47](#)
- [“DSP Projects” on page 1-57](#)
- [“VisualDSP++ Help System” on page 1-73](#)

VisualDSP++ Features

VisualDSP++ includes all the tools needed to build and manage DSP projects.

Integrated Development and Debugging

The VisualDSP++ IDDE provides complete graphical control of the edit, build, and debug process. In this integrated environment, you can move easily between editing, building, and debugging activities.

Code Development Tools

Depending on the DSP development tools purchased, VisualDSP++ includes one or more of the following components.

- C/C++ compiler with run-time library
- VisualDSP++ Component Software Engineering (VCSE) Interface Definition Language (VIDL) compiler
- Assembler, linker, preprocessor, and archiver
- Loader and splitter
- Simulator
- EZ-KIT Lite™ development system (must be purchased separately)
- Emulator (must be purchased separately)

VisualDSP++ supports ELF/DWARF-2 (Executable Linkable Format) executable files. VisualDSP++ supports all executable file formats produced by the linker.



If your system is configured with third-party development tools, you can select the compiler, assembler, linker, or loader to use for a particular target build.

Source File Editing Features

VisualDSP++ simplifies tasks involving source files. All the activities necessary to create, view, print, move within, and locate information are easy to perform.

- **Edit text files.** Create and modify source files and view listing or map files generated by the DSP code development tools.

Source files are the C/C++ language or assembly language files that make up your project.

DSP projects can include additional files such as data files and a Linker Description File (LDF), which contains command input for the linker. For more information about .LDF files, see [“Linker” on page 1-34](#).

- **Editor windows.** Open multiple editor windows to view and edit related files, or open multiple editor windows for a single file. The VisualDSP++ editor is an integrated code-writing tool that enables you to focus on code development.
- **Specify syntax coloring.** Configure options that specify the color of text objects viewed in an editor window.

This feature enhances the view and helps locate portions of the text, because keywords, quotes, and comments appear in distinct colors.

- **Context-sensitive expression evaluation.** Move the mouse pointer over a variable that is in the scope and view the variable’s value.

VisualDSP++ Features

- **Status icons.** View icons that indicate breakpoints, bookmarks, and the current PC position.
- **View error details and offending code.** From the **Output** window's **Build** view, display error details by highlighting the error code (such as cc0251) and pressing the F1 key. Double-click an error line to jump to the offending code in an editor window.

Project Management Features

VisualDSP++ provides flexible project management for the development of DSP applications, including access to all the activities necessary to create, define, and build DSP projects.

- **Define and manage projects.** Identify files that the code development tools process to build your project. Create this project definition once, or modify it to meet changing development needs.
- **Access and manage code development tools.** Configure options to specify how the DSP code development tools process inputs and generate outputs. Tool settings correspond to command-line switches for code development tools. Define these options once, or modify them to meet your needs.
- **View and respond to project build results.** View project status while a build progresses and, if necessary, halt the build.

Double-click on an error message in the **Output** window to view the source file causing the error, or iterate through error messages.

- **Manage source files.** Manage source files and track file dependencies in your project from the **Project** window to provide a display of software file relationships. VisualDSP++ uses code development tools to process your project and to produce a DSP program. It also provides a source code control (SCC) interface, which enables you to access SCC applications without leaving the IDDE.

Debugging Features

While debugging your project, you can:

- **View and debug mixed C/C++ and assembly code.** View C/C++ source code interspersed with assembly code. Line number and symbol information help you to source-level debug assembly files.
- **Run command-line scripts.** Use scripts to customize key debugging features.
- **Use memory expressions.** Use expressions that refer to memory.
- **Use breakpoints to view registers and memory.** Quickly add and remove, and enable and disable breakpoints.
- **Set simulated watchpoints.** Set watchpoints on stacks, registers, memory, or symbols to halt program execution.
- **Statistically profile the target processor's PC** (JTAG emulator debug targets only). Take random samples and display them graphically to see where the program uses most of its time.
- **Linearly profile the target processor's PC** (Simulation only). Sample every executed PC and provide an accurate and complete graphical display of what was executed in your program.
- **Generate interrupts using streaming I/O.** Set up serial port (SPORT) or memory-mapped I/O.
- **Create customized register windows.** Configure a custom register window to display a specified set of registers.
- **Plot values from processor memory.** Choose from multiple plot styles, data processing options, and presentation options.

VisualDSP++ Features

- **Trace program execution history.** Trace how your program arrives at a certain point and show reads, writes, and symbolic names.
- **View pipeline depth of assembly instructions.** Display the pipeline stage by querying the target processor(s) through the pipeline interface.

For details, refer to the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors* and the VisualDSP++ online Help.

VDK Features

The VisualDSP++ Kernel (VDK) is a scalable software executive specially developed for effective operations on Analog Devices 32-bit processors. The VDK is tightly integrated with VisualDSP++.

The kernel enables you to abstract the details of the hardware implementation from the software design. As a result, you can concentrate on the processing algorithms.

The kernel provides all the basic building blocks required for application development. Properties of the kernel can be characterized as follows.

- **Automatic.** VisualDSP++ automatically generates source code framework for each user-requested object in the user-specified language.
- **Deterministic.** VisualDSP++ specifies whether the execution time of a VDK API is deterministic.
- **Multitasking.** Kernel tasks (threads) are independent of one another. Each thread has its own stack.
- **Modular.** The kernel comprises various components. Future releases may offer additional functionality.

- **Portable.** Most of the kernel components can be written in ANSI Standard C or C++ and are portable to other Analog Devices processors.
- **Pre-emptive.** The kernel's priority-based scheduler enables the highest-priority thread not waiting for a signal to be run at any time.
- **Prototypical.** The kernel and VisualDSP++ create an initial file set based on a series of template files. The entire application is prototyped and ready to be tested.
- **Reliable.** The kernel provides run-time error checking.
- **Scalable.** If a project does not include a kernel feature, the support code is not included in the target system.

VisualDSP++ 3.5 Features

VisualDSP++ 3.5 includes the following new features and enhancements.

- **New processor support.** These new SHARC processors are supported: ADSP-21261, ADSP-21262, ADSP-21266, ADSP-21267, ADSP-21363, ADSP-21364, and ADSP-21365. These new TigerSHARC processors are supported: ADSP-TS202 and ADSP-TS203. Refer to the processor's Hardware Reference and chip Data Sheet for details.
- **Multiple project support.** VisualDSP++ provides the ability to switch among multiple open projects in the same IDDE session. The **Project** window displays active projects.
- **License management in the IDDE.** License management (installation and validation) has been integrated into the VisualDSP++ IDDE. Installing a Flexible License Manager (FlexLM) license server is still handled by the separate installation application.

- **Data streaming and logging.** VisualDSP++ now offers the ability to stream and log data from a target processor without halting the processor. The IDDE takes advantage of this capability in plot windows. If the target supports background telemetry channel (BTC), the plot window is updated while the target is running. See the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors* for information about using BTC.
- **Profile-guided optimization (PGO) in the IDDE.** VisualDSP++ includes facilities to run common PGO scenarios simply and provides a mechanism for advanced applications that require more control over the profiling process via scripting. The PGO process involves setting up and executing data sets to produce an optimized application. A *data set* is the association of zero or more input streams with one .PGO output file.

The most common scenario for collecting PGO data is setting up one or more simple *File to Device* streams. The *File* is a standard ASCII stream input file, and the *Device* is any stream device (such as memory or a peripheral) supported by the simulator target. You create, edit, and delete data sets in VisualDSP++ and then run them to optimize your application. For PGO operations, the **Tools** menu provides a new **PGO** submenu and a **Manage Data Sets** option.

Note that each compiler manual associated with VisualDSP++ 3.5 has a new chapter called “Achieving Optimal Performance from C/C++ Source Code.” Its focus is to help maximize code performance from the compiler while minimizing code size. The chapter starts by discussing some general optimization principles and how the compiler can most help your optimization effort. Optimal coding styles are then considered in detail. Special features such as compiler switches, built-in functions, and pragmas are also discussed. The chapter ends with a short example that demonstrates how the optimizer works.

Additional information about PGO is in the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors* and in the online Help.

- **Integrated Source Code Control (SCC).** VisualDSP++ uses the Microsoft Common Source Code Control (MCSCC) interface to provide a connection from the IDDE to SCC applications (such as Visual SourceSafe, PVCS Version Manager, and ClearCase) installed on your machine. You can now conveniently access commonly-used SCC features from VisualDSP++ without leaving the IDDE. Advanced and application-specific SCC features not available from the IDDE must be run directly from the SCC applications.
- **Automation aware scripting engine.** VisualDSP++ includes a scripting engine that uses the Microsoft ActiveX script host framework. The engine enables you to use multiple scripting languages (such as VBScript, JavaScript, and so on) to access the VisualDSP++ Automation API.

Interact with the IDDE by using a single command or a script file similar to the Tcl scripting functionality, which was available in previous versions of VisualDSP++.

- **Profiling code with Expert Linker.** Use Expert Linker to profile object sections in a program. When the program halts, Expert Linker graphically displays how much time was spent in each object section. Use this display to locate code “hotspots” and then move that code to faster, internal memory.
- **Address bar in Disassembly and Memory windows.** When enabled, an address bar is displayed in **Disassembly** windows and memory windows. Use the address bar to navigate by address, symbol, or expression. The address bar maintains a most recently used history of visited locations.
- **Menus with Icons.** Icons now appear beside menu commands that have corresponding toolbar buttons.

- **Program flow manipulation for the TigerSHARC cycle-accurate simulator.** You can manually change the program counter value in the **PC Register** window to specify which instruction the simulator executes next. When you change a PC value, the simulator automatically flushes the pipeline and aborts all its instructions. After the pipeline is flushed, normal execution resumes from a memory address indicated by the new PC value.
- **New data format support in Memory and Register windows.** Data in a **Register** or **Memory** window can be displayed in 8-bit or 32-bit character format. When compiled and loaded into memory, a `code char string[]` can be displayed in a **Memory** window as defined. Select the memory address of the symbol string and choose the display format for which the code was compiled.
- **New Select Loader Program command added to the Settings→Simulator menu for TigerSHARC processors.** You can specify a custom loader program that will run automatically every time before a user program is loaded. When you start VisualDSP++, the simulator defaults to a standard loader program:
`VisualDSP\Ts\ldr\TS201_prom.dxe.`
- **Splitter support for TigerSHARC processors.** VisualDSP++ 3.5 includes a TigerSHARC splitter.

License Management

VisualDSP++ is a licensed software product. This section describes licensing options, license status, license installation, software registration, validation codes, product updates, and product serial numbers.

Licensing Options

Two licensing options are available: *single-user* and *client*. A *server* license is required before you can install a client license (see [Table 1-1](#)).

Table 1-1. VisualDSP++ Licenses

License	Description
Single-User	Also called <i>node-locked</i> or <i>per-user</i> licenses, single-user licenses are locked to the machine ID of the host computer. Once installed, tools run only on that one machine. You may install and register the software up to three times (for example, at work, at home, and on a laptop computer). Use, however, is restricted to one installed software at a time.
Client	Client licenses are a client/server-based application. The server manages a pool of licenses installed on the server. One license is installed on the server for each purchased copy of VisualDSP++. In this model, you can have as many client installations as desired. When a client starts the software, it checks out a license from the server. When the software exits, the license is returned to the server. As long as licenses are available on the server, clients can access VisualDSP++. Example: Assume a license server has been set up with 10 licenses, and 20 client machines are installed in three labs. Ten simultaneous developers (any combination) can use the software. When the 11th client tries to use VisualDSP++, a message appears, stating that no more licenses are available. This allows sharing of the software resources in an environment that needs more locations than developers.
Server	Allows multiple users to access VisualDSP++ on computers sharing client licenses across a network. A server license must be installed prior to installing client licenses.

License Management

Server-based *floating licenses* consist of two parts: server and client. The server manages the license pool that is stored on the server. The clients “check out” licenses when the software is started and return licenses to the server when developers exit VisualDSP++.

License Status

The **Licenses** page of the **About VisualDSP++** dialog box displays the status of all recognized licenses.

The status for each serial numbered product can be:

- Validated (Permanent)
- Not Validated
- Test Drive
- For EZ-KIT Lite: Permanent, Restricted; Expiring in X days; or Expired

Temporary Licenses

A temporary license indicates the number of days remaining before the product can no longer be used. Test drive versions of VisualDSP++ (serial number beginning with “TST”) carry temporary licenses.

An unrestricted version of VisualDSP++ includes its permanent license. If you do not install the validation code after purchasing a full (unrestricted) license, the status of the license is marked “Not Validated (Expiring in X days).” Install the validation code to change the status to “Permanent.”

Valid Versus Expired Licenses

An expired license is indicated by “Expired”. A valid license is indicated by “Permanent” unless it is for an EZ-KIT Lite evaluation system. A valid license for an EZ-KIT Lite is indicated by “Permanent, Restricted”.

Client Licenses

When a client license is installed, the “server_name” appears under **Serial Number**, “client” appears under **Family**, and “use_server” appears under **Status**.

License Installation

After installing VisualDSP++, you have to license the software. Licensing involves these three tasks:

- Installing the single-user or client serial number

Note that a server license must be installed before client licenses can be installed.

- Registering products
- Entering validation codes

You perform license management activities within VisualDSP++ by using the **About VisualDSP++** dialog box. See the online Help for detailed installation, registration, and validation procedures.

Test drives require online registration to receive a “TST” serial number, which expires 90 days after installation. Test drives do not require a validation code.

Installing a License Shipped With EZ-KIT Lite



As of VisualDSP++ 3.5, EZ-KIT Lite versions of VisualDSP++ require online registration and a validation code. The “KIT” serial number is on the label on the back of the CD wallet.

License Management

“KIT” serial numbers impose these restrictions on VisualDSP++:

- The size of a user program is limited to 10K of the ADSP-21262 processor’s internal memory space.
- No connections to Simulator or Emulator sessions are allowed.
- Only one EZ-KIT Lite board can be connected to the host PC and debugged at a time.
- The EZ-KIT Lite hardware must be connected and powered up in order to use VisualDSP++ with a kit license.

These limitations do not prevent processor evaluation on the EZ-KIT Lite evaluation board, but they encourage the purchase of a full (unrestricted) VisualDSP++ license.

Installing a Single-User License

VisualDSP++ must be licensed. A single-user (per-user, node-locked) license allows VisualDSP++ to be used on one computer. Installing the serial number creates a `license.dat` file.

 This procedure requires a serial number, which is located on the software registration card or on the CD’s sleeve.

This procedure assumes you have installed VisualDSP++ from the CD and have installed the latest service pack (if applicable). Service packs are available from the Analog Devices Web site at:

<http://www.analog.com/dsp/tools/fixes.html>

If the installation is successful, a “success” message appears. Additional information appears, depending on the installed license. At this point, the software has a temporary, 30-day license.

For serial numbers that begin with “ADI” or “KIT”, a message instructs you to register the serial number to receive a validation code.

For serial numbers that begin with “TST”, a message informs you that the license can be installed only once.

You install a single-user license from the **Install New License** dialog box, accessed from the **Licenses** page of the **About VisualDSP++** dialog box.

Installing a Server License

Installing a server license allows multiple users to use VisualDSP++ on computers using “client licenses” across a network. A server license must be installed before client licenses can be installed.

 This procedure requires the entry of a server node name, which is obtained from the **Identification** page of the **Network** applet in Windows **Control Panel**.

The installation procedure uses the FlexLM network license manager included with VisualDSP++. FlexLM installs the NT service and path information. Note that when a seat or a permanent license is added, the FlexLM license manager must be restarted/reread each time the license file is updated.

You install a server license by running **Install License** and **Install Server License** from the VisualDSP++ CD ROM.

Installing a Client License

A client license allows access to VisualDSP++ over a network.

 A server license allows multiple users to use VisualDSP++ on computers using “client licenses” across a network. A server license must be installed before client licenses can be installed.

License Management

If the installation is successful, a “success” message is displayed.

 If a `license.dat` file already exists on the client machine (`VisualDSP\system`), a message indicates that the current `license.dat` file will be renamed to `license.bak` and the server license will be copied to the client machine.

You install a client license from the **Install New License** dialog box, accessed from the **Licenses** page of the **About VisualDSP++** dialog box.

Software Registration

After installing or floating a single-user license, register the product (via the Analog Devices Web site) within 30 days to obtain a validation code.

 “ADI” and “KIT” serial numbers require registration. “TST” serial numbers do not require registration.

You register a license from the **Licenses** page of the **About VisualDSP++** dialog box, accessed from the **Help** menu.

Validation Codes

After successfully registering an “ADI” or “KIT” serial number on the Analog Devices Web site, you receive a validation code that must be entered to create a permanent license. If this code is entered successfully, a message indicates that a permanent license has been created.

You enter a validation code from the **Enter Validation Code** dialog box, accessed from the **Licenses** page of the **About VisualDSP++** dialog box.

Product Upgrades

From time to time, Analog Devices releases new software versions.

The upgrade procedure does not change the previous version's folder structure or license file. The new installation process uses the previous version's path and license.



Check the Analog Devices Web site to ensure you have the latest software version.

Product Serial Numbers

Product serial numbers are located on product CD sleeves. A product's serial number can also be viewed from within VisualDSP++.

If you cannot locate a serial number, contact your local sales representative or Analog Devices sales.

- Send e-mail to: dsptools.support@analog.com
- Phone: 1-800-ANALOGD (1-800-262-5643)

Provide details about the exact products, versions, and operating system being used.

Within VisualDSP++, you view product serial numbers from the **Licenses** page of the **About VisualDSP++** dialog box, accessed from the **Help** menu.

Project Development

During project development, VisualDSP++ helps you interactively observe and alter the data in the processor and in memory.

Overview of Programming With VisualDSP++

Programming effectively with VisualDSP++ depends on how well you master a four-step process. You must learn how to:

1. Work with VisualDSP++.

Project Development

2. Implement structured software design with VisualDSP++.
3. Optimize performance with VisualDSP++.
4. Test and debug your programs with VisualDSP++.

Working With VisualDSP++: You should have a working knowledge of VisualDSP++, the front end for all available targets and platforms. You should know how and when to use its various features and have a firm foundation in these project basics:

- Working with property pages. These pages are analogous to command-line switches.
- Setting up debug sessions. Know the distinctions between the three development stages: evaluation (via an EZ-KIT Lite development system), simulation, and emulation.
- Understanding how program sections and memory segments relate to physical processor memory. Become familiar with Expert Linker.
- Accessing peripherals. This task includes setting up and handling interrupts in both C and assembly.

Designing Structured Software With VisualDSP++: Consider elements of software design, code reuse, and interoperability. If you are new to embedded systems, try to acquire a clear understanding of:

- The role of and motivation behind component software
- How to create and use a VisualDSP++ Component Software Engineering (VCSE) component
- The role of an RTOS
- How to use VDK to manage multiple threads of execution and the communication between those threads

Optimizing Performance With VisualDSP++: At this stage, you should understand how to access the features of the processor and how to use a structured approach to develop software. Next optimize your software to take full advantage of the processor's computational power. This step entails:

- Understanding the compiler optimizer
- Writing mixed C and assembly programs
- Accessing C/C++ data structures in assembly
- Harnessing the power of C++
- Setting up and using overlays
- Configuring emulation L1 memory for cache versus SRAM with cache visualization
- Using statistical profiling

Testing and Debugging With VisualDSP++: At this stage, you should have a good understanding of the various facilities available for producing optimal software. The last step is applying software testing and debugging techniques, which include:

- Collecting data and using the advanced plot windows
- Using compiled simulation
- Using ActiveX and COM Automation to create regression test environments and to take advantage of interoperability with other applications

DSP Project Development Stages

The typical project includes three phases: *simulation*, *evaluation*, and *emulation*. These phases are shown in [Figure 1-1](#).

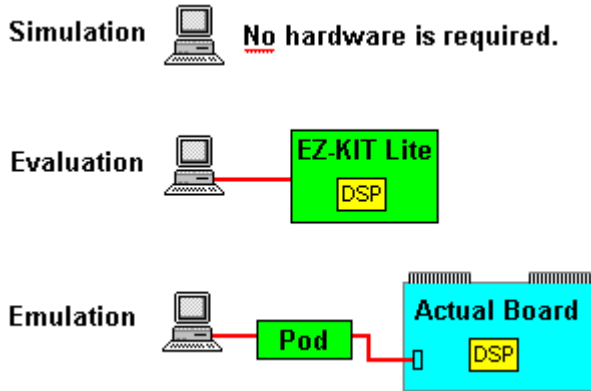


Figure 1-1. Project Development Stages

You use VisualDSP++ during both simulation and emulation.

Simulation

Project development typically begins in a simulation environment while hardware engineers are developing the new hardware (cell phone, computer, and so on). Simulation mimics system memory and I/O, which enables portions of the target system hardware to be viewed. A *simulator* is software that mimics the behavior of a DSP chip. Running VisualDSP++ with a simulation target without a physical processor enables you to build, edit, and debug your DSP program before a DSP chip is manufactured.

Evaluation

Use an EZ-KIT Lite evaluation system in your project's early planning stage to determine the processor that best fits your needs. Your PC connects to the EZ-KIT Lite board via cable, which enables you to monitor processor behavior.

Emulation

Once the hardware is ready, move directly to a JTAG *emulator*, the hardware that connects your PC to the actual processor target board. An emulator enables application software to be downloaded and debugged from within VisualDSP++. Emulator software performs the communications that enable you to see how your DSP code affects DSP performance.

Targets

A *target* (or debug target) refers to the communication channel between VisualDSP++ and a processor (or group of processors). A target can be a simulator, EZ-KIT Lite evaluation board, or an emulator. Your system can include multiple targets.

For example, the JTAG emulator communicates with one or more physical devices over the host PC's PCI bus, and the Apex-ICE™ emulator communicates with a device via the PC's USB port.

Simulation Targets

A *simulation target*, such as the ADSP-2106x Family Simulator, is a pure software module and does **not** require the presence of a processor for debugging.

During simulation, VisualDSP++ reads an executable (.EXE) file and executes it in software, similar to the way a processor executes a processor image in hardware. VisualDSP++ simulates the memory and I/O devices specified in an .LDF file.

Project Development

EZ-KIT Lite Targets

An *EZ-KIT Lite target* is a development board used to evaluate a particular processor. Analog Devices provides several EZ-KIT Lite evaluation systems, which include demonstration programs.

Emulation Targets

An *emulation target* is a module that controls a physical processor connected to a JTAG emulator system. For example, the Summit-ICE™ emulator communicates with one or more physical devices through the host PC's PCI bus, and the Apex-ICE emulator communicates with a device through the PC's USB port.

Platforms

A *platform* refers to the configuration of processors with which a target communicates. Several platforms may exist for a given debug target. For example, if three emulators are installed on your system, you might select emulator 2 as the platform that you want to use. The platform that you use depends on your project development stage. (See [Table 1-2.](#))

Table 1-2. Development Stages and Supported Platforms

Stage	Platform
Simulation	Typically one or more processors of the same type. By default, the platform name is the identical DSP simulator.
Evaluation	An EZ-KIT Lite evaluation system
Emulation	Any combination of devices. You must configure the platform for a particular target with the JTAG ICE Configurator. When the debug target is a JTAG emulator, a platform refers to a JTAG chain.

Hardware Simulation

When connected to a simulation target in VisualDSP++, you can simulate the following hardware conditions.

- Random interrupts that can occur during program execution
- Data transfer through the processor's I/O pins
- Processor booting from a PROM or host processor

Setting up VisualDSP++ to generate random interrupts during program execution enables you to exercise interrupt service routines (ISR) in your code.

Debugging Overview

Once you successfully build a DSP project and generate a DSP executable file, you can debug the project. Projects developed in VisualDSP++ are run as hardware and software *debug sessions*.

In [Table 1-3](#), the check mark (✓) indicates which debugging tools are available during the process of building and debugging a DSP program.

Table 1-3. Tools Available During Simulation, Evaluation, and Emulation

Tool	Simulation	Evaluation	Emulation
Linear profiles	✓		
Interrupts	✓		
Streams	✓		
Traces (SHARC processors only)	✓		

Table 1-3. Tools Available During Simulation, Evaluation, and Emulation

Tool	Simulation	Evaluation	Emulation
Pipeline Viewer (TigerSHARC processors only)	✓		
Breakpoints	✓	✓	✓
Watchpoints	✓		
Hardware breakpoints			✓
Plotting	✓	✓	✓
Statistical profiles			✓

You can attach to and control the operation of any Analog Devices processors or DSP simulator. Download your application code to the processor and use VisualDSP++'s debugging facilities to ensure that your application functions as desired.

VisualDSP++ is a window into the inner workings of the target processor or simulator. From this user interface, you can:

- Run, step, and halt the program and set breakpoints and watchpoints
- View the state of the processor's memory, registers, and stacks
- Perform a cycle-accurate statistical profile or linear profile
- Perform integrated multiprocessor debugging (emulator sessions only)

VisualDSP++ Kernel

A 32-bit project can optionally include the VisualDSP++ Kernel (VDK), which is a software executive between DSP algorithms, peripherals, and control logic.

The **Project** window's **Kernel** tab accesses a tree control for structuring and scaling application development. From this tree control, you can add, modify, and delete Kernel elements such as thread types, boot threads, round-robin priorities, semaphores, events, event bits, interrupts, and device drivers.

Two VDK-specific windows, **VDK State History** and **Target Load**, provide views of VDK information. Another VDK window, **VDK Status**, provides thread status data when a VDK-enabled program is halted. Refer to the *VisualDSP++ 3.5 Kernel (VDK) User's Guide for 32-Bit Processors* for complete details.

Program Development Steps

In the VisualDSP++ environment, program development consists of the following steps.

1. Create a project.
2. Configure project options.
3. Add and edit project source files.
4. Define project build options.
5. Build a debug version (executable file) of the project.
6. Create a debug session and load the executable.
7. Run and debug the program.
8. Build a release version of the project.

Project Development

By following these steps, you can build DSP projects consistently and accurately with minimal project management. This process reduces development time and lets you concentrate on code development.

These steps, described below, are covered in detail in the online Help and in the “Basic Tutorial” chapter of the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors*.

Step 1: Create a Project

All development in VisualDSP++ occurs within a project. The project (.DPJ) file stores your program’s build information: source files list and development tools option settings.

Step 2: Configure Project Options

Define the target processor and set up your project options (or accept default settings) before adding files to the project. The **Project Options** dialog box provides access to project options, which enable the corresponding build tools to process the project’s files correctly.

Step 3: Add and Edit Project Source Files

A project normally contains one or more C, C++, or assembly language source files. After creating a project and defining its target processor, add new or existing files to the project by importing or writing them. Use the VisualDSP++ Editor to create new files or edit any existing text files.

Adding Files to Your Project

You can add any type of file to the project. The DSP development tools selectively process only recognized file types when you build the project.

Creating Files to Add to Your Project

You can create new text files. The editor can read or write text files with arbitrary names. Adding files to your project updates the project's file tree in the **Project** window.

Editing Files

You can edit the file(s) that you add to the project. To open a file for editing, double-click on the file icon in the **Project** window.

The editor has a standard Windows-style user interface and supports normal editing operations and multiple open windows. You can customize language- and processor-specific syntax coloring, and create and search for bookmarks.

Managing Project Dependencies

Project dependencies control how source files use information in other files, and consequently determine the build order. VisualDSP++ maintains a makefile, which stores dependency information for each file in the project. VisualDSP++ updates dependency information when you change the project's build options, add a file to the project, or choose **Update Dependencies** from the **Project** menu.

Step 4: Define Project Build Options

After creating a project, setting the target processor, and adding or editing the project's source files, configure your project's build options. Specify options or accept the default options in VisualDSP++ before using the development tools that create your executable file. You can specify options for a whole project or for individual files, or you can specify a custom build.



VisualDSP++ retains your changes to the build options. Settings reflect your last changes, not necessarily the original defaults.

Project Development

Configuration

A project's configuration setting controls its build. By default, the choices are **Debug** or **Release**.

- Selecting **Debug** and leaving all other options at their default settings builds a project that can be debugged. The compiler generates debug information.
- Selecting **Release** and leaving all other options at their default settings builds a project with limited or no debug capabilities. Release builds are usually optimized for performance. Your test suite should verify that the Release build operates correctly without introducing significant bugs.

You can modify VisualDSP++'s default operation for either configuration by changing the appropriate entries on the **Compile**, **Assemble**, and **Link** property pages. You can create custom configurations that include the build options and source files that you want.

Project-Wide File and Tool Options

Next, you must decide whether to use project-wide option settings or individual file settings.

For projects built entirely within VisualDSP++ with no pre-existing object or archive (library) files, you typically use project-wide options. New files added to the project inherit these settings.

Individual File and Tool Options

Occasionally, you may want to specify tool settings for individual files. Each file is associated with two property pages: a **General** page, which lets you choose output directories for intermediate and output files, and a tool-specific property page (**Compile**, **Assemble**, **Link**, and so on), which lets you choose options. For information about each tool's options, see the online Help or the manual for each tool.

Step 5: Build a Debug Version of the Project

Next, build a debug version of the project.

Status messages from each code development tool appear in the **Output** window as the build progresses.



The output file type *must* be an executable (.EXE) file to produce debugger-compatible output.

Step 6: Create a Debug Session and Load the Executable

After successfully building an executable file, set up a debug *session*. You run DSP projects that you develop as either hardware or software sessions. After specifying target and processor information, load your project's executable file. On the **General** page in the **Preferences** dialog box, you can configure VisualDSP++ to load the file automatically and advance to the `main` function of your code.

Step 7: Run and Debug the Program

After successfully creating a debug session and building and loading your executable program, run and debug the program.

If the project is not current (has outdated source files or dependency information), VisualDSP++ prompts you to build the project before loading and debugging the executable file.

Step 8: Build a Release Version of the Project

After you finish debugging your application, build a Release version of your project to run on the product's processor.

Code Development Tools

This section describes the following DSP development tools.

- C/C++ compiler with run-time libraries
- VIDL compiler
- Assembler
- Linker
- Expert Linker
- Preprocessor
- Archiver
- Splitter
- Loader

Available code development tools differ, depending on the processor. The options available on the tabbed pages of the **Project Options** dialog box enable you to specify tool preference.

VisualDSP++ supports ELF/DWARF-2 (Executable Linkable Format, Debugging Information Format) executable files. VisualDSP++ supports all executable file formats produced by the linker.

If your system is configured with third-party development tools, you can select the compiler, assembler, or linker to use for a particular target build.

Compiler

The compiler processes C/C++ programs into assembly code. The term *compiler* refers to the compiler utility shipped with the VisualDSP++ software.

The compiler generates a linkable object file by compiling one or more C/C++ source files. The compiler's primary output is a linkable object file with a `.OBJ` extension.

To specify compiler options for your build, select **Project**, **Project Options**, and the **Compile** tab. Then on the **Compile** page, select a **Category** of options.

Compiler options are grouped into the categories described in [Table 1-4](#).

Table 1-4. Groups of Compiler Options

Category	Purpose
General	Optimization, compilation, and termination options
Preprocessor	Macro and directory search options
Processor	Processor-specific options
Warning	Warning and error reporting options



The available compile options depend on your target processor and your code development tools.

For more information, refer to the VisualDSP++ 3.5 C/C++ Compiler and Library Manual for your processor.

C++ Run-Time Libraries



You must be running VisualDSP++ to use the C++ run-time libraries.

The C and C++ run-time libraries are collections of functions, macros, and class templates that can be called from source programs. Many functions are implemented in the processor assembly language.

C and C++ programs depend on library functions to perform operations that are basic to the C and C++ programming languages. These operations include memory allocations, character and string conversions, and math calculations. The libraries also include multiple signal processing functions that ease DSP code development. The run-time library simplifies software development by providing code for a variety of common needs.

The compiler provides a broad collection of C functions including those required by the ANSI standard and additional Analog Devices-supplied functions of value for DSP programming. This release of the compiler software includes both the Standard C Library and the Abridged Library, a conforming subset of the Standard C++ Library.

For more information about the algorithms on which many of the C library's math functions are based, refer to the Cody and Waite text *Software Manual for the Elementary Functions* from Prentice Hall (1980).

For more information about the C++ library portion of the ANSI/ ISO Standard for C++, refer to the Plauger text *Draft Standard C++ Library* from Prentice Hall (1994) (ISBN: 0131170031).

Assembler

The assembler generates an object file by assembling source, header, and data files. The assembler's primary output is an object file with a `.DOJ` extension.

To specify assembler options, select **Project**, **Project Options**, and the **Assemble** tab.

Assembler terms are defined as follows.

Instruction set

Set of assembly instructions that pertain to a specific processor. For information about the instruction set, refer to your processor's Hardware Reference.

Preprocessor commands

Commands that direct the preprocessor to include files, perform macro substitutions, and control conditional assembly

Assembler directives

Directives that tell the assembler how to process source code and set up DSP features. Use directives to structure your program into logical *segments* or sections that support the use of a Linker Description File (`.LDF` file) to construct an image suited to the target system.

For more information, refer to the VisualDSP++ 3.5 Assembler and Preprocessor Manual for your processor.

Linker

The linker links separately assembled files (object files and library files) to produce executable (.DxE) files, shared memory (.SM) files, and overlay (.OVL) files, which can be loaded onto the target.

The linker's output files (.DxE, .SM, .OVL) are binary, executable, and linkable files (ELF). To make an executable file, the linker processes data from a Linker Description File (.LDF file) and one or more object (.DOJ) files. The executable files contain program code and debugging information. The linker fully resolves addresses in executable files.

To specify linker options, select **Project**, **Project Options**, and the **Link** tab. Then on the **Link** page, select a **Category** of options. Linker options are grouped into the following categories.

- General
- LDF Preprocessing
- Elimination
- Processor

Linker terms are defined as follows.

Link against

Functionality that enables the linker to resolve symbols to which multiple executables refer. For instance, shared memory (.SM) executable files contain sections of code that other processor executable (.DxE) files link against. Through this process, a shared item is available to multiple executable files without being duplicated.

Link objects

Object files (.OBJ) that become linked and other items, such as executable (.EXE, .SM, .OVL) files, that are linked against

LDF file

File that contains the commands, macros, and expressions that control how the linker arranges your program in memory

Memory

Definitions that provide the linker with a description of your target DSP system

Overlays

Files that your overlay manager swaps in and out of run-time memory, depending on code operations. The linker produces overlay (.OVL) files.

Sections

Declarations that identify the content for each executable file that the linker produces

For more information, refer to the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

A Linker Description File (.LDF file) describes the target system and maps your program code with the system memory and processors.

The .LDF file creates an executable file by using:

- The target system memory map
- Defined segments in your source files

The parts of an `.LDF` file, from the beginning to the end of the file, are described as follows.

- Memory map – describes the processor’s physical memory, at the beginning of the `.LDF` file
- `SEARCH_DIR`, `$LIBRARIES`, and `$OBJECTS` commands – define the path names that the linker uses to search and resolve references in the input files
- `MEMORY` command – defines the system’s physical memory and assigns labels to logical segments within it. These logical segments define program, memory, and stack memory types.
- `SECTIONS` command – defines the placement of code in physical memory by mapping the sections specified in program files to the sections declared in the `MEMORY` command. The `INPUT_SECTIONS` statement specifies the object file that the linker uses to resolve the mapping.

For details, refer to the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Expert Linker

Expert Linker is a graphical tool that enables you to:

- Define a target processor's memory map
- Place a project's object sections into that memory map
- View how much stack or heap has been used after you run a DSP program

This interactive tool speeds up the configuration of system memory. It uses your application's target memory description, object files, and libraries to create a memory map that you can manipulate to optimize your system's use of memory.



Expert Linker works with the linker. For more information about linking, refer to the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Expert Linker graphically displays the available project information in an .LDF file as input. This information includes object files, LDF macros, libraries, and target memory descriptions. Use the drag-and-drop function to arrange the object files in a graphical memory-mapping representation. When you are satisfied with the memory layout, generate the executable file (.EXE) via VisualDSP++ project options.



You can use default .LDF files that come with VisualDSP++ or the Expert Linker wizard to create and customize a new .LDF file.

When opened in a project that already includes an .LDF file, Expert Linker parses the .LDF file and graphically displays the target processor's memory map and the object mappings. The memory map appears in the **Expert Linker** window ([Figure 1-2 on page 1-39](#)). Use this display to modify the memory map or the object mappings. When the project is ready to be built, Expert Linker saves the changes to the .LDF file.

Expert Linker can graphically display space allocated to program heap and stack. After you load and run your program, Expert Linker indicates the used portion of the heap and stack. You can then reduce the size of the heap or stack to minimize the memory allocated for the heap and stack. Freeing up memory in this way enables you to use it for storing other things like DSP code or data.

You can launch the Expert Linker from VisualDSP++ in three ways:

- Double-click the `.LDF` file in the **Project** window.
- Right-click the `.LDF` file in the **Project** window to display a menu and then choose **Open in Expert Linker**.
- From the VisualDSP++ main menu, choose **Tools, Expert Linker**, and **Create LDF**.

The **Expert Linker** window ([Figure 1-2](#)) is displayed.

Expert Linker Window

The Expert Linker window (Figure 1-2) enables you to modify the memory map or the object mappings. You can specify a color for each type of object (internal memory, external memory, unused memory, reserved memory, output sections, object sections, overlays in live space, and overlays in run space). The objects are displayed in color when you view the **Memory Map** pane in graphical memory map mode. When the project is ready to be built, Expert Linker saves the changes to the .LDF file.

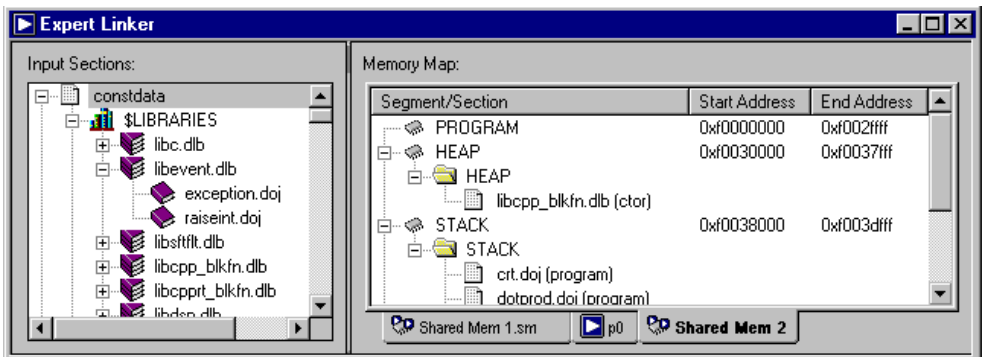


Figure 1-2. Expert Linker Window

The **Expert Linker** window contains two main panes:

- The **Input Sections** pane displays a tree structure of the input sections.
- The **Memory Map** pane displays each memory map in a tree or graphical representation.

You can dock or float the **Expert Linker** window in the VisualDSP++ main window.

Memory Map Pane Right-Click Menu

Table 1-5 describes the commands on the Memory Map right-click menu.

Table 1-5. Memory Map Pane Right-Click Menu

Command	Purpose
View Mode→Memory Map Tree	Displays the memory map in tree mode
View Mode→Graphical Memory Map	Displays the memory map in graphical blocks
View→Mapping Strategy (Pre-Link)	Displays the memory map, which shows where you intended to place object sections
View→Link Results (Post-Link)	Displays the memory map, which shows where the object sections are actually placed
New→Memory Segment	Opens the Memory Segment Properties dialog box, from which you specify the name, address range, type, width, and so on of the memory segment that you want to add
New→Output Section	Adds an output section to the selected memory segment Note: Right-click on a memory segment to access this command.
New→Shared Memory	Opens the Shared Memory Properties dialog box, from which you specify the name of the shared memory output file and processors that share the memory This command is not available on single-processor systems.
New→Overlay	Opens the Overlay Properties dialog box, from which you add a new overlay to the selected output section or memory segment Note: The new overlay's run space is in the selected output section.
Delete	Deletes the selected object
Pin-to-Output Section	Pins an object section to an output section to prevent it from overflowing to another output section This command is available only after you right-click on an object section that is part of an output section set to overflow to another section.

Table 1-5. Memory Map Pane Right-Click Menu (Cont'd)

Command	Purpose
View Section Contents	Opens the Section Contents dialog box, which displays the contents of the input or output section This command is available only after you link or build the project and then right-click on an input or object section.
Add Hardware Page Overlay Support	Sets up hardware overlay live and run spaces for all available hardware pages by: a) Checking if memory segments are currently defined in all hardware pages. If memory segments are located, you are queried about whether to delete those segments. b) Creating a memory segment containing an overlay (live space) in each hardware page c) Creating a memory segment containing all overlay run spaces in hardware page 0 d) Creating a default mapping for each overlay. The default mapping maps objects containing the section, “pmpage0” to the hardware overlay on PM page 0, “pmpage1” to PM page 1, “dmpage0” to DM page 0, and so on.
View Symbols	Opens the View Symbols dialog box and displays the symbols for the project, overlay, or input section This command is available after you link the project and then right-click on the Memory Map pane for a processor, memory segment, output section, or input section.
Expand All	Expands all items in the memory map tree to make their contents visible
View Legend	Opens the Legend dialog box, which shows all possible icons in the tree window, with a brief description of each icon. The Colors page displays a list of colors used in the graphical memory map. You can specify each object's color.
View Global Properties	Opens the Global Properties dialog box for the selected object. The dialog box's title and content depend on the selected object.

Stack and Heap Usage

Expert Linker enables you to adjust the size of the stack and heap, and make better use of memory.

Expert Linker can:

- Locate stacks and heaps and fill them with a marker value

This operation occurs after you load the program into a processor target. The stacks and heaps are located by their memory segment names, which may vary across processor families.

- Search the heap and stack for the highest memory locations written to by the DSP program

This operation occurs when the target halts after you run the program. Assume that these values are the start of the unused portion of the stack or heap. Expert Linker updates the memory map to show how much of the stack and heap are unused.

Be aware of the following stack and heap restrictions.

- The heap, stack, and system stack must be defined in output sections named `HEAP`, `STACK`, and `SYSSTACK`, respectively.
- The heap, stack, and system stack must be the only items in those output sections. You cannot place other objects in those output sections.

For other processor families, the restrictions on memory segment names differ according to what is used in the default `.LDF` files. If you do not heed these restrictions, you cannot view stack and heap usage after running your program.

Figure 1-3 shows an example memory map after you run a SHARC program.

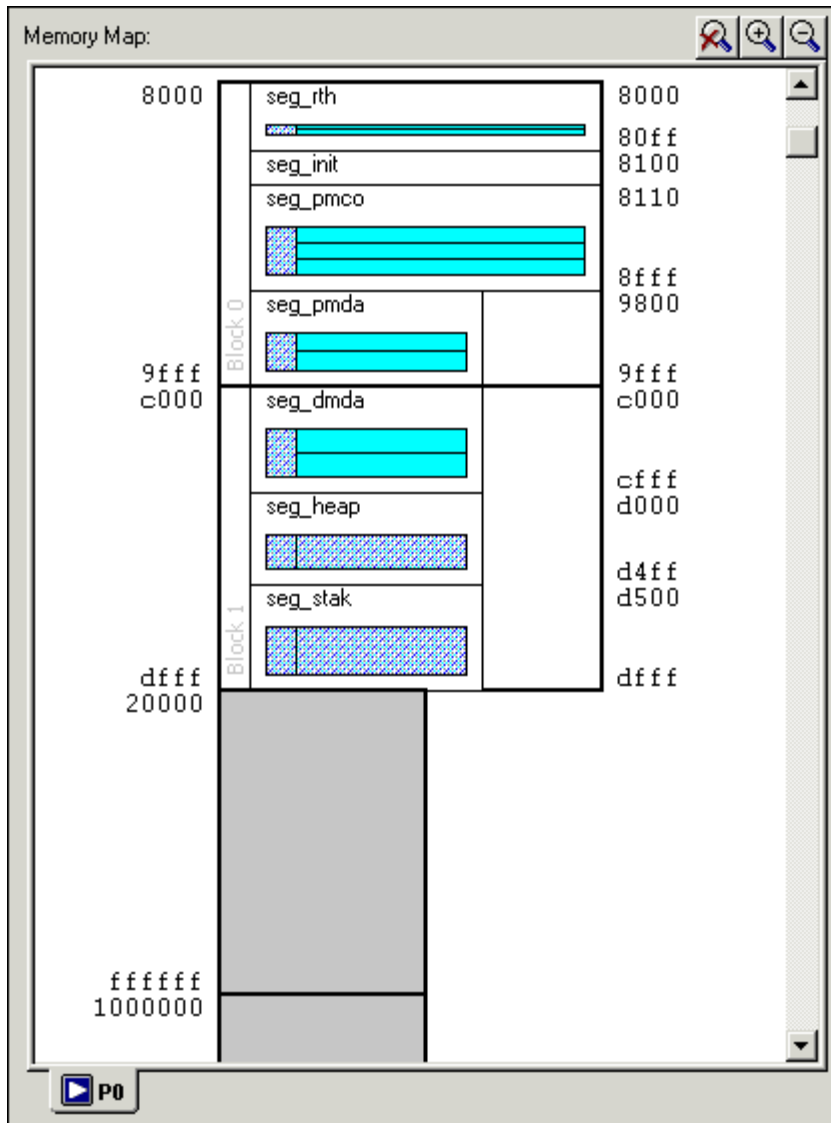


Figure 1-3. Memory Map Example After Running a SHARC Program

Archiver

The VisualDSP++ archiver (`elfar.exe`) combines object (`.DOJ`) files into library (`.DLB`) files, which serve as reusable resources for project development. The linker searches library files for routines (library members) referred to by other objects and links them in your executable program.

Run the archiver from within VisualDSP++ or from the command line. From VisualDSP++, create a library file as your project's output.

To modify or list the contents of a library file (or perform other operations on it), you must run the archiver from a command line. For details, refer to the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Splitter

The splitter (`elfspl21k.exe`) processes executable files to prepare non-bootable programmable read-only memory (PROM) image files. These files are executed from the processor's external memory.

The splitter's primary output is a PROM file with these extensions:

- `.S_#`
- `.H_#`
- `.STK`

For TigerSHARC processors, output from the splitter is 32 bit. For SHARC processors, output from the splitter is 32 bit, 40 bit, 48 bit, or 64 bit.

To specify splitter options, select **Project**, **Project Options**, and the **Split** tab.

Splitter terms are defined as follows.

Non-bootable PROM-image files

The splitter's output, which consists of PROM files that cannot be used to boot-load a system

Splitter

The splitter application, such as `elfspl21k.exe`, contained in the software release

For more information about the splitter and options used to generate loader files, refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors*.

Loader

The SHARC and TigerSHARC loader (`elfloader.exe`) generates boot-loadable files for 32-bit processors by processing executable files in addition to a loader kernel. The loader output (`.LDR`) file enables the DSP to boot from an external device (host or ROM).



The loader creates programs that execute from internal memory. The splitter generates programs that execute from external memory.

To specify loader options, select **Project**, **Project Options**, and the **Load** tab.

For more information about the options used to generate loader files, refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors*.

Code Development Tools

Loader terms are defined as follows.

Boot kernel

The executable file that performs the memory initialization on the target

Boot-loadable file

The loader's output(.LDR file), which contains the boot loader and the formatted system configurations. This file is a bootable image file.

Boot loading

The process of loading the boot loader, initializing system memory, and starting the application on the target

Loader

The loader application, such as `elfloader.exe`, contained in the software release

For more information about the loader, refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors*.

VCSE

VCSE consists of a combination of tools and guidelines that simplify the process of developing components and help to document and validate such components. These tools and guidelines:

- Enable applications to incorporate and use software algorithm components from other developers easily and with confidence
- Ensure that components from multiple vendors do not interact with each other in unpredictable ways or have resource clashes
- Allow components to be developed in assembler, C, or C++ and be used from applications developed in any of these languages
- Allow components to be reused easily
- Allow comparison of algorithms that offer the same functionality
- Provide support for testing of components as they are developed or used by an application
- Enable the use of components to be optimized automatically
- Encourage third-party developers to provide the implementation of algorithms as easily used components

For more information, refer to the *VisualDSP++ 3.5 Component Software Engineering User's Guide for 32-Bit Processors*.

VCSE Components

VCSE provides support for creating and using software components that are specifically targeted at the embedded space.

VCSE Component Model Specification

Components that adhere to the VCSE Component Model specification enable you to:

- Create software algorithms as reusable components
- Use software algorithms as components from other developers
- Use components from an assembler, C, or C++ program irrespective of the language in which they were implemented
- Use components from multiple sources predictably in any application without resource conflicts

VCSE Component Model

The VCSE component model does the following.

- Defines a binary standard to allow component interoperability
- Specifies a mechanism that is language independent and usable by assembly, C, and C++ components
- Provides a robust mechanism to cope with the evolution of components over time
- Defines a naming standard to ensure the uniqueness of component names

The binary standard defining the mechanism allows function calls between components and supports the grouping of available functions or methods into interfaces that are accessible as a unit. Each component can support more than one interface. Each component must support a base interface that can be used to access any other interface supported by the component.

VCSE Tools

VCSE tools help you create and use components without having to become familiar with the detail of the model and the mechanisms it involves. As a result, you can concentrate on the application itself.

VCSE consists of tools and guidelines that simplify the process of developing components and automate the conformance testing of such components. VCSE components are integrated with VisualDSP++. They simplify the process of incorporating and utilizing components from a variety of developers.

Use of VCSE Components With VisualDSP++

VCSE automatically generates an interface header file that defines the services it offers and provides access to those services from assembly, C, and C++ files.

VCSE components can be integrated with VisualDSP++, so you can view information on all the components that are available. From VisualDSP++, you can view a list of all the registered components in VisualDSP++. Some components may not have a complete implementation, but they are available for purchase. You can access the information for each registered component and view the list of interfaces it supports.

When you add a component to a project, VisualDSP++ adds the following.

- Relevant object and header files to the project
- Supported interfaces

You can use the New Interface Wizard to create an interface and add methods and parameters to it. This process generates the necessary Interface Definition Language (IDL) source code.

VCSE User Interface

The integration of VCSE with VisualDSP++ simplifies the process of creating components and of incorporating and utilizing components from a variety of developers. The VCSE menu, accessed from the **Tools** menu, includes options for:

- Creating a VCSE component project that supports the creation of a VCSE component
- Creating a compressed VisualDSP++ Component Package File (.VCP) for distributing a fully developed component
- Creating a new interface specification
- Creating a unique identifier (IID) for a manually created interface
- Installing and viewing components on the local system
- Downloading and installing new or updated components from the Analog Devices Web site
- Adding an installed component to a project

Tool Chain Integration

You tell VisualDSP++ to produce a VCSE component library from the **Project** page of the **Project Options** dialog box. Under **Type**, select **VCSE component library**. Specify VCSE compiler options from the **VIDL** page of the **Project Options** dialog box. Specify IDL font and color preferences for editing on the **Editor** page of the **Preferences** dialog box.

Wizards

VCSE wizards lead you through various tasks.

- New Component Project Wizard

Creating a new VCSE component project is simple with the New Component Project Wizard. After making the required decisions in the wizard, a new VCSE Component Library project is created and added to the current workspace. In addition, the IDL source code describing your component is generated and added to the project.

- New Component Package Wizard

Create a new component package with the New Component Package Wizard. Select an XML component manifest, review component information, and then include files in the component package.

- New Interface Wizard

Creating a new interface is easy with the New Interface Wizard. You define methods and parameters for the new interface, and the wizard generates the IDL source code for that interface and adds it to the current project.

Once a component is downloaded and installed on your system, you can easily add the component to a project.

Component Manager

You can view the list of currently installed components, add installed components to a project, and interactively view and download new components from the Analog Devices Web site. Use the Component Manager to browse components at various locations.

From the Component Manager dialog box, you can:

- Browse components

View the list of installed components on your system or the new and updated components on the ADI Web site. Each component includes a description.

- Filter your view of the components

Sort the component list by name, category, company, status, supported processor, or implemented interface.

- Install components

Install components directly from the Analog Devices Web site or from a third party. You must download the component to your PC from a Web site or copy the component to your PC by any means.

- Uninstall components from your PC

Structure of VCSE

VCSE specifies the requirements that a component must meet and provides a set of tools to help ensure that a component conforms to the ADI component standard for DSP algorithms and other objects. VCSE greatly simplifies the task of enabling components within a system to communicate. It also provides a degree of abstraction that offers greater flexibility in the choice and use of components.

VCSE support for DSP algorithms makes it much easier for integrators to exploit algorithms from one or more vendors. This support also provides the algorithm developer with a standardized framework to make algorithms more interoperable and usable at a minimal cost.

VCSE provides a set of specifications and tools that comprise:

- A software architecture that is designed to be efficient and effective for DSP applications and processors. The language-neutral architecture provides support for inter-working between software written in assembly, C, and C++. The architecture is designed to operate within multiple environments such as single or multithreaded applications.
- A component model that provides encapsulation (hiding of the implementation and other information) of the algorithms and objects and supports the idea of abstract interfaces and a single inheritance model. The VCSE component model is specifically designed for use within DSP and embedded applications.
- An IDL that supports simple bindings for C, C++, and assembly. The IDL is supported by the VCSE IDL (VIDL) compiler, which processes the IDL-specified component and interface definitions. The VIDL compiler also generates interface headers, component shells, and HTML-based documentation for the component.

IDL incorporates support for documenting the interfaces, and enables the generation of standardized documentation, which allows the other statements about the interface to be automatically validated or even generated.

- A set of rules and guidelines to which each component must adhere to be a conforming component or algorithm. Validation of conformance to some of these rules is effected automatically by VisualDSP++.
- An interface wizard that provides a visual user interface to allow the object or algorithm provider to define interfaces. The interface wizard generates an IDL specification of the interface, which you can then compile by using the VIDL compiler.

- Support for the inclusion of VCSE components in a project and the display of associated component documentation. Each VCSE component can be packaged as a compressed package, called a component package (.VCP) file. You can install this file into VisualDSP++ by using the same techniques used to install and identify debug-targets from third-party suppliers.

Each such package contains the component interface headers, documentation, and (optionally) the implementation and any necessary license information. Packages that do not contain the implementation provide a means of promoting a component or algorithm in a convenient form for VisualDSP++ developers.

Interface Definition Language (IDL) and Compiler

VCSE supports an Interface Definition Language (IDL) and compiler that enable developers to specify and then create and use components without having to become familiar with the detail of the model and its mechanisms. The VIDL compiler processes the specification of the interfaces supported by a component and generates the framework code needed to implement the component. The developer of the component can thus concentrate on providing the implementation of the methods that the component will provide. In addition, the VIDL compiler can generate a simple test harness to help in testing the component.

The VIDL compiler compiles the supplied IDL definitions of interfaces and components and generates up to five possible output items, which can be emitted for each interface. These items are:

- An interface header file, which can be used by a client of the interface to request services from the interface

Each interface header file can be used within assembly, C, and C++ source files.

- An interface component shell, which provides a standard framework for providing the implementation of each interface supported by the component in the chosen implementation language

This shell supports the interface but leaves the implementation of each method to the developer. Consequently, the developer is free to concentrate on the implementation of interface services without having to know the details of the VCSE binary standard.

The binary standard requires all functions within the interface to obey the C run-time model. The generated assembler shell for an interface ensures that the requirement is met by using the appropriate macros in the generated code. The component shell can be generated in the assembly, C, or C++ language.

- HTML-based documentation of the component and all of its supported interfaces in a standardized way

The documentation is derived from the IDL definition and the embedded auto-doc comments that the developer is encouraged to provide as part of the IDL definition.

- An .XML file, which can be used by the New Component Package Wizard when a component is packaged for distribution
- A test shell component, which can be used to automatically validate the behavior of a component or ensure that an application is using a component properly

In addition to generating checks automatically, you can add additional validation for each method in an interface.

The IDL language supports all the standard C/C++ base types (as well as arrays, structs, and enums) and the use of `typedef`. The only explicit pointer types that IDL supports are interface pointers. All *out* parameters are mapped to arrays or pointers. All *in* arrays and structs are also mapped to arrays and pointers.

The VIDL compiler inputs `.IDL` files and produces the files shown in Figure 1-4.

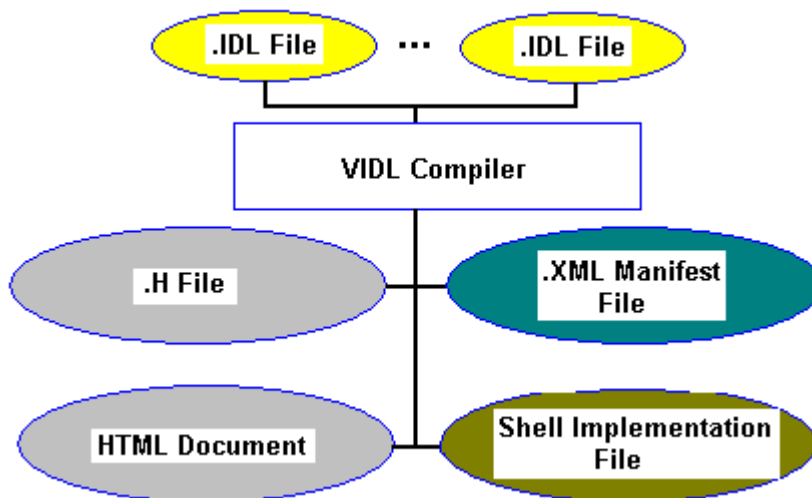


Figure 1-4. Files Produced by the VIDL Compiler

DSP Projects

Your goal is to create a program that runs on a single-processor (or multi-processor) system. A project is the structure where DSP programs are built. All development in VisualDSP++ occurs within a project.

A project refers to the collection of source files and tool configurations used to create a DSP program. A project (`.DPJ`) file stores program build information.

VisualDSP++ provides flexibility for setting up projects. You configure settings for DSP code development tools and configurations, and you specify build settings for the project and for individual files. You can set up folders that contain your source files. A project can include VDK support.

Use the **Project** window to manage projects from start to finish. Within the context of a DSP project, you can:

- Specify DSP code development tools
- Specify project-wide and individual-file options for Debug or Release configurations of project builds
- Create source files

VisualDSP++ facilitates movement among editing, building, and debugging activities.

Project Options

Project options apply to the entire DSP project. Specify project options in the **Project Options** dialog box. [Figure 1-5](#) shows the top portion of this dialog box.

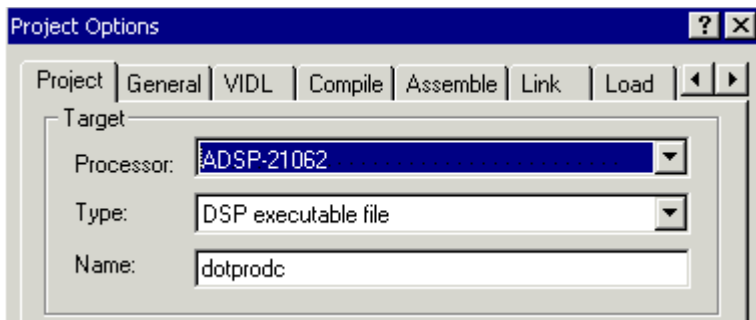


Figure 1-5. Top Portion of the Project Options Dialog Box

For each code development tool (compiler, assembler, linker, splitter and loader), a tabbed page provides options that control how each tool processes inputs and generates outputs. The available pages depend on your target. Options correspond to an individual tool's command-line switches. You can define these options once or modify them to meet changing development needs.

 Tools can also be accessed from the operating system's command line.

Project options also specify the following information.

- Project target
- Tool chain
- Output file directories
- Post-build options

Project Groups

Project groups enable you to work with a number of projects at once. A project group can be empty or contain any number of projects. Opening a project adds it to the project group. Closing a project removes it from the project group.

Similar functionality is found in Microsoft Visual Studio. The capabilities provided by project groups are particularly useful in VCSE development, which involves multiple projects.

The **Project** window (Figure 1-6) displays the project group icon and the projects opened in that workspace.

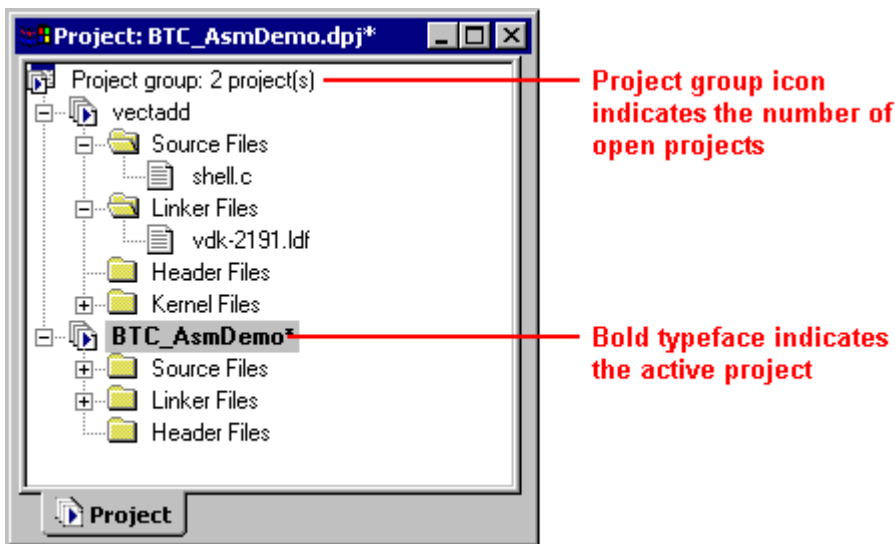


Figure 1-6. Project Window

Each workspace has one project group. When you switch among workspaces, the project group is loaded and the same set of projects are opened just as when you last closed the workspace.

DSP Projects

One project is active at a time. The active project responds to commands and messages from menus and toolbars. The **Project** window displays the active project with bold typeface. A **Project** box, located by default with the toolbar buttons, displays the name of the active project (see [Figure 1-7](#)).



Figure 1-7. Project Box Showing the Active Project

Though commands are sent to the active project, they may also be carried out by a project on which the active project depends. For example, assume that project A is active and depends on project B. Executing a **Rebuild All** command on project A builds project B first. The same logic applies to the **Clean** command, which deletes intermediate and target files.

Exporting a makefile exports one makefile for each open project. In the makefile of a project depending on another project, one subtarget is created for each project on which it depends. Thus, building a project builds all dependent projects first.

Source Code Control (SCC)

VisualDSP++ includes Source Code Control (SCC), which enables you to use the Microsoft Common Source Code Control (MCSCC) interface to connect the VisualDSP++ IDDE to SCC applications installed on your machine.

Various SCC products (such as Visual SourceSafe, PVCS Version Manager, and ClearCase) support the MCSCC interface. Using the convenient VisualDSP++ interface, you can access the commonly used features of these applications without leaving the IDDE. You can launch the SCC application from the plug-in menu to use non-supported features.

When you create a project, you are prompted to add the project to SCC. When you open a project in the IDDE, the SCC plug-in connects to the selected SCC application and locates a controlled copy of the project and its source files. If a controlled copy is not located, the SCC application must locate it. Typically, you are queried to browse for it. If the controlled copy is successfully found or added, the plug-in keeps its application-specific path in the project file and reconnects with this path in the future. You can subsequently reconnect to the controlled copy without having to browse for it.

Operations executed on large numbers of files tend to take longer to finish. A message box provides status information by displaying the operation currently executing. A button on the message box cancels the operation. The **Output** window's **Console** view displays finished operations. Messages indicate what has been done. Warnings and error messages may also appear in the **Output** window.

SCC applications provide dialog boxes and GUI displays for some file operations such as show history, show difference, and show properties. These operations can be run from VisualDSP++.

Makefiles

Use a makefile (.MAK or .MK) to automate builds within VisualDSP++. The output make rule is compatible with the gnumake utility (GNU Make V3.77 or higher) or other make utilities. VisualDSP++ generates a project makefile that controls the orderly sequence of code generation in the target. You can also export a makefile for use outside of VisualDSP++. For more information about makefiles, go to:

<http://www.gnu.org/manual/make/>

A project can have multiple makefiles, but only one makefile can be enabled (active).

The project in [Figure 1-8](#) includes an active makefile (indicated by ).

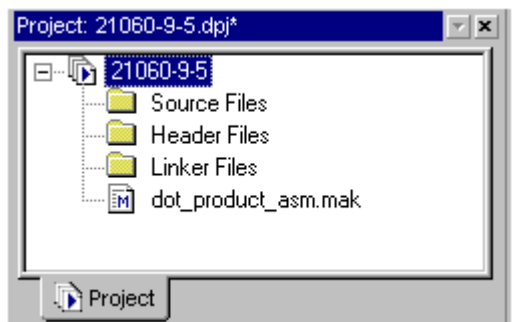


Figure 1-8. Enabled Makefile dot_product_asm.mak

The active makefile, with its explicit `gmake` command line, builds the project. When no makefile is enabled for a project, VisualDSP++ uses specifications configured in the **Project Options** dialog box.

You can view a makefile's command line. To change the makefile's target, use the **Configuration** box, shown in [Figure 1-9](#).



Figure 1-9. Configuration Box

When you close a project, the Make commands and the target list associated with each makefile are serialized in the project (.DPJ) file.

Rules

You can enable only one makefile when building a project. If you enable more than one makefile, VisualDSP++ generates an error message. After you build your project with an external makefile, the executable file is not automatically loaded (even when this option is configured).

Output Window

Make command error messages and standard output appear in the **Output** window. Double-clicking on an error-message position opens the makefile in an editor window to the line of code causing the error.

Keywords in the makefile are syntax-colored.

Note: The error message format of `gmake` is parsed correctly when you double-click on an error message. If you use another make utility, the double-click feature does not function.

Example Makefile

An example of a makefile appears below.

```
# Generated by the VisualDSP++ IDDE

# Note: Any changes made to this Makefile will be lost the next
# time the matching project file is loaded into the IDDE. If you
# wish to preserve changes, rename this file and run it
# externally to the IDDE.

# The syntax of this Makefile is such that GNU Make v3.77 or
# higher is required.

# The current working directory should be the directory in which
# this Makefile resides.

# Supported targets:
#     Debug
#     Debug_clean
#     Release
#     Release_clean

# Define ADI_DSP if it is not already defined. Define this
# variable if you wish to run this Makefile on a host other than
# the host that created it and VisualDSP++ may be installed in a
# different directory.

ifndef ADI_DSP

ADI_DSP=C:\Program Files\Analog Devices\VisualDSP

endif

# $VDSP is a gmake-friendly version of ADI_DIR

empty:=

space:= $(empty) $(empty)
```



```
VDSP_INTERMEDIATE=$(subst \,/,$(ADI_DSP))

VDSP=$(subst $(space),\$(space),$(VDSP_INTERMEDIATE))

# Define the command to use to delete files (which is different
# on Win95/98 and Windows NT/2000)

ifeq ($(OS),Windows_NT)

RM=cmd /C del /F /Q

else

RM=command /C del

endif

#

# Begin "Debug" configuration

#

ifeq ($(MAKECMDGOALS),Debug)

Debug : ./debug/mean.dxe

./debug/mean.doj :./mean.c ../../include/stdio.h

$(VDSP)/cc21k -c .\Mean.c -g -proc ADSP-21062 -o
.\Debug\Mean.doj

./debug/benchmark.doj :./benchmark.asm
../../include/asm_sprt.h ../../include/def21060.h

$(VDSP)/easm21k.exe -proc ADSP-21062 -o .\Debug\benchmark.doj -g
.\benchmark.asm

./debug/mean.dxe :./debug/mean.doj ./debug/benchmark.doj

$(VDSP)/cc21k.exe .\Debug\Mean.doj .\Debug\benchmark.doj -proc
```

DSP Projects

```
ADSP-21062 -L .\Debug -flags-link -od,.\Debug -o .\Debug\Mean.dxe

endif

ifeq ($(MAKECMDGOALS),Debug_clean)
Debug_clean:$(RM) ".\Debug\Mean.doj"
              $(RM) ".\Debug\benchmark.doj"
              $(RM) ".\Debug\Mean.dxe"
              $(RM) ".\Debug\*.ipa"
              $(RM) ".\Debug\*.opa"
              $(RM) ".\Debug\*.ti"

endif

# Begin "Release" configuration

#

ifeq ($(MAKECMDGOALS),Release)

Release : ./release/mean.dxe

./release/mean.doj :./mean.c

$(VDSP)/cc21k -c .\Mean.c -O1 -proc ADSP-21062 -o
.\Release\Mean.doj

./release/benchmark.doj :./benchmark.asm

$(VDSP)/easm21k.exe -proc ADSP-21062 -o .\Release\benchmark.doj
.\benchmark.asm

./release/mean.dxe :./release/mean.doj ./release/benchmark.doj

$(VDSP)/cc21k.exe .\Release\Mean.doj .\Release\benchmark.doj
-proc ADSP-21062 -L .\Release -flags-link -od,.\Release -o
.\Release\Mean.dxe

endif
```

```

ifeq ($(MAKECMDGOALS),Release_clean)

Release_clean:
    $(RM) ".\Release\Mean.doj"
    $(RM) ".\Release\benchmark.doj"
    $(RM) ".\Release\Mean.dxe"
    $(RM) ".\Release\*.ipa"
    $(RM) ".\Release\*.opa"
    $(RM) ".\Release\*.ti"

endif

```

Project Configurations

By default, a project includes two configurations, Debug and Release, described in [Table 1-6](#). In previous software releases, the term *configuration* was called “build type.”

Table 1-6. Default Project Configurations

Configuration	Description
Debug	Builds a project that enables you to use VisualDSP++ debugging capabilities
Release	Builds a project with optimization enabled

Available configurations appear in the configuration box, which is part of the **Project** toolbar, as shown in [Figure 1-10](#).



Figure 1-10. Configuration Box



You cannot delete the Release or Debug configuration.

Customized Project Configurations

You can add a configuration to your project. A customized project configuration can include various project options and build options to help you develop your project. [Figure 1-11](#) shows a customized configuration (**Version2**) listed in the configuration box.

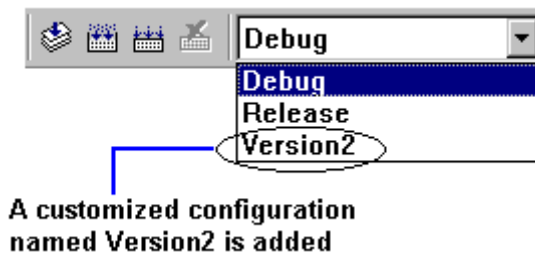


Figure 1-11. Customized Configuration Version2

Project Build

The term *build* refers to the process of performing operations (such as preprocessing, assembling, and linking) on projects and files. During a build, VisualDSP++ processes project files that have been modified since the previous build as well as project files that include modified files.

A build differs from a *rebuild all*. When you execute the **Rebuild All** command, VisualDSP++ processes all the files in the project, regardless of whether they have been modified.

Building a project builds all outdated files in the project and enables you to make your program. An *outdated file* is a file that has been modified since the last time it was built or a file that includes a modified file. For example, if a C file that has not been modified includes a header file that has been modified, the C file is out of date.

VisualDSP++ uses *dependency* information to determine which files, if any, must be updated during a build.



Note the following.

- A file with an unrecognized file extension is ignored at build time.
- If an included header file is modified, VisualDSP++ builds the source files that include (`#include`) the header file, regardless of whether the source files have been modified since the previous build.
- File icons in the **Project** window indicate file status (such as excluded files or files with specific options that override project settings).

Build Options

You can specify options for the entire project and for individual files. [Table 1-7](#) describes these build options.

Table 1-7. Build Options

Options	Description
Project-wide	Specify these options from a tabbed page (for example, Compile or Link) for each of the DSP code development tools.
Individual-file	These settings override project-wide settings.
Custom-build step	For maximal flexibility, edit the command line(s) issued to build a particular file. For example, you might call a third-party utility.

File Building

Building a file compiles or assembles the file and locates and removes errors. You can build a single file or multiple files that you select.

The build process updates the selected source file's output (.OBJ) file and the output file's debug information. Building a single file is very fast. Large projects, however, may require hours to build.

If you change a common header file that requires a full build, you can build only the current file to ensure that your change fixes the error in the current file.

Post-Build Options

Post-build options are typically DOS commands that are executed after a successful project build. These commands invoke external tools.

For example, you can use a post-build command to copy the final output file to another location on the hard drive or to invoke an application automatically.

Automatically copying files and cleaning up intermediate files after a successful build can be very useful.

Command Syntax

Depending on your operating system, you must place “cmd /C” or “command /C” at the beginning of each DOS command line.

For example, to execute `copy a.txt b.txt`, use one of the commands shown in [Table 1-8](#). The “C” after the slash in the commands must be uppercase.

Table 1-8. Operating System and Required Command Syntax

Operating System	Command
Windows 98	<code>command /C copy a.txt b.txt</code>
Windows Me	<code>command /C copy a.txt b.txt</code>
Windows NT	<code>cmd /C copy a.txt b.txt</code>
Windows 2000	<code>cmd /C copy a.txt b.txt</code>
Windows XP	<code>cmd /C copy a.txt b.txt</code>

Project Dependencies

Dependency data determines which files must be updated during a build. The following are examples of dependency information.

```

..\..\include\cplus\cstddef
..\..\include\cplus\exception
..\..\include\cplus\new
..\..\include\cplus\xstddef
..\..\include\def21060.h
..\..\include\limits.h
..\..\include\cplus\stddef
..\..\include\stdio.h
..\..\include\string.h
..\..\include\VDK_Internals.h
..\..\include\VDK_Public.h
..\..\include\yvals.h

```

Project Rules

The **Project** window displays a project's files, as shown in [Figure 1-12](#).

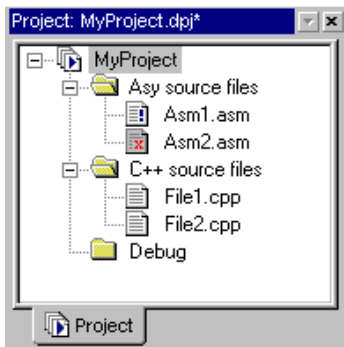


Figure 1-12. Example of Project Files

The following rules dictate how files and subfolders behave in the **Project** window's file tree.

- You can include any file in a project.
- Only one .LDF file is permitted.
- You cannot add the same file into the same project more than once.
- Only one project (project node) is permitted.
- A file with an unrecognized file extension is ignored at build time.
- When you add a file to a project, the file is placed in the first folder configured with the same extension. If no such folders are present, an added file goes to the project level.

VisualDSP++ Help System

The VisualDSP++ Help system is designed to help you obtain information quickly. Use the Help system's table of contents, index, full-text search function, bookmark function, and extensive hyperlinks to jump to topics.

VisualDSP++ Help is a merge of several Help systems (.CHM files). Each is identified with a book icon  in your product installation's Help folder.

Most of the Help system comprises VisualDSP++ tools manuals, such as the Assembler and Preprocessor manuals. These manuals are also provided in PDF format (on installation disk) for printing and are available from Analog Devices as printed books.

Some .CHM files support pop-up messages for dialog box controls (buttons, fields, and so on). These messages, which appear in little yellow boxes, compose part of the context-sensitive Help in VisualDSP++.

The Help system describes the VisualDSP++ user interface. Help files include concepts, procedures, and reference information. Each toolbar button, menu-bar command, and debugging window in VisualDSP++ is linked to a topic in one of these files.

For details about using the Help system, refer to [“Online Help Features and Operations”](#) on page A-48.

2 ENVIRONMENT

This chapter describes the features of the VisualDSP++ environment, including the main window, debugging windows, window operations, and customization.

The topics are organized as follows.

- [“Parts of the User Interface” on page 2-1](#)
- [“VisualDSP++ Windows” on page 2-15](#)
- [“Window Operations” on page 2-46](#)
- [“Debugging Windows” on page 2-52](#)

Parts of the User Interface

VisualDSP++ is an intuitive, easy-to-use user interface for programming Analog Devices processors. When you open VisualDSP++, the application’s main window appears. [Figure 2-1](#) shows an example of the VisualDSP++ main window.

This work area contains everything necessary to build, manage, and debug your DSP project. You can set up preferences that specify the appearance of application objects (fonts, visibility, and so on). You can open project files by dragging and dropping them into the main window.

Parts of the User Interface

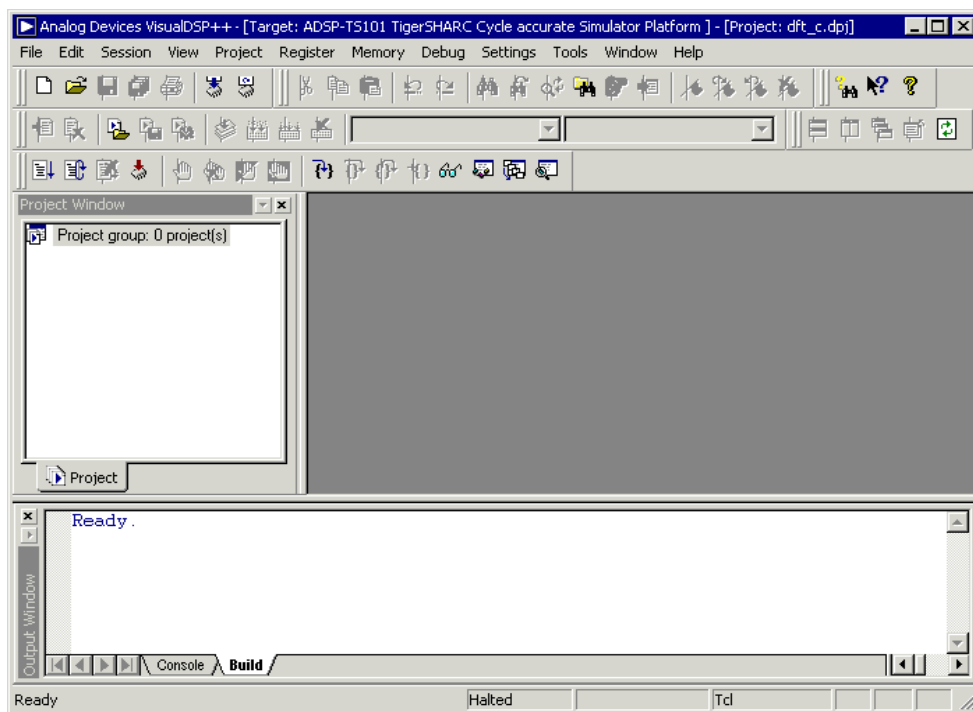


Figure 2-1. VisualDSP++ Main Window

The VisualDSP++ main window includes these parts:

- Title bar
- Menu bar
- Project window
- Control menu
- Toolbars

- Output window
- Status bar

VisualDSP++ also provides many debugging windows to facilitate project development. You have to learn only one interface to debug all your DSP applications.

VisualDSP++ supports ELF/DWARF-2 (Executable Linkable Format, Debugging Information Format) executable files. VisualDSP++ supports all executable file formats produced by the linker.

Title Bar

Figure 2-2 shows the different parts of the title bar.

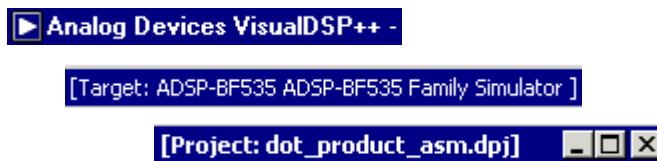


Figure 2-2. Title Bar (Split into Three Parts to Fit the Page)

The title bar includes these components:

- Control menu button 
- Application name – Analog Devices VisualDSP++
- Name of the active target
- Project name
- File name (when an editor window is maximized in the main window)
- Standard Windows buttons

Parts of the User Interface

Clicking the control menu button opens the control menu, which contains commands for positioning, resizing, minimizing, maximizing, and closing the window. Double-clicking the control button closes VisualDSP++. The title bar right-click menu ([Figure 2-3](#)) and control menu ([Figure 2-4](#)) are identical.

Additional Information in Title Bars

A register window title bar displays its numeric format (such as hexadecimal). An editor window title bar displays the name of the source file.

Title Bar Right-Click Menus

A menu like the one in [Figure 2-3](#) appears when you right-click within the VisualDSP++ title bar or within the title bar of a child (sub) window.



Figure 2-3. Right-Clicking in the Window's Title Bar

From the VisualDSP++ title bar's right-click menu, you can:

- Resize or move the application window
- Close VisualDSP++

Control Menu

Commands on a control menu (system menu, shown [Figure 2-4](#)) move, size, or close a window.



Figure 2-4. VisualDSP++ Control Menu

Program Icons

Click a program icon to open a control menu.

 Program icon for the application and debugging windows

 Program icon for editor windows

When you place the mouse pointer over a control menu command, a brief description of the command appears in the status bar at the bottom of the application window.

Editor Windows

A floating editor window's control menu includes **Next**, which moves the focus to another window.

When an editor window floats in the main application window, its program icon resides at the left side of its title bar. When an editor window is maximized, the program icon resides at the left end of the menu bar.

Parts of the User Interface

Debugging Windows

Each debugging window has a control menu. You can open a debugging window's control menu only when the window is floating in the main window. For more information, see [“Debugging Windows” on page 2-52](#).

Menu Bar

The menu bar, shown in [Figure 2-5](#), appears directly below the application title bar and displays menu headings, such as **File** and **Edit**.

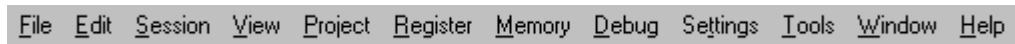


Figure 2-5. VisualDSP++ Menu Bar

To display menu commands and submenus, click a menu heading. You can also access many menu bar commands as follows.

- Click toolbar buttons
- Type keyboard shortcuts
- Right-click the mouse and choose a command from a context menu

Command Information

When the mouse pointer is over a menu bar command (or a toolbar button), a short description (tool tip) of the command appears in the status bar at the bottom of the main window.

Context-sensitive Help is available for each command.

To learn more about an individual menu command:

- Press **Shift+F1** or click the toolbar's Help button .

The pointer becomes a Help pointer .

- Move the Help pointer over a menu command.

If necessary, navigate through submenus.

- Click the mouse to display Help.

View the description of the command in the Help window.

Toolbars and User Tools

A toolbar is a set of buttons. You can run a command quickly by clicking a toolbar button.

Use toolbars to organize the tasks most often used. Position the toolbars on the screen for fast access to the tools that you plan to use.

The application includes standard (built-in) toolbars, though you can create custom toolbars.

Toolbar Customization

By default, nine standard toolbars appear near the top of the application window, below the menu bar.

You can change the appearance of toolbars by:

- Moving, docking, or floating the toolbars
- Adding or removing buttons to or from toolbars
- Displaying *cool look* buttons, *large buttons*, or both

You can also:

- Hide toolbars from view
- Add and delete custom-built toolbars

Toolbars: Docked Versus Floating

By default, toolbars are located under the application's menu bar, but they can be moved to these locations.

- Over a docked window
- On the main window
- Anywhere on the desktop

When a toolbar is attached to a window, it is called a *docked* toolbar. You can tell when a toolbar is going to dock by the size and shape of its moving outline as you drag it. Its outline becomes slightly smaller than its floating outline. To prevent a toolbar from docking, press and hold the **Ctrl** key while dragging the toolbar to a new location.

Parts of the User Interface

A toolbar can be detached from a window and moved to another location anywhere on the desktop. A *floating* toolbar is a standalone window, as it is not docked. A docked toolbar does not show its name, but a floating toolbar displays its title.

Figure 2-6 shows a floating Help toolbar.



Figure 2-6. Floating Help Toolbar

Toolbar Button Appearance

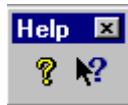
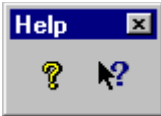
You can choose the appearance of the toolbar buttons. Two options, *cool look* and *large buttons*, provide slightly different button appearances.

The *cool look* option includes a pair of vertical bars on the toolbar's left side, but removes the square box from each button. The vertical bars visually separate toolbar buttons into groups (toolbars).

The *large buttons* option makes the area of each button larger.

Table 2-2 shows how small and large buttons appear with the *cool look* option turned off (disabled) and on (enabled).

Table 2-2. Toolbars in Different Viewing Options

Option Settings	Docked	Floating
Cool look – Off Large buttons – Off		
Cool look – On Large buttons – Off		
Cool look – Off Large buttons – On		
Cool look – On Large buttons – On		

Toolbar Shape

You can change the shape of a floating toolbar. [Table 2-3](#) shows two toolbar shapes.

Table 2-3. Toolbars in Two Orientations

Horizontal	Vertical
	

Depending on the number of tools in the toolbar, you can create other length and width arrangements.

Toolbar Rules

When working with toolbars, be aware of these rules:

- You can customize a built-in toolbar (for example, by removing a button from the **File** toolbar), but you cannot delete a built-in toolbar. You can reset the buttons in a built-in toolbar to their original default settings.
- You can change the name of a user-defined toolbar, but not the name of a built-in toolbar. For example, the **File** toolbar cannot be changed to a different name.

User Tools

Save time running commands by configuring user tools. Up to ten user tools can be configured.

A user tool runs a command, which can:

- Contain parameters to launch an application
- Be a script command

Access configured user tools from the **Tools** menu or from the **User Tools** toolbar, as shown in [Figure 2-7](#).



Figure 2-7. Default User Tools

When a user tool is configured, its menu name (label) appears in the **Tools** menu. The label also appears when you move the mouse pointer over a user tool button.

Status Bar

The status bar, located at the bottom of the main application window, provides various informational messages. [Figure 2-8](#) shows different information displayed on the status bar.



Figure 2-8. The Status Bar's Appearance Depends on Context

Parts of the User Interface

The type of information that appears in the status bar depends on your context (what you are doing).

- When you move the mouse pointer over a toolbar button or a menu bar command, a brief description of the button or command appears.
- When you halt program operation with a **Halt** command, the address where the program halted appears.
- When you use some script commands, the status bar provides information, such as when the menu item has focus.

While you are editing a file, the right side of the status bar provides editor window information, described in [Table 2-4](#).

Table 2-4. Status Bar Information While Editing

Item	Indicates
Line ###	Cursor current line number
Col ###	Cursor current column number
CAP	The keyboard's Caps Lock key is latched down
NUM	The keyboard's Num Lock key is latched down
SCRL	The keyboard's Scroll Lock key is latched down

VisualDSP++ Windows

From the application's main window, you can open a **Project** window, editor windows, an **Output** window, and various debugging windows.

Project Window

To open a **Project** window, choose **View** and **Project Window**. [Figure 2-9](#) shows a **Project** window with VDK enabled.

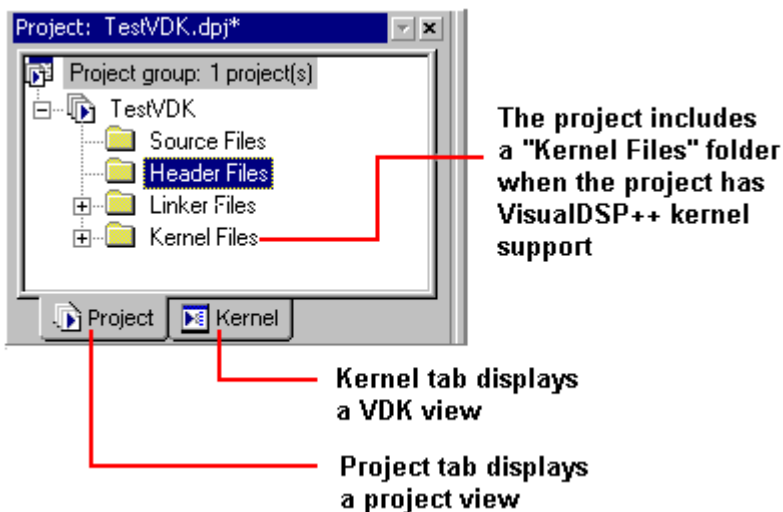


Figure 2-9. Project Window With Kernel Tab

The **Project** window can include two subtabs:

- The **Project** tab , which is always available, provides a hierarchal representation of a debug session's projects, folders, files, and dependencies.
- The **Kernel** tab  appears when VDK is enabled for a project.

Project View

The **Project** view displays a project group, which may contain any number of projects. Only one project, however, is active at a time. Nodes are arranged in a hierarchy similar to the file structure in Windows Explorer.

Figure 2-10 shows some of the information displayed in the **Project** view.

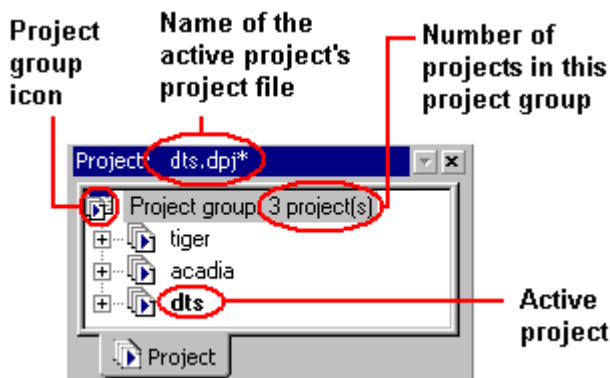


Figure 2-10. Project View

Project Dependencies

A project may depend on other projects. The  icon indicates dependency and identifies the dependency. Building a project also builds the projects on which it depends.

Figure 2-11 shows how project dependencies are indicated in the **Project** view.

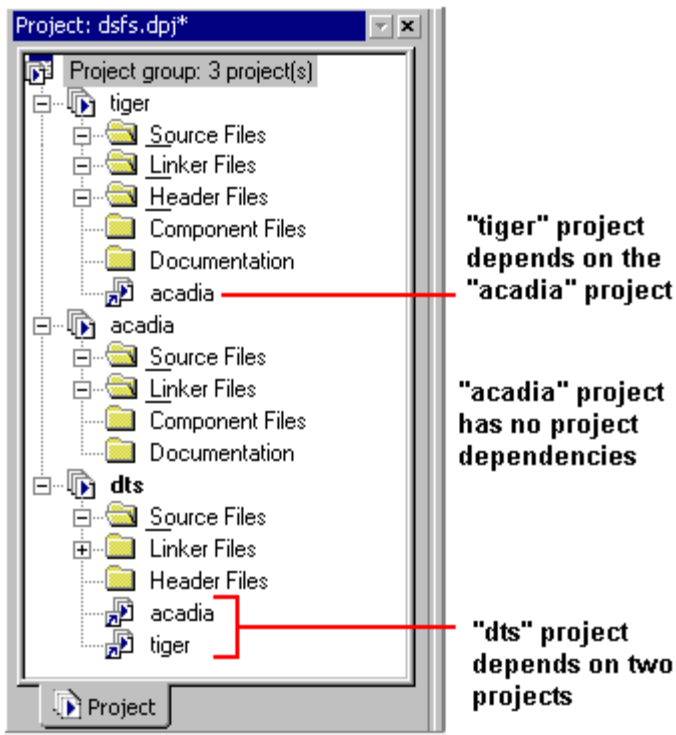


Figure 2-11. Projects Dependencies Indicated in the Project View

Project Nodes

The **Project** window comprises the types of nodes described in [Table 2-5](#).

Table 2-5. Types of Nodes in the Project Window

Node	Icon	Description
Project group		Only one project group permitted in a debug session
Project		Multiple projects permitted, but only one is active (indicated with bold typeface)
Folder		Closed folder
		Opened folder revealing its contents
File		File that uses project settings
		File whose options differ from the project options
		File excluded from the current configuration
		Enabled (active) makefile
Project dependency		Project on which another project depends

Project Page Right-Click Menus

Right-click menus (also called context menus or pop-up menus) operate on **Project** window objects (the project group, projects, folders, and files). These menus provide fast access to many menu bar and toolbar commands. The commands available from a right-click menu depend on context (the selected object).

Project window right-click menus offer these standard commands:

- **Allow Docking** (dock the **Project** window to the frame)
- **Hide** (remove the **Project** window from view)
- **Float in Main Window**

Project Group Icon Right-Click Menu

The project group icon () right-click menu ([Figure 2-12](#)) provides a project group context from which you can:

- Create a new project
- Open a project and add it to the project group
- View the project group's properties



Figure 2-12. Project Group Icon's Right-Click Menu

Project Icon Right-Click Menu

The project icon () right-click menu ([Figure 2-13](#)) provides a project context from which you can:

- Build the project
- Clean (delete intermediate and target files)
- Specify the active project
- Add folders and files
- View and specify project options
- View project properties

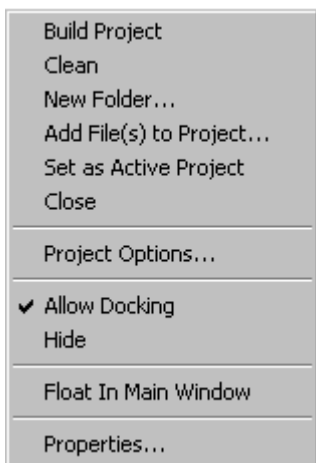


Figure 2-13. Project Icon's Right-Click Menu

Folder Icon Right-Click Menu

The selected folder icon ( or ) right-click menu ([Figure 2-14](#)) provides a “container” context from which you can perform these “local” operations:

- Add or delete a folder
- Add files to the folder
- View folder properties



Figure 2-14. Folder Icon Right-Click Menu

File Icon Right-Click Menu

The selected file icon ( or  or ) right-click menu ([Figure 2-15 on page 2-22](#)) provides a file context from which you can:

- Open the selected file for editing
- Build the file
- Remove the file from a project
- Specify options for the file
- View the file’s properties



Figure 2-15. File Icon Right-Click Menu

 File icon commands apply to the selected file in the **Project** window, not to a source file in an editor window.

Project Folders

Project window folders ( and ) organize files within a project. You can specify properties for folders.

Folders can be nested to any depth. Folders carry no attributes to the build process, as they do not reflect the file system. Folders do not appear in directory listings, as in Windows Explorer.

When you add files to the project tree with automatic file placement, each file is placed in the first folder configured with the same file extension. After automatic placement, you can manually move a file anywhere.

To move a file out of one folder and into another folder, select the file and drag it onto the other folder.

Project Files

In the **Project** window, files are represented by the icons in [Table 2-6](#).

Table 2-6. Icons in the Project Window

Icon	Description
	Files that use project options
	Files that use options that differ from project options
	Files excluded from the current configuration
	Enabled (active) makefile

The files appear in an expandable and collapsible node tree.

Source files are the C/C++ language or assembly language files in your project. Source files provide the project with code and data. You can add, delete, and modify source files.

Each project must include an `.LDF` file, which contains command input for the linker. If an `.LDF` file is not included in the project, the project is built with a default `.LDF` file.

A DSP project can also include data files and header files.

Project Window Icons for Source Code Control (SCC)

Icons in the **Project** window indicate source code control (SCC) status. Files with a green check mark () are under SCC and are checked in. Files with a red check mark () are under SCC and are checked out. When a file is not connected to a controlled copy under SCC, the file icon has no check mark.

The file icons in [Table 2-7](#) indicate SCC status.

Table 2-7. SCC Status Icons

Icon	Description
	File is under SCC and is checked in
	File is under SCC and is checked out
	Project file is checked out
	File includes a file-specific build command and is checked out
	Makefile is checked out
	File is excluded from the build and is checked out

File Associations

VisualDSP++ associates the file extensions in [Table 2-8](#) as the input to particular DSP code development tools.

Table 2-8. File Associations

Tool	File Extensions
Compiler	.C, .CPP, and .CXX
Assembler	.ASM, .S, and .DSP
Linker	.LDF, .DLB, and .DOJ



Note that VisualDSP++ is case insensitive to file extensions.

Automatic File Placement

Automatic file placement enables you to drag and drop files into designated folders on the **Project** page in the **Project** window. This feature saves time when you add files to a project.

Folder properties that you specify and file placement rules determine where files are placed. By default, project folders are associated with the file extensions listed in [Table 2-9](#).

Table 2-9. Files Associated With Project Folders

Folder	Default Associations
Source Files	.C, .CPP, .CXX, .ASM, .DSP, .S
Header Files	.H, .HPP, .HXX
Linker Files	.LDF, .DLB, .DOJ
Kernel Files	.VDK

File Placement Rules

The following rules dictate file placement when you add files to a project.

- Dragging and dropping files

When you drag and drop a file onto the **Project** page, the file is added to the first folder associated with the file's extension. The **Project** page accepts dragged files only when a project is opened.

- Using menu commands to add files

Files are added to the folders that you select on the **Project** page. If you add a file to a project that has no folders, the file is added at the project level (root level).

If you select the project node or a file node, the file is added to the first folder associated with the file's extension.

Example

You create a folder labeled “C Source Files” and specify it with `.C`, `.CPP`, and `.CXX` file extensions. You create a second folder labeled “Asm Files” and associate it with `.ASM` files.

If you drag three files (`file1.cpp`, `file1.asm`, and `file2.c`) into the **Project** window, `file1.cpp` and `file2.c` go into the C Source Files folder, and `file1.asm` goes into the Asm Files folder.



After automatic file placement, you can manually move a file anywhere by selecting and dragging the file.

Kernel Page

The **Kernel** tab of the **Project** window is available only to VDK-enabled projects.

From the **Kernel** page, you can add, modify, and delete kernel elements such as thread types, priorities, semaphore, and events. VisualDSP++ automatically updates `vdk_config.cpp` and `vdk_config.h` to reflect the changes made on the **Kernel** page.

The example in [Figure 2-16](#) shows an expanded view of the elements on the **Kernel** page for a VDK-enabled project.

For complete details about VDK, refer to the *VisualDSP++ 3.5 Kernel (VDK) User's Guide for 32-Bit Processors*.

VisualDSP++ Windows

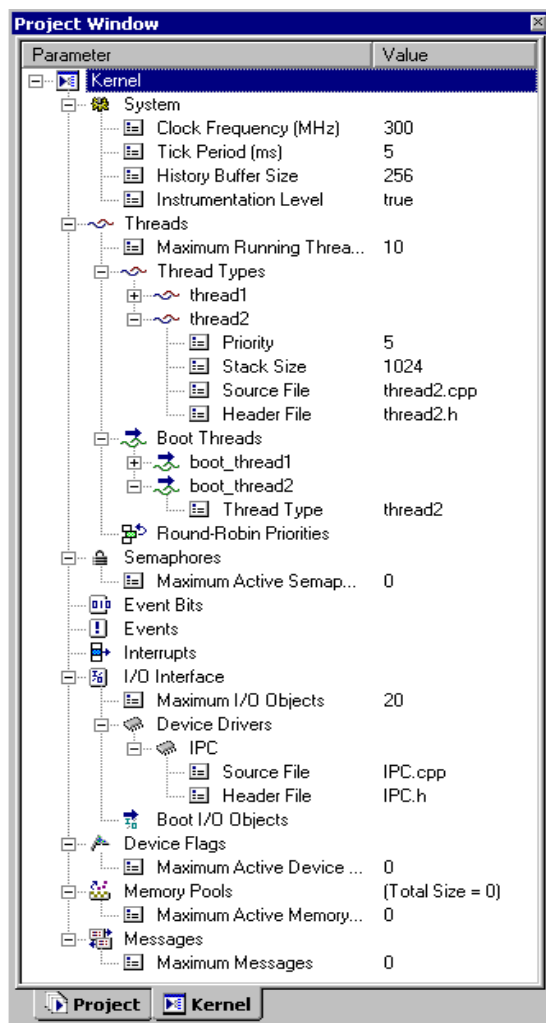


Figure 2-16. Expanded View of Elements on the Kernel Page

Editor Windows

Use editor windows to view and edit project files. Open an editor window from the **Project** window by double-clicking on a file or by choosing **Open File** from a file's right-click menu. Figure 2-17 shows items that can be customized in editor windows.

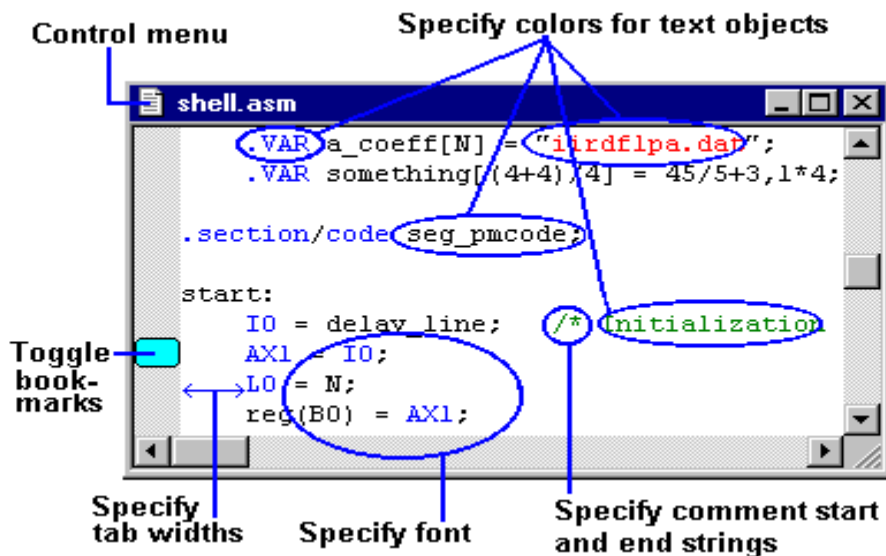


Figure 2-17. Items That Can Be Customized

You can open as many editor windows as you like and:

- Define color-coded comments, strings, keywords, and tabs
- Preview and print window data
- Load a script
- Define headers and footers
- Set bookmarks

VisualDSP++ Windows

- Find, replace, use wraparound search and expression matching
- Go to a specified line number
- Jump to the next or previous syntax error
- Copy, cut, paste, undo, and redo more than 500 levels of edits for each open file
- Enable **Editor Tab** mode to switch quickly between source files (see [“Editor Tab Mode” on page 2-31](#))
- Locate matching brace characters and auto-position brace characters (to line up with the preceding opening brace)
- Open header files from the right-click menu. When you right-click on an `#include` statement, choose **Open Document** “filename.h” to open that file.
- Drag and drop highlighted sections of text (usually a valid source statement) to an open **Expressions** window. When dropped, the text is automatically added to the window and is evaluated.

Right-Click Menu

The editor window’s right-click menu provides these commands:

- **Undo** or **Redo** the last edit
- **Cut**, **Copy**, or **Paste** text
- **Toggle Bookmark** or go to the **Next Bookmark**
- **Display Line Numbers** or **Go To** a specified line number
- **Run to Cursor**
- **Locate Match Brace** characters

- Find a specified value by indicating various search parameters
- Select Format (Hex, Float, Unsigned Integer, Integer, Octal)

Editor Tab Mode

Editor Tab mode provides an alternative, tab-based user interface for managing multiple source files in editor windows. When this mode is enable from the **View** menu, a tab for each open source file appears at the bottom of the editor window. Click the tabs to switch between files.

Figure 2-18 shows an editor window with the **Editor Tab** option enabled.

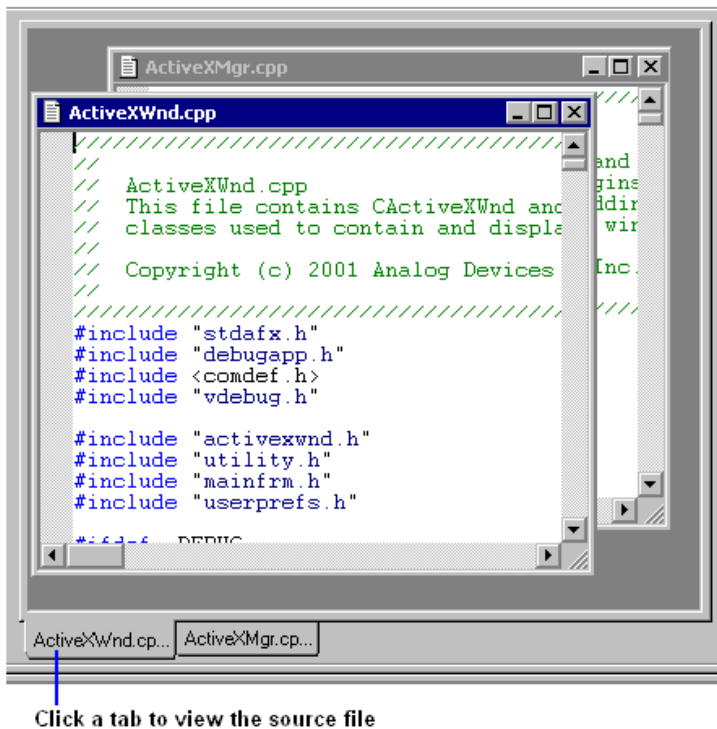


Figure 2-18. Editor Tab Mode Enabled

Output Window

The **Output** window does the following.

- Displays standard I/O text messages such as file load status and error messages
- Displays build status information for the current project build
- Provides access to errors in source files
- Acts as a scripting interface

The **Output** window shown in [Figure 2-19](#) contains build status information. Display the **Output** window by choosing **View** and **Output Window**.

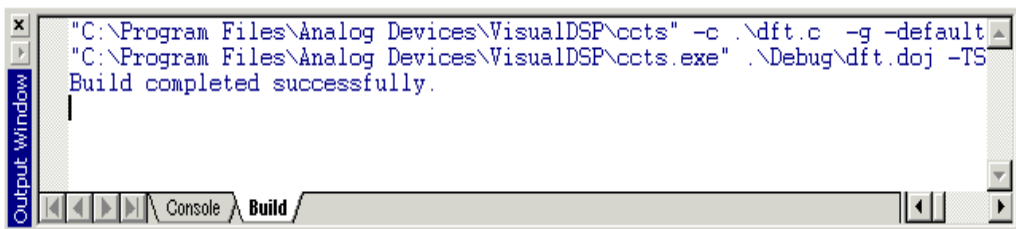


Figure 2-19. Build Status Information in the Output Window

Output Window Tabs

Clicking the **Output** window's two tabs, **Console** and **Build**, displays pages that provide different information and capabilities.

Build Page

The **Build** page (Figure 2-20) displays error messages generated during a build. Double-click on an error message to jump to the offending code in an editor window.

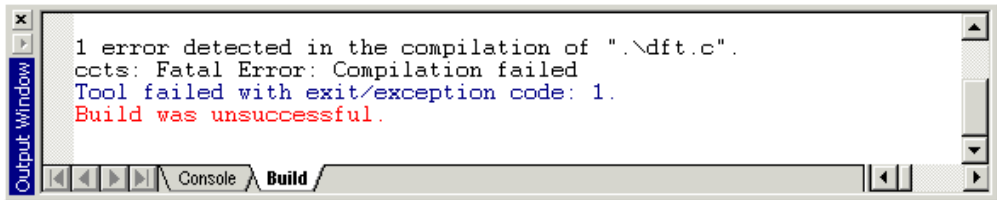


Figure 2-20. Error Messages in the Output Window

Scroll through error messages by choosing **Next Error** or **Prev Error** from the **Edit** menu.

By default, VisualDSP++ output is blue and tool output is black, but these colors can be changed in the **Preferences** dialog box.

Console Page

From the **Output** window's **Console** page (Figure 2-22 on page 2-42), you can:

- View VisualDSP++ or target status error messages
- View STDIO output from C/C++ programs
- View I/O (streams) messages
- Scroll through previous commands by pressing the keyboard's up arrow (↑) and down arrow (↓) keys
- Perform multiline selection, copy, paste, and clear
- Issue script commands and view script command output

VisualDSP++ Windows

- Auto-complete script commands
- Execute a previously issued script command by double-clicking on the command
- Enter multiline script commands by adding a backslash character (\) to the end of a statement
- Use bookmarks
- Toggle a bookmark by pressing **Ctrl+F2**
- Move to the next bookmark by pressing the keyboard's **F2** key

All text displayed on the **Console** page is also written to the VisualDSP++ log file.

Output Window Error Messages

The DSP code development tools that perform batch processing can produce error and warning messages when returning a result. These informational messages appear on the **Build** page in the **Output** window.

Every error is identified with a unique six-character code, such as pp0019, that is consistent from release to release. Error descriptions include an explanation of the condition that caused the error and a suggested remedy to fix the problem. Where applicable, error messages include the source file's name and the line number of the offending code.

Error Message Severity Hierarchy

Each error message has one or more severity levels.

Table 2-10. Error Message Severity Levels

Severity Level	Description
Fatal error	Identifies errors so severe that further processing of the input is suspended. Fatal errors are sometimes called catastrophic errors.
Error	Identifies problems that cause the tool to report a failure. An error might allow further processing of the input to permit the reporting of additional problems to be reported.
Warning	Identifies situations that do not prevent the tool from processing the input, but may indicate potential problems
Remark	Provides information of possible interest

You can change the severity level of an error marked “discretionary.” You cannot change the severity level of an error marked “non-discretionary.”

Syntax of Help for Error Messages

In Help, each error message can include several parts. The information displayed depends on the tool and the message.



To view all the details, you must view the error message in the Help system window. If you run a tool from a command-line interface (such as a **Command Prompt** window or **MS-DOS Prompt** window), the error message shows only the ID code, error text, and error location.

Table 2-11 describes the syntax for error message help.

Table 2-11. Syntax for Error Message Help

Part	Description
Identification Code	Six-character code, unique to the error. The first two characters identify the tool: • ar (archiver) • cc (compiler) • ea (assembler) • el (expert linker) • li (linker) • pp (preprocessor) • vc (VIDL compiler) • vu (VCSE)
Error Text	Text that appears after the identification code in the Output window
Description	Detailed description of the error
Severity	The degree of hardship imposed by the error. Some messages can take more than one severity level. You can change the severity level of an error marked “discretionary.” You cannot change the severity level of errors marked “non-discretionary.”
Recovery	Extra information, provided only if applicable
Example	Example code
How to Fix	The remedy for correcting the error
Related Information	Link(s) to more information

How to Promote, Demote, and Suppress Error Messages

You can change the severity level of an error marked “discretionary.” Refer to the tools documentation for command-line switches that override error message severity. The VisualDSP++ environment’s **Project Options** dialog box includes options that override severity.

A discretionary message can be promoted, demoted, or suppressed. For example, you might promote a remark or warning to an error, or you might decide to demote an error to a warning or remark.

For example, if a condition in the input crashes the tool, restrict the severity level of the problem to ensure that an error (instead of a fatal error) is reported.

Another way to suppress the reporting of an individual error message is to use pragmas in the input source via the tool’s command line. For more information about pragmas, refer to your processor’s VisualDSP++ 3.5 C/C++ Compiler and Library Manual.

The following examples demonstrate how you can promote, demote, and suppress messages. The source file `test.c` is being compiled.

```
#include <stdio.h>
int foo(void)
{
    printf("In foo\n"); // doesn't return a value
}

int main(void)
{
    int x; // no initial value
    printf("x = %d\n", x);
    return foo();
}
```

- Example 1: Compiling From the Command Line (Interface)

Compiling the `test.c` file yields these two warning messages:

```
$ cc21k -c test.c
"test.c", line 5: cc0117: {D} warning: non-void function
"foo" should return a value
}
^
"test.c", line 10: cc0549: {D} warning: variable "x" is used
before its value is set
printf("x = %d\n", x);
^
build completed successfully
```

Note that the compiler appended the letter `D` to each warning message (cc0117 and cc0549) to indicate that the message is discretionary.

- Example 2: Promoting Warnings to Errors

Typing `$ cc21k -c test.c -Werror 549` in a command window promotes one of the two warnings to an error.

```
"test.c", line 5: error cc0117: {D} warning: non-void
function "foo" should return a value
}
^
"test.c", line 10: error cc0549: {D} error: variable "x" is
used before its value is set
printf("x = %d\n", x);
^

1 error detected in the compilation of "test.c".
cc21k: Fatal Error: Compilation failed
```


- **Example 3: Demoting Messages to Remarks**

You can demote messages to remarks. By default, however, the compiler does not display anything less significant than a warning.

The `-Wremarks` flag in the following command outputs the two warnings plus additional remarks.

```
$ cc21k -c test.c -Wremarks
```

The `-Wremark 549,117` flag in the following command specifies that two specific messages be demoted to remarks. The command produces no output because all the messages are changed to remarks, which are not displayed.

```
$ cc21k -c test.c -Wremark 549,117
```

The following command changes the two warnings to remarks and then displays all seven remarks.

```
$ cc21k -c test.c -Wremark 549,117 -Wremarks
```

- **Example 4: Suppressing Messages**

The following command suppresses two specific warning messages. The command outputs five remarks, but the two warnings are not displayed even though the `-Wremarks` flag requests all the remarks.

```
$ cc21k -c test.c -Wsuppress 549,117 -Wremarks
```

How to Suppress the Reporting of Compiler Warnings and Remarks

You can suppress compiler remarks as well as compiler warnings.



You cannot suppress compiler warnings without also suppressing remarks.

VisualDSP++ Windows

You control the output of compiler warnings and remarks from the Project Options dialog box in VisualDSP++ or from the command line. Refer to your processor's compiler, assembler, and linker manuals for available flags (options).

From the **Compile** page (**Warning** category) of the **Project Options** dialog box, you can specify the options listed in [Table 2-12](#).

Table 2-12. Options Available From the Compile Page

Option	Purpose
Implicit function declarations	Warns on all implicit functions. This option corresponds to the compiler's <code>-fimplicit-declarations</code> command-line switch.
Functions not in-lined	Issues a warning when the compiler is unable to generate in-line code for a function that has the in-line keyword
Enable remarks	Issues remarks, which are diagnostic messages of a milder nature than warnings. This option corresponds to the compiler's <code>-wremarks</code> command-line switch.
Disable all warnings and remarks	Withholds warning messages. This option corresponds to the compiler's <code>-w</code> command-line switch.
Additional options	Enables you to enter more compiler options

How to View Error Message Details

Each DSP tool error message has associated explanatory text. View the information in the Help window by selecting the six-character error identifier (for example, `cc0251`) on the **Build** page and by pressing the **F1** key. A complete explanation of the error message appears in the Help window.

Log File

The VisualDSP++ log file contains all status and error messages written to the **Output** window's **Console** page.

Figure 2-21 shows a sample log file.

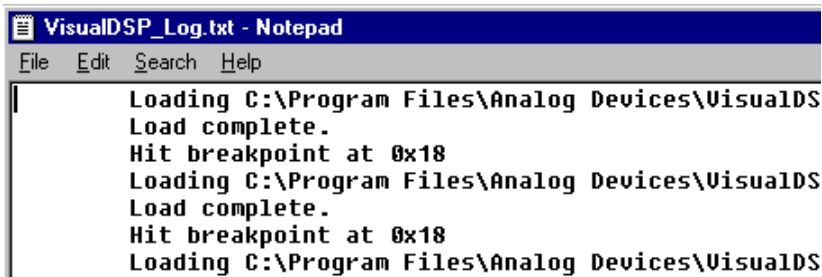


Figure 2-21. Portion of a Sample Log File



The file path specified in the log file assumes that you installed VisualDSP++ by accepting the default settings.

All sessions append to the log file. Occasionally, open the file and delete parts of it (or all of it) to conserve disk space.

Output Window Customization

You can specify preferences that:

- Configure **Output** window fonts and colors
- Enable command auto-completion

By default, the **Output** window resides at the bottom of the main application window. You can resize or move the **Output** window to a different portion of the screen by dragging it to the selected location. You can dock, hide, or float the window.

The **Output** window's **Console** page can interact with script engines. All script input and output is sent to the **Console** page, shown in [Figure 2-22](#).

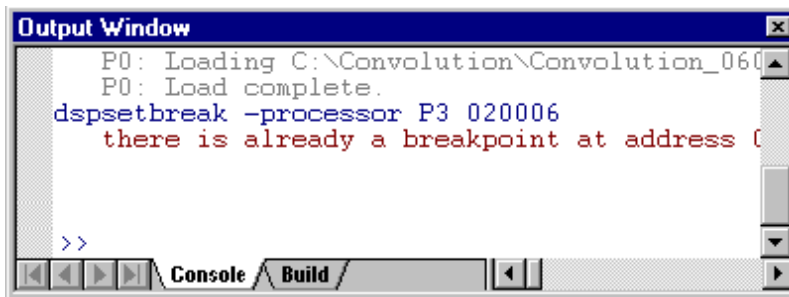


Figure 2-22. Messages in the Project Window's Console Page

These messages are saved to the log file `VisualDSP_Log.txt`, which is located in the installation's `Data` directory.

Right-Click Menu

The **Output** window's right-click menu is shown in [Figure 2-23](#).

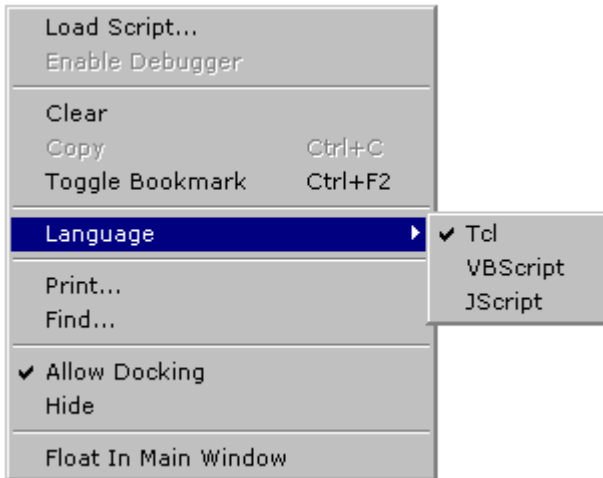


Figure 2-23. Output Window's Right-Click Menu

This menu enables you to:

- Load a script or enable the debugger
- Clear the text in the window or copy selected text
- Toggle bookmarks
- Choose a scripting language
- Print or find text in the window
- Dock, hide, or float the window. (To display the hidden window, choose **Output Window** from the **View** menu.)

Script Command Output

Scripts provide a powerful means of developing full-blown test applications of DSP systems. VisualDSP++ 3.5 includes a language-independent scripting host that uses the Microsoft ActiveX® script host framework. This scripting host lets you use multiple scripting languages that conform to the Microsoft ActiveX script engine.

The main benefit of calling scripts in these languages is that they have support for COM scripting, which allows access to the VisualDSP++ Automation API. VisualDSP++ supports the following Microsoft ActiveX script engines (languages):

- Visual Basic® (Scripting Edition)
- JScript®



The Tool Command Language (TCL) interpreter included with VisualDSP++ is not a Microsoft ActiveX script engine.

VisualDSP++ permits the use of other script engines (languages) that are not supported by Analog Devices technical support.

Script output is logged to `VisualDSP_log.txt` for viewing and analysis. By default, this file is located in the installation's `Data` directory.

In the **Output** window's **Console** view, you can:

- Issue script commands and view script command output

For more information about issuing script commands, refer to [“Extensive Scripting” on page A-34](#).

- Enable the Microsoft Script Debugger

Right-click in the **Output** window and choose **Enable Debugger**. The debugger steps through code, set breakpoints, and so on. Once enabled, the debugger stops on the first error encountered in the script.

Note that although most script engines (languages) support this option, some may not. Consult the script engine's documentation for further details on whether it supports the debugging interfaces within the Microsoft ActiveX script engine framework.

- Specify the scripting language

Right-click in the **Console** view and select a language from a list of scripting languages installed on your machine.

The name of the current scripting language appears in the status bar at the bottom of the VisualDSP++ main window, as shown in [Figure 2-24](#).

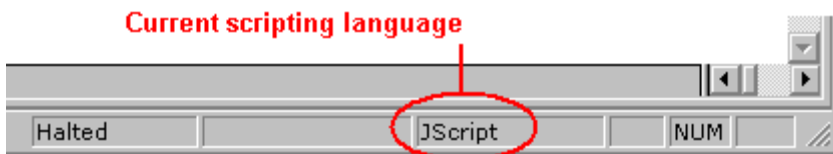


Figure 2-24. Scripting Language Displayed in Status Bar

- Load a script

You can load a script by selecting **Load Script** from the **File** menu, from the **Console** view's right-click menu, or from the editor window's right-click menu. The script loads and runs until the script finishes running or until you halt the script by choosing **Halt Script** from the **Debug** menu.

The **Console** view supports script command auto-completion if you enable this feature on the **General** page in the **Preferences** dialog box, accessed from the **Settings** menu.

The VisualDSP++ installation directory includes example scripts in the "Scripting Examples" folder located under the processor family name (for example, 21k) and the `Examples` folder.

Window Operations

Similar to many Windows applications, VisualDSP++ provides ways to adjust the view of the user interface.

Window Manipulation

The **Window** menu commands, shown in [Figure 2-25](#), enable you to manipulate your window display and update windows during program execution. Refer to your Windows documentation for more information.

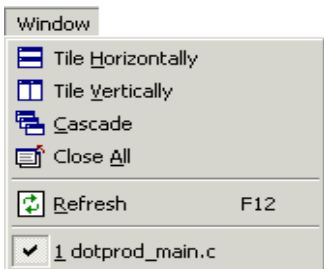


Figure 2-25. Window Menu Commands

Right-Click Menu Options

A menu appears when you right-click in a window or on its title bar. The menu options in [Table 2-13](#) affect window behavior.

Table 2-13. Window Right-Click Menu Commands

Option	Description
Allow Docking	Enables or disables docking
Close	Closes the window
Float in Main Window	Causes the window to become a normal MDI child window (like an editor window) and disables its docking ability

Scroll Bars and Resize Pull-Tab

Scroll bars appear along the right and bottom edges of the application or document window, as shown in [Figure 2-26](#).

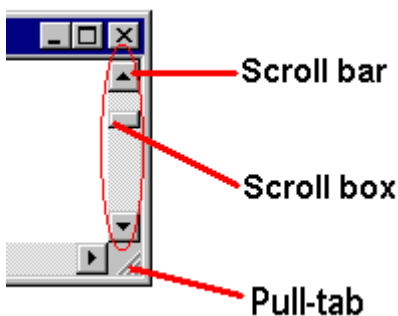


Figure 2-26. Scrolling to Move the Viewing Area

The scroll boxes inside the scroll bars indicate the vertical and horizontal location in the document. Use the mouse to scroll to other parts of the document.

When the application window is not maximized, the resize pull-tab appears in the lower-right corner of the window. Click and drag the pull-tab to resize the application window.

Windows: Docked vs. Floating

A window attached to the application's frame is referred to as a *docked window*.

You can detach a window from the main window and move it to another location anywhere on the desktop. A *floating window* stands alone, because it is not docked.

Window Operations

Depending on your needs, you can:

- Dock a window to the application's main window (frame)
- Float a window

A window's right-click menu provides commands to dock or float the window. The **Allow Docking** option and the **Float In Main Window** option are mutually exclusive.

Example of a Docked Window

The **Project** window shown in [Figure 2-27](#) is docked. (**Allow Docking** is selected.)

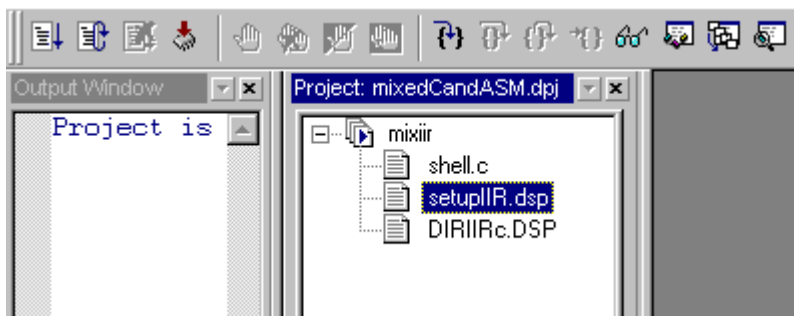


Figure 2-27. Example of a Docked Project Window

To prevent a window from docking, hold down the keyboard's **Ctrl** key while dragging the window to another position.

Examples of Floating Windows

The Project window in [Figure 2-28](#) is floating in the main window. (**Float In Main Window** is selected.)

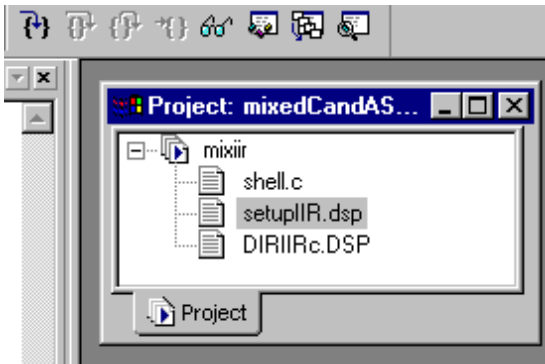


Figure 2-28. Project Window Floating in Main Window (1 of 2)

The Project window in [Figure 2-29](#) is also floating in the main window. (**Float In Main Window** is selected.)

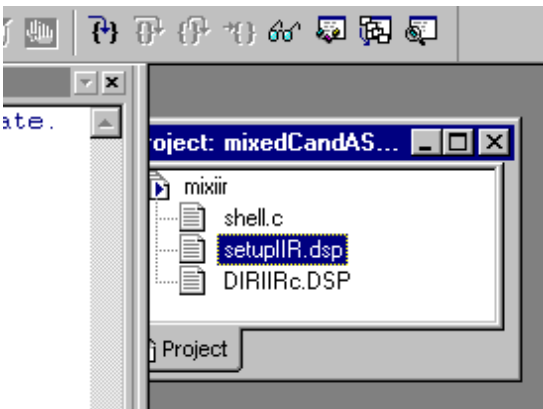


Figure 2-29. Project Window Floating in Main Window (2 of 2)

Window Operations

The Project window in [Figure 2-30](#) is floating, but not in the main window. (**Float In Main Window** is not selected.)

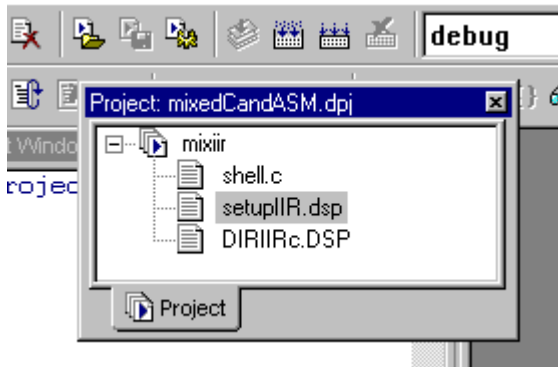


Figure 2-30. Project Window Floating But Not in Main Window

Window Position Rules

The following rules apply to window positions.

- Unless **Allow Docking** is disabled, a window must reside within the main window.
- An editor window cannot be docked to the main window.
- A window specified as an MDI child cannot be positioned over a docked window.
- Unless the **Output** window is floating in the main window, a window specified as an MDI child cannot be positioned over the **Output** window.

Standard Windows Buttons

The standard Windows buttons are located on the right side of the title bar, as shown in [Figure 2-31](#).



Figure 2-31. Title Bar Showing Standard Window Buttons

These buttons resize and close the window as described in [Table 2-14](#).

Table 2-14. Standard Windows Buttons

Button	Name – Purpose
	Minimize – reduces the window to its Windows icon
	Maximize – enlarges the window to fill the screen
	Restore – returns the window to its last non-minimized, non-maximized position after you maximize the window
	Close – closes the application window and exits the program

Debugging Windows

VisualDSP++ provides debugging windows to display DSP program operation and results. [Table 2-15](#) describes these windows.

Table 2-15. Debugging Windows

Window	Provides
Output	A Console page that displays standard I/O text messages such as file load status, and error messages and streams, and a Build page that displays build messages. You can interactively enter script commands and view script output.
Editor	Syntax coloring, context-sensitive expression evaluation, and status icons that indicate breakpoints, bookmarks, and the current PC position
Disassembly	Code in disassembled format. This window provides fill and dump capability.
Expressions	The means to enter an expression and see its value as you step through program execution
Trace	A history of processor activity during program execution, including buffer depth (instruction lines), cycle count, and instructions executed such as memory fetches, program memory writes, and data/memory transfers (SHARC processors only)
Locals	All local variables within a function. Use this window with Step or Halt commands to display variables as you move through your program.
Linear Profiling Results	(Simulation only) Samples of the target's PC register taken at every instruction cycle, which provides an accurate picture of where instructions were executed. Linear profiling is much slower than statistical profiling.
Statistical Profiling Results	(JTAG emulation only) Random samples of the target processor's program counter (PC) and a graphical display of the resulting samples, showing where the application spends time
Call Stack	A means of moving the call stack back to the previous debug context

Table 2-15. Debugging Windows (Cont'd)

Window	Provides
Register	Current values of registers. You can change register contents and change the number format.
Custom Registers	Current values of registers. Select the registers that you want to monitor.
Memory	A view of processor memory. Similar number format and edit features as register windows, plus fill and dump capability.
BTC Memory	A view of background telemetry channel contents in real time. The window displays the contents of the address that you want to see. (SHARC emulator sessions only)
Memory Map	The memory map of the selected processor
Plot	A graphical display of values from memory addresses. The window supports linear and FFT (real and complex) visualization modes and allows you to export an image to a file, the clipboard, or to a printer.
Multiprocessor	Current status of each processor in a multiprocessor system (emulator sessions only). This window allows you to define and manage groups of processors for synchronous multiprocessor commands.
Pipeline Viewer	Display of instructions in the pipeline and event details (TigerSHARC processors only)
Cache Viewer	Analysis of a DSP application's use of cache, which is helpful in optimizing DSP application performance
VDK State History	(VDK-enabled projects only) History buffer of threads and events
Target Load	(VDK-enabled projects only) Percent of time the target spent in the idle thread
VDK Status	(VDK-enabled projects only) At a program halt, thread state and status data
Image Viewer	A view of BMP, JPEG, PPM, or MPEG data from processor memory or from a file on your PC. You can edit, copy, print, or export image data.

Editor Windows

An editor window provides:

- Status icons
- Expression evaluation
- Two view formats (source mode or mixed mode)

Syntax Coloring

Specifying colors can help you locate information in the types of files listed in [Table 2-16](#).

Table 2-16. File Types That Support Syntax Coloring

File Type	File Extension
Assembly	.ASM
C	.C
Linker Description Files	.LDF
C++	.CPP
Header	.H
Script	Various extensions, such as .JS and .VBS

Right-Click Menu

The editor window's right-click menu provides the commands shown in [Figure 2-32](#).

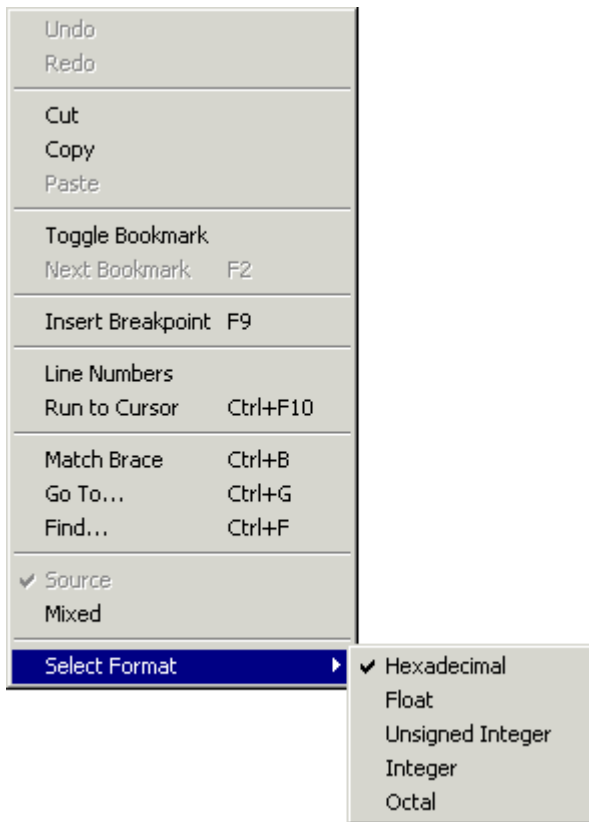


Figure 2-32. Editor Window's Right-Click Menu

i The available number formats listed under **Select Format** depend on the target processor. An additional command, **Source Script**, is available when you are editing a script.

Debugging Windows

Editor Window Symbols

The editor window's gutter (left margin) displays icons for breakpoints, bookmarks, and the current position of the program counter (PC).

[Table 2-17](#) describes these icons.

Table 2-17. Editor Window Symbols

Symbol	Indicates
	The current source line to be executed (PC location)
	An enabled breakpoint
	A disabled breakpoint
	A bookmark

Bookmarks

Bookmarks are pointers in editor windows. You bookmark a location to return to it quickly later.

Context-Sensitive Expression Evaluation

You can evaluate an expression in an editor window only if your .DXE program is loaded for debugging.

As you move the mouse pointer over a variable, with the pointer still on top of the variable, VisualDSP++ evaluates the variable. If the variable is in scope, the value appears in a tool tip window.

Viewing an Expression

An expression can be viewed in different ways. When the editor window is in mixed mode, view an expression by moving the pointer over a register in an assembly instruction. The register contents are displayed in a tool tip.

Highlighting an Expression

Highlight an expression in the editor window and then move the pointer on top of the highlighted expression to display its value in a tool tip.

Source Mode Versus Mixed Mode

You can specify an editor window's display format as either source mode or mixed mode.

Source Mode

Source mode, shown in [Figure 2-33](#), displays C code only.

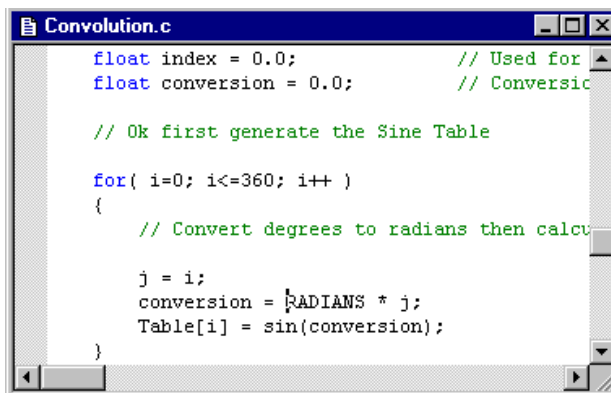


Figure 2-33. Editor Window in Source Mode Format

Debugging Windows

Mixed Mode

Mixed mode displays the assembled code after the line of the corresponding C code. The assembly code takes a specified color.

Note:

- The source file must be compiled with debugging information to to be viewed in mixed mode.
- The display of pipeline symbols can be enabled or disabled in mixed mode.

Figure 2-34 shows an example of the mixed mode format.

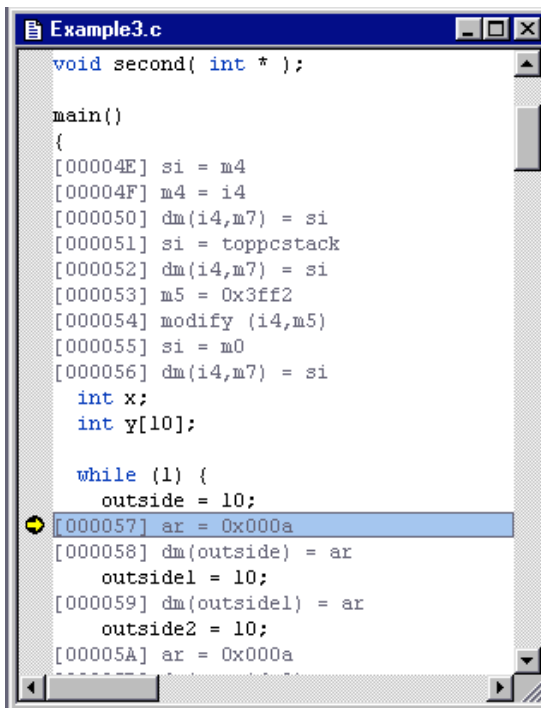


Figure 2-34. Editor Window in Mixed Mode

Disassembly Windows

By default, a **Disassembly** window appears when you open a new session. You can also open a **Disassembly** window by choosing **View, Debug Windows, and Disassembly**.

[Figure 2-35](#) and [Figure 2-36](#) show examples of **Disassembly** windows, one with and one without the address bar enabled.

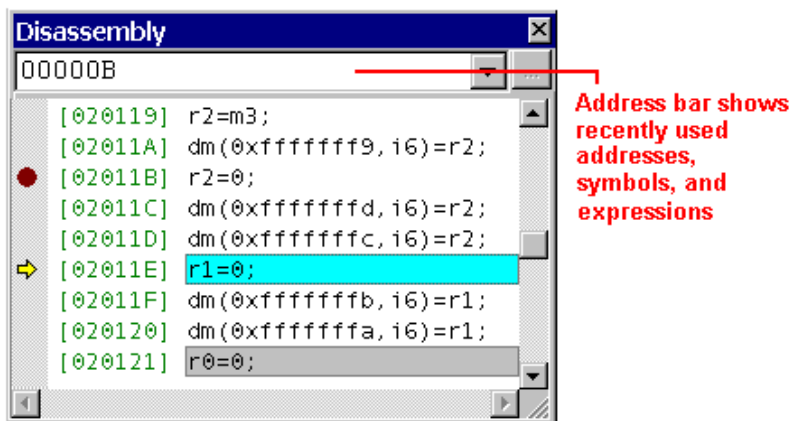


Figure 2-35. Disassembly Window With Address Bar

Debugging Windows

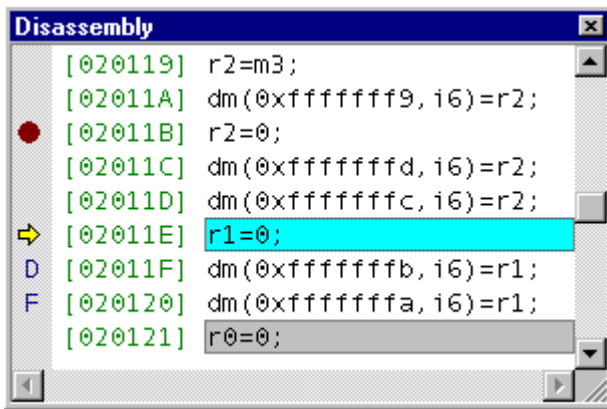


Figure 2-36. Disassembly Window Without Address Bar

Disassembly windows display code in disassembled form, which is useful for temporarily modifying the code to test a change or to view code when no source is available. The **Disassembly** window enables you to examine the assembly code generated by the C/C++ compiler. Choosing **View Source** from the **Disassembly** window's right-click menu enables you to view the C/C++ source code for the loaded file.

To make changes permanent, modify the code and rebuild the project.

Disassembly windows provide:

- Number format and edit features, similar to register windows
- Dump-and-fill capability
- Symbols at the far left of the window, denoting program execution stages and pipeline stages

You can enable and disable the display of pipeline symbols while in mixed mode (C/C++ and assembly).

- An optional address bar that enables you to navigate to an address, symbol, or expression. The address bar maintains a most recently used history of visited locations.

To display the address bar, right-click in a **Disassembly** window and choose **Address Bar**. A check mark next to this option on the right-click menu indicates that this feature is enabled.

By default, the current source line to be executed is highlighted by a light-blue horizontal bar, as shown in [Figure 2-37](#).

```
➡ [008005] jump fftrad2 (db);
```

Figure 2-37. Current Source Line in the Disassembly Window

You can configure the color of the current source line and other window items.

Other Disassembly Window Features

From the **Disassembly** window, you can perform the operations described in [Table 2-18](#).

Table 2-18. Disassembly Window Operations

To...	Place the mouse pointer over...
Move to a different address	An address field and double-click. Then select the address from the ensuing Go To dialog box. Note that you can also use the address bar to navigate to an address, symbol, or expression.
Insert or remove a breakpoint	An instruction and double-click
Toggle (enable or disable) a breakpoint	An instruction and right-click. Then choose the appropriate command from the ensuing menu.

Debugging Windows

Right-Click Menu

The Disassembly window's right-click menu provides access to the commands shown in [Figure 2-38](#).

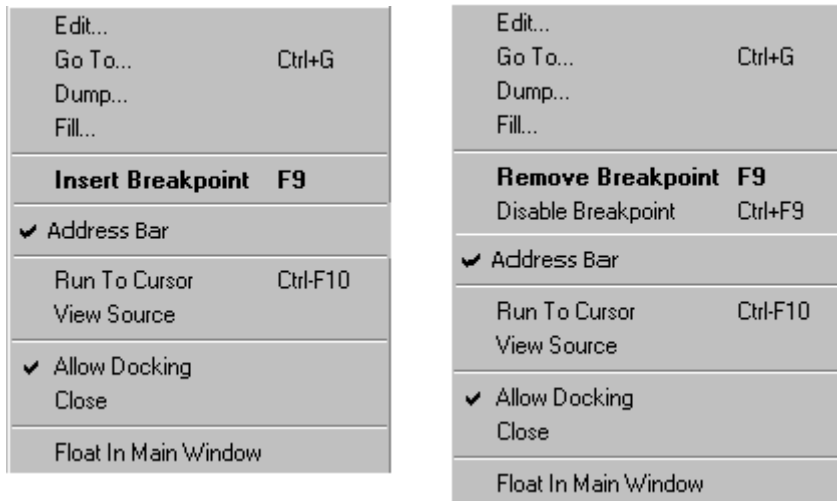


Figure 2-38. Disassembly Window Right-Click Menus

Disassembly Window Symbols

Symbols at the far left of the **Disassembly** window indicate program execution stages.

The display of pipeline stages is available only when:

- The selected session is connected to a SHARC simulator target.
- **Enable pipeline display** is selected on the **General** tab page of the **Preferences** dialog box, available from the **Settings** menu.

The symbols displayed at the left of the **Disassembly** window are shown in [Table 2-19](#). Stages P and A appear only for ADSP-2136x targets.

Table 2-19. Disassembly Window Symbols (SHARC Simulation Only)

Symbol	Description
	(Gray arrow) The current instruction is being aborted due to a branch or jump instruction.
	A breakpoint is enabled.
	A breakpoint is disabled.
F	This instruction is currently in the Fetch Address stage of the pipeline.
P	This instruction is currently in the Predecode (Fetch2) stage of the pipeline.
D	This instruction is currently in the Instruction Decode stage of the pipeline.
A	This instruction is currently in the Address Decode stage of the pipeline.
	(Yellow arrow) This instruction is currently in the Execute stage of the pipeline.

Expressions Window

The **Expressions** window (Figure 2-39) lets you enter an expression to evaluate in your program. Evaluations are based on the current debug context. Open this window by choosing **View, Debug Windows, and Expressions**.

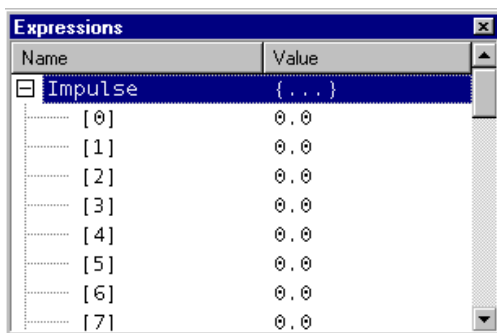


Figure 2-39. Expressions Window

Because of the way registers are saved and restored on the stack, the register value on which the expression relies may be incorrect if you change VisualDSP++'s context from the **Call Stack** window.

Right-Click Menu

The **Expressions** window's right-click menu ([Figure 2-40](#)) includes commands for changing the display's number format.

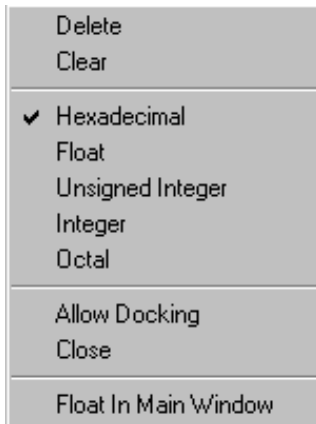


Figure 2-40. Expressions Window Right-Click Menu

Expressions in an Expression Window

Table 2-20 describes the types of expressions that you can enter in an Expressions window.

Table 2-20. Types of Expressions Allowed in an Expressions Window

Expression	Description
Memory address	Precede memory identifiers with a \$ sign, for example: \$dm(0xF0000000)
Register expression	Precede register names with a \$ sign, for example: \$r0, \$r1, \$ipend, \$po, or \$imask
C/C++ statements	Use standard C/C++ arithmetic and logical operators.
Multiprocessor expression (emulator sessions only)	Expressions can be evaluated on a particular processor by using the format: <pre>@processor_name(expression)</pre> where processor_name is the name of one of your MP processors, and expression is the expression that you want to evaluate on that processor. For example, on an MP system with two processors (master and slave), this expression evaluates the PC register on master:@master(\$PC)

The **Expressions** window displays the current value of each expression as you step through your program.

The evaluation of expressions is based on the current debug context. For example, if you enter expression “a” and a global variable “a” exists, you see its value. If you then step into a function that has local variable “a”, you see the local value until the debug context leaves the function. When a variable goes out of context, a string displays next to the variable to inform you that the variable is out of context.

The expressions described above are C expressions. The current syntax also allows you to use registers in expressions. For example, the following is a valid expression.

```
$R0 + $I0
```

Register expressions and C expressions can be mixed in an expression.

Register expressions follow these rules:

- Precede register names with a \$ character.
- Register names can be uppercase or lowercase characters.
- Registers have no context. A register expression always evaluates to the current value of the register.

Trace Windows

Perform a trace (also called an execution trace or a program trace) to analyze the run-time behavior of your DSP program, to enable I/O capabilities, and to simulate source-to-target data streaming. [Figure 2-41](#) shows data displayed in a **Trace** window.

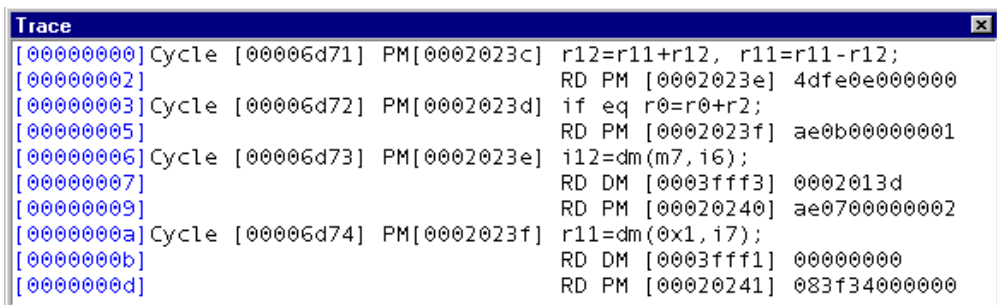


Figure 2-41. Example of Data in a Trace Window

Debugging Windows

The **Trace** window displays:

- Buffer depth (**Custom** in the **Trace Buffer Depth** dialog box)
- The clock cycle when the instruction occurred
- The address of the instruction executed
- The disassembled instruction

Memory results have the following fields.

- Access type (RD or WR)
- Memory type (PM or DM)
- The address, in brackets ([])
- The data value written or read



TigerSHARC processors do not support traces.

Locals Window

The **Locals** window displays the value of local variables within a function, as shown in [Figure 2-42](#). Open this window from the **View** menu by choosing **Debug Windows** and **Locals**.

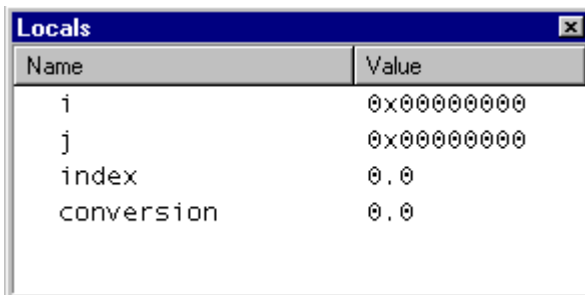


Figure 2-42. Locals Window

Use this window with a **Step** or **Halt** command to display the current value of variables when moving through your program.

Complex variables, C structures, and C++ classes appear with a plus  sign. Click on the plus sign to display all variable information.

The window's right-click menu provides the commands shown in [Figure 2-43](#).

Debugging Windows

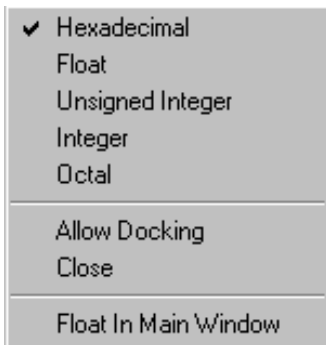


Figure 2-43. Locals Window Right-Click Menu

Statistical/Linear Profiling Results Window

To open a profiling results window, choose **Tools, Linear Profiling or Statistical Profiling, and New**. Depending on the target, the window's title is **Statistical Profiling Results** or **Linear Profiling Results**. The window comprises two panes, as shown in [Figure 2-44](#).

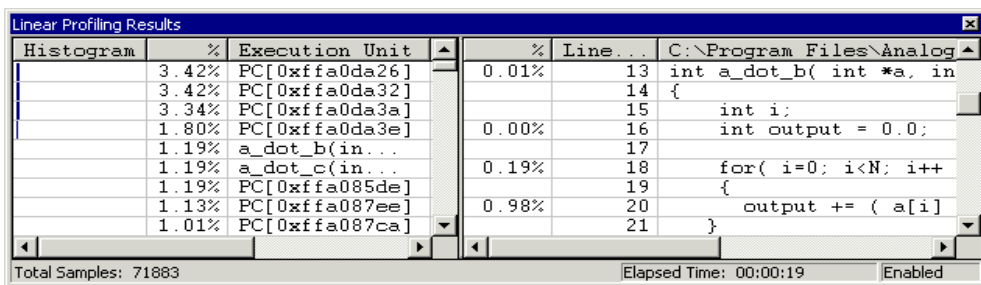


Figure 2-44. Example of a Linear Profiling Results Window

Window Components

The window, which comprises two panes and a status bar, provides a right-click menu for performing various window functions.

Left Pane

The window's left pane displays a list of the executed functions, assembly source lines, and PCs (with no debug information). The time that each item spends on execution appears as a histogram and as a percent. The order of the items in the display is determined by the percentage of global execution time for each item.

The left pane includes the information described in [Table 2-21](#).

Table 2-21. Left Pane Information

Column	Displays	Purpose
Histogram	Horizontal bars	Graphically represents the execution percentage
% -or- Count	A percent with two decimal places, for example: 15.01% -or- a number	Displays execution in percent or as a count. Right-click and choose View Execution Percent to view execution as a percent, or choose View Sample Count to view the PC sample count.
Execution Unit	Functions, assembly source lines, and PCs for which no debug information exists	These items are sorted by the percentage of global execution time that each item took to execute. The highest percentage items appear at the top of the list

Double-clicking on a line with a function or assembly source line in the left pane displays the corresponding source file in the right pane. The top of the function or assembly source line is shown in the source file. If you double-click on a PC address with no debug information, the **Disassembly** window opens to that address.

Debugging Windows

Right Pane

The right pane includes the information described in [Table 2-22](#).

Table 2-22. Information in the Right Pane

Column	Displays
%	Execution percent in text format with two decimal places, for example: 1.03% -or- the PC sample count for each source line
Line	Line numbers of the source file
File	Entire source file. Each source line occupies one line in the grid control.

Status Bar

The status bar at the bottom of the window indicates the total number of collected PC samples, the total elapsed time, and whether statistical profiling is enabled.

Right-Click Menu

The **Statistical Profiling Results** and **Linear Profiling Results** windows provide a right-click menu. The menu commands depend on the context (whether you right-click in the left pane or right pane) and the current settings.

[Table 2-23](#) describes the menu commands.

Table 2-23. Profiling Results Window Right-Click Menu Commands

Command	Description
Enable	Enables or disables profiling
Load Profile	Opens the Select a Statistical /Linear Profile to Load dialog box from which you can load profile data saved from a previous run
Save Profile	Saves the current run's data to a file
Concatenate Profile	Merges profiling data stored from a previous run with current data
Clear Profile	Clears statistics saved from a previous run
View Execution Percent	Displays the execution percent in each execution unit or source line. This value is the sample count for that execution unit divided by the total number of samples.
View Sample Count	Displays the sample count for that execution unit
Mixed -or- Source	Sets the display mode for C/C++ source lines from the right pane only. Choose Mixed to display both C/C++ source lines and assembly lines. C/C++ source lines appear in black type, and assembly lines appear in gray. Profiling data appears for each assembly line. Choose Source to display only the C/C++ source lines.
Properties	Opens the Profile Window Properties dialog box, where you can view or change window settings. When performing linear profiling, you can select display options such as cache hits, cache misses, execution count, reads, and writes.

Debugging Windows

Window Operations

You can select various options for the **Statistical/Linear Profiling Results** window and perform various window operations.

Changing the Window View

After you specify properties for the **Statistical/Linear Profiling Results** window and enable profiling, the profiler collects data when you run a program. Depending on the filtering options selected, the window's **Execution Unit** column displays:

- Function names (such as `main`)
- Single addresses, for example, `PC(0x2000)`
- Address ranges, for example, `[2000-2050]`

 Single addresses and address ranges are in hexadecimal format. The “0x” notation, however, appears beside single addresses only.

Displaying a Source File

Double-clicking on a function name in the **Execution Unit** column not only displays the source of the function in the right pane but also the profiling data for each line of the function. [Figure 2-45](#) shows an example of code displayed for a function.

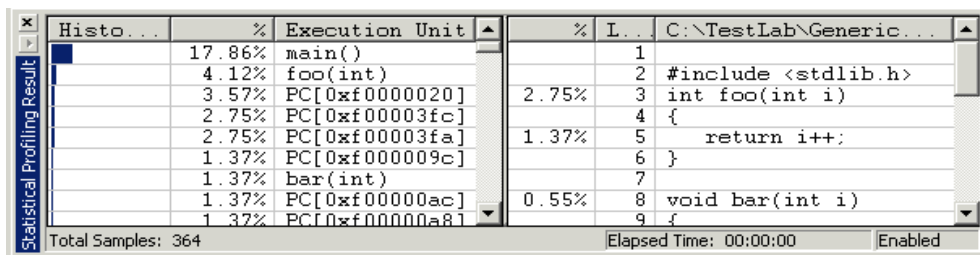


Figure 2-45. Code Displayed for a Function

Displaying Functions in Libraries

The profiling window enables you to display functions in libraries, as shown in [Figure 2-46](#).

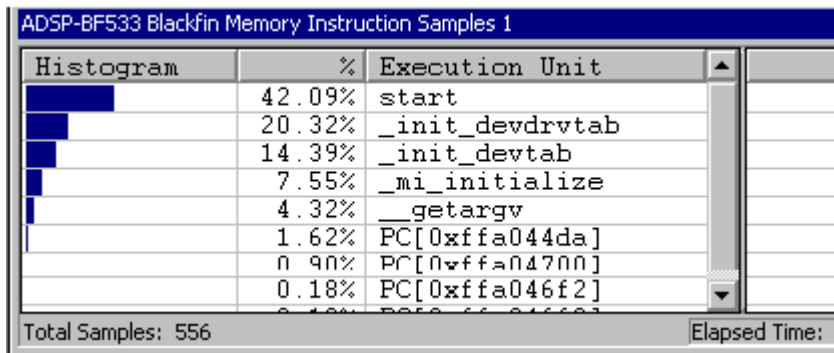


Figure 2-46. Profiling Window Showing Library Functions

To use this feature, right-click in the profiling window and choose **Properties** to open the **Profile Window Properties** dialog box. Next, click the **Filter** tab, select **C/C++ functions**, and click the **Add** button to open the **Add Functions** dialog box. Then select the **Show all functions** option.

Working With Ranges

Clicking on the icon in an address range expands or contracts the list of functions within that address range.

When expanded, the list of functions appears and profiling data appear immediately after the address range.

Switching Display Modes

The right-click menu’s **Mixed** and **Source** commands simplify switching between two views. [Figure 2-47](#) shows the source mode view and [Figure 2-48](#) shows the mixed mode view.

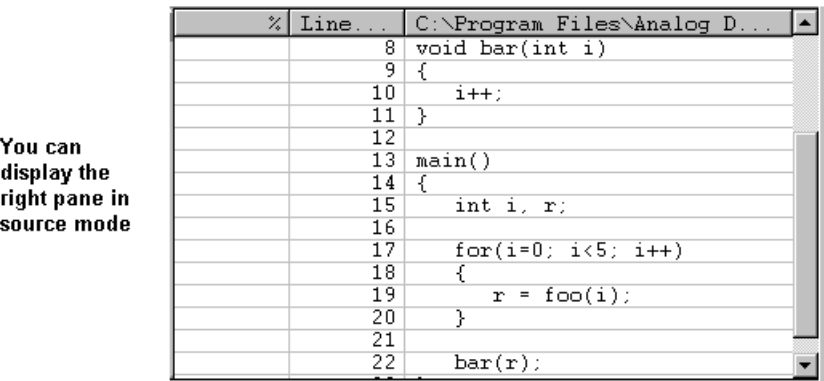


Figure 2-47. Source Mode View

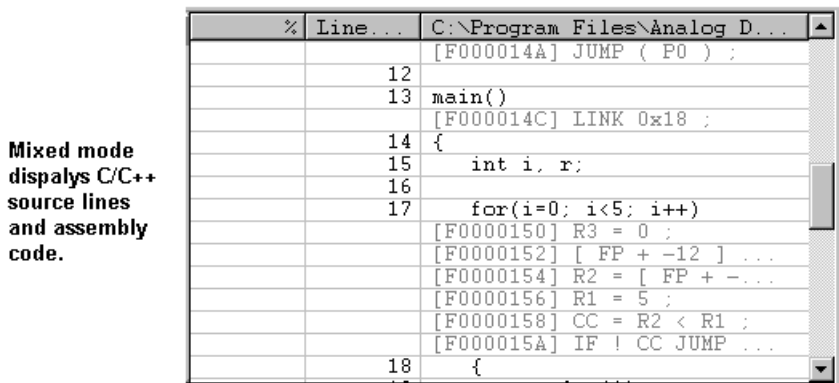


Figure 2-48. Mixed Mode View

When you view the window in mixed mode, profiling data for each assembly line is displayed, as shown in [Figure 2-49](#). Mixed mode displays profiling statistics for individual assembly instructions.

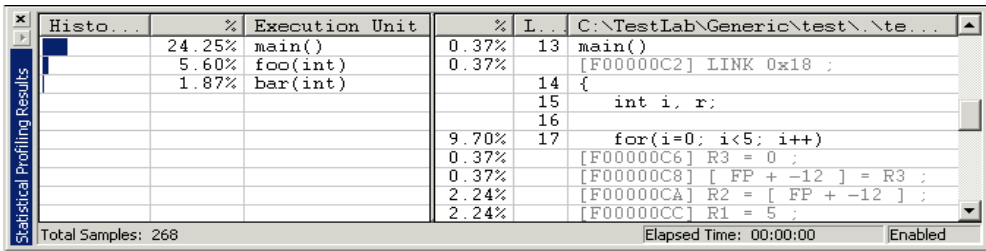


Figure 2-49. Profiling Data for Each Assembly Line (Mixed Mode)

Debugging Windows

Filtering PC Samples With No Debug Information

Since you spend most of your time building a “debug version” of your code, eliminate non-debug code, such as C run-time library initialization code.

The profiling results in [Figure 2-50](#) show where a lot of time is spent before filtering.

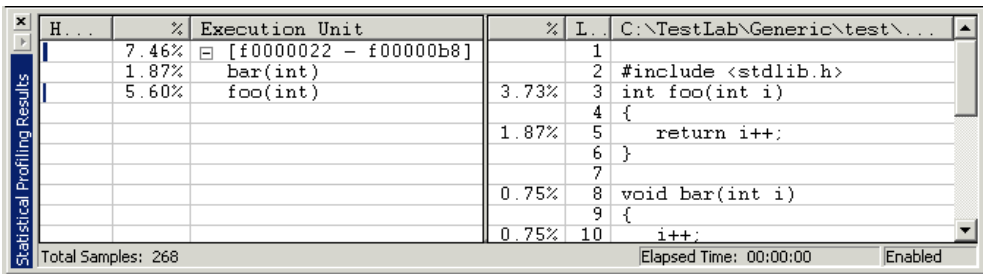


Figure 2-50. Profiling Results Before Filtering

The profiling results after filtering ([Figure 2-51](#)) reflect the difference.

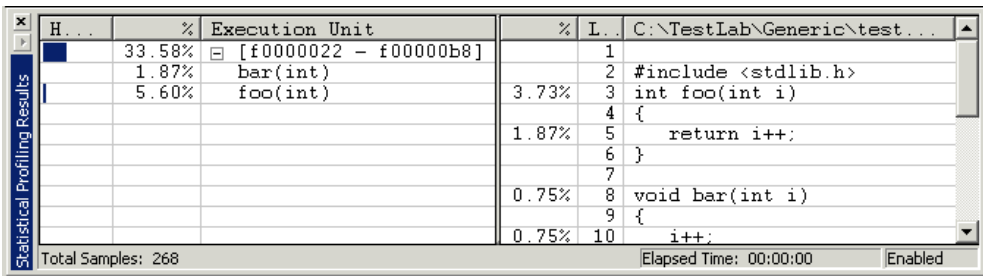


Figure 2-51. Profiling Results After Filtering

Call Stack Window

The **Call Stack** window (Figure 2-52) enables you to double-click on a stack location to move the call stack back to a previous debug context. Open this window by choosing **View, Debug Windows, and Call Stack**.

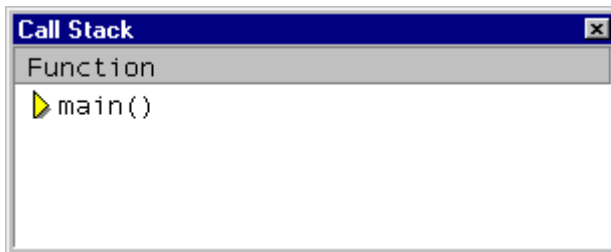


Figure 2-52. Call Stack Window



This window functions with C/C++ code only.

Use this window to analyze the state of parent functions when erroneous data is being passed to the currently executing function and to see the context from which the current function is being called. You can walk up the call stack and view local variables in different scopes.

Memory Windows

Memory windows let you:

- View and edit memory contents
- Display the address of a value. Move the mouse pointer over the value, and hold down the keyboard's **Ctrl** key.
- Lock the number of columns currently displayed. This action resizes the window horizontally without altering the display.
- Track one expression

Debugging Windows

You open memory windows from the **Memory** menu.

Memory windows provide:

- Number format and edit features
- Fill-and-dump capability
- An optional address bar for fast navigation to recently used addresses, symbols, or expressions

To display the address bar, right-click in a memory window and choose **Address Bar**. A check mark next to this option on the right-click menu indicates that this feature is enabled.

Memory Number Formats

The memory windows in Figures 2-53, 2-54, 2-55, and 2-56 show examples of different memory number formats.

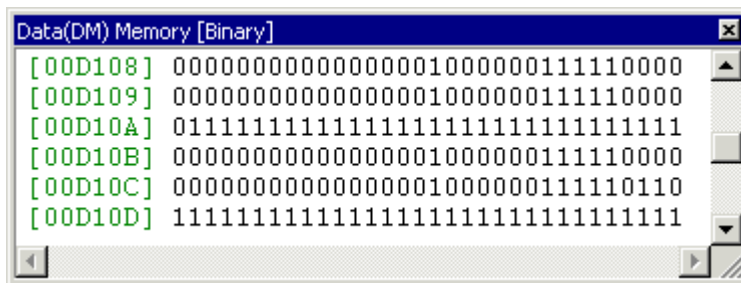


Figure 2-53. SHARC Memory in Binary Format

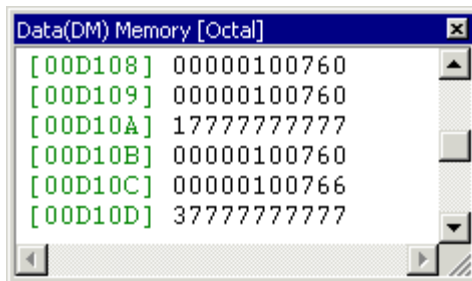


Figure 2-54. SHARC Memory in Octal Format

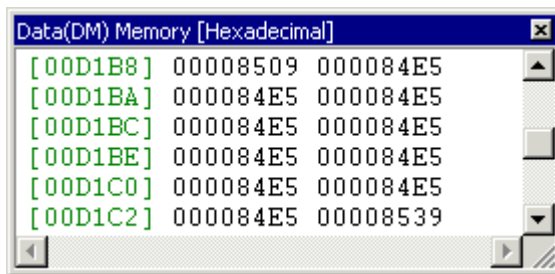


Figure 2-55. SHARC Memory in Hexadecimal Format

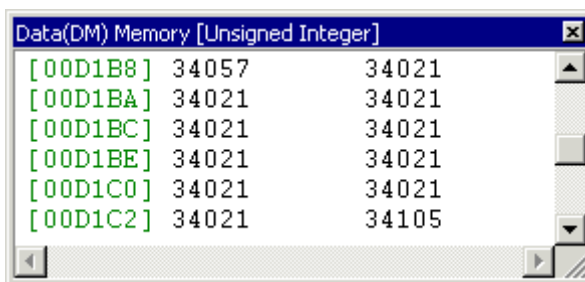


Figure 2-56. SHARC Memory Window in Unsigned Integer Format

Right-Click Menu

Memory windows provide a right-click menu. The **Select Format** command enables you to change the number format of the display. [Figure 2-57](#) shows the formats available for SHARC processors.

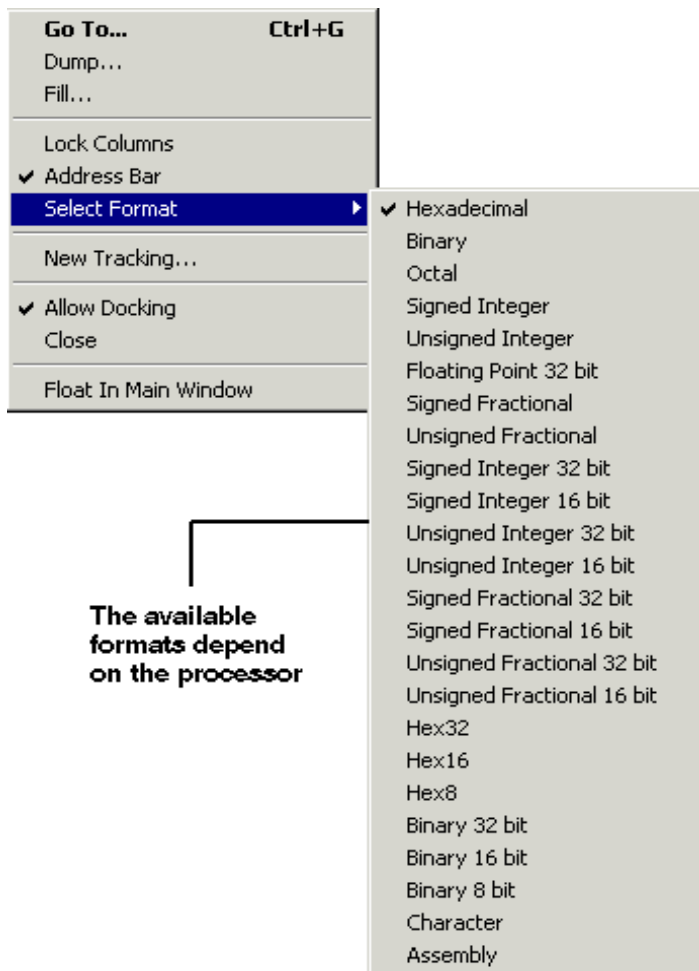


Figure 2-57. Memory Window Formats for SHARC Processors

Figure 2-58 shows the formats available for TigerSHARC processors.

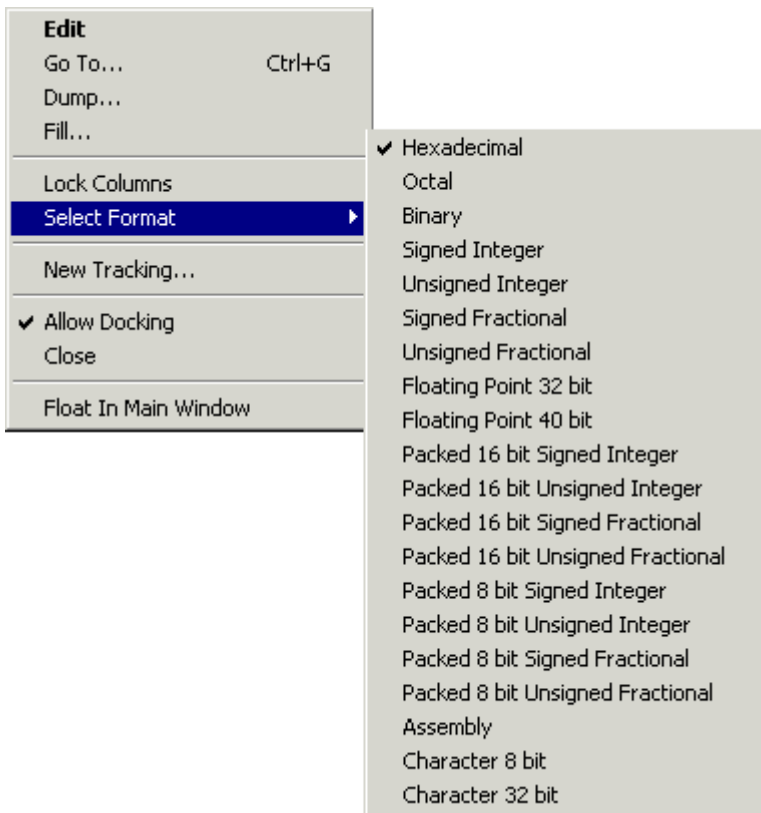


Figure 2-58. Memory Window Formats for TigerSHARC Processors

Expression Tracking in a Memory Window

While you step through your code, a memory window configured for expression tracking shows the memory at the address specified by the expression.

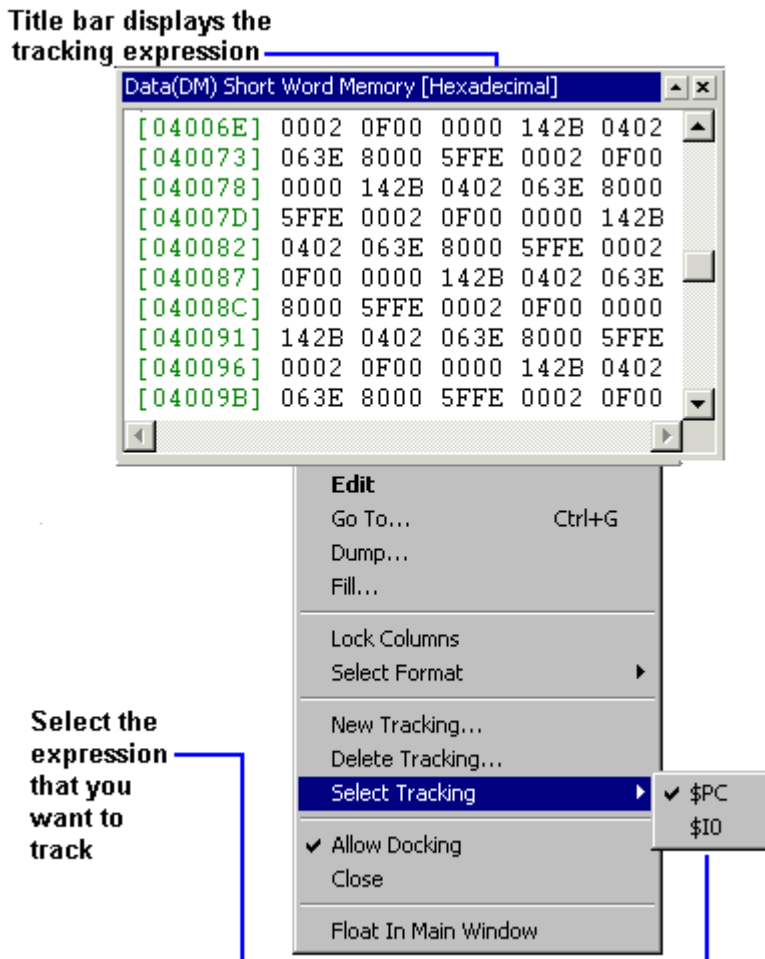


Figure 2-59. Expression Tracking in a Memory Window

Every time the target halts, the tracking expression is evaluated and the memory window jumps to that address. For example, if “\$PC” is used as the tracking expression, the memory window behaves like the **Disassembly** window.

Note:

- In a memory window, several expressions for tracking can be configured.
- In a memory window, only one expression at a time can be tracked.
- The active expression appears in the memory window’s title bar.
- The memory window’s right-click menu displays a list of configured expressions, and you can select one of them for tracking.
- To track multiple expressions, open multiple memory windows and track one expression per window.

Memory Window Display Customization

You can specify the colors used for symbols, data, address values modified values, and undefined memory regions. You can also adjust the width of the window to display a particular number of data columns. For example, the memory window in [Figure 2-60](#) is sized to display five columns.

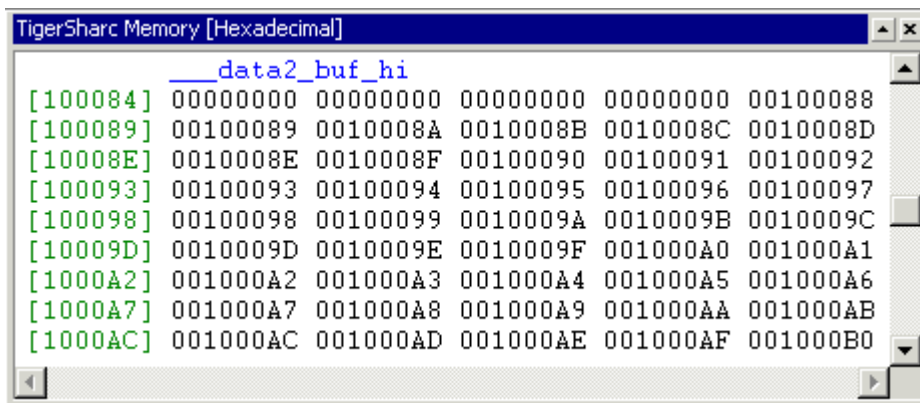


Figure 2-60. Memory Window (Five-Column Display)

i The commands displayed from the menu bar's **Memory** menu and the memory window's right-click menu depend on the processor that you are debugging.

Background Telemetry Channel (BTC) Window

Background telemetry channel (BTC) enables VisualDSP++ and a processor to exchange data via the JTAG interface while the processor is executing. Before BTC, all communication between VisualDSP++ and a processor took place when the processor was in a halted state.

i BTC is supported only in SHARC emulator sessions. For information about using BTC, see the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors*.

BTC Definitions in Your Program

Background telemetry channels are defined on a per program (.DXX) basis. The channels are defined when a specific program is loaded onto a processor. Define channels in your program by using simple macros.

The following example code shows channel definitions.

```
#include "btc.h"

.section/DM seg_dmda; // for ADSP-2126x processors

BTC_MAP_BEGIN
    BTC_MAP_ENTRY ('Channel0', 0xf0001000, 0x00100)
    BTC_MAP_ENTRY ('Channel1', 0xf0002000, 0x01000)
    BTC_MAP_ENTRY ('Channel2', 0xf0003000, 0x10000)
BTC_MAP_END
```

The first step in defining channels in a program is to include the BTC macros by using the `#include btc.h` statement. Then each channel is defined with the macros. The definitions begin with `BTC_MAP_BEGIN`, which marks the beginning of the BTC map. Next, each individual channel is defined with the `BTC_MAP_ENTRY` macro, which takes the parameters described in [Table 2-24](#).

Table 2-24. Parameters for the `BTC_MAP_ENTRY_ASM` Macro

Parameter	Description
Name	Name of the channel (32 characters max)
Starting address	Starting address of the channel in memory
Length	Length of the channel in 32-bit words for ADSP-2126x processors

Once the channels are defined, end the BTC map by using the `BTC_MAP_END` macro.

How to Enable BTC on ADSP-2126x Processors

After the channel definitions are added, the BTC must be initialized with a call to the `_btc_init` function during the application's start-up code.

After initialization, BTC commands from the host are processed via the low priority emulator interrupt (EMULI). A vector to the interrupt service routine must first be installed.

In assembly language, the vector can be installed with a jump to `_btc_isr` instruction placed at the `emuli` vector location:

```
jump _btc_isr;
```

After the interrupt vector is installed, the interrupt itself must be enabled with the following code.

```
// setup imask
ustatl = imask;
bit set ustatl EMULI;
imask = ustatl;

// enable interrupts
ustatl = model;
bit set ustatl IRPTEN;

model = ustatl;
```

In C/C++, the vector can be installed with the `interrupt` function as follows.

```
interrupt(SIG_EMUL,    btc_isr);
```

In C/C++, the `interrupt` function enables the interrupt for you.

After adding code to initialize BTC and enable the BTC interrupt, you must link with the BTC library (`libbtc26x.dlb` for assembly applications or `libcbtc26x.dlb` for C/C++ applications). This library contains the initialization function, interrupt service routine, and other functions that permit data transfer over the BTC.

BTC Priority

On the SHARC ADSP-2126x processor, BTC data transfer is handled through the low priority emulator interrupt (`EMULI`). Since the priority of this interrupt is fixed, the priority of BTC is also fixed.

The priority of the BTC can impact the response time from when the host requests data and the processor responds. Once the processor begins to service the request, interrupts can still be serviced by the processor. BTC performance can be affected by the frequency of system interrupts.

Debugging Windows

Examples

The **BTC Memory** window lets you view background telemetry channel contents in real time. The window displays the contents of the address that you want to see. You can change the window's view to meet your needs.

Open this window by choosing **View, Debug Windows, and BTC Memory**.

The view in [Figure 2-61](#) shows the contents of a specified channel only (for example, Channel1).

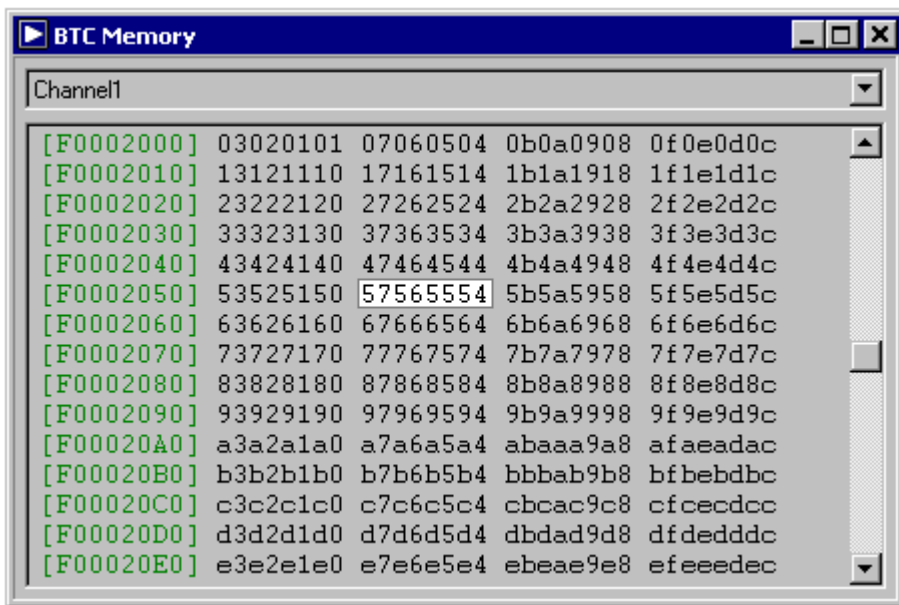


Figure 2-61. Viewing Contents of a Specified Channel Only

The view in [Figure 2-62](#) shows the list of currently defined channels and the contents of the selected channel.

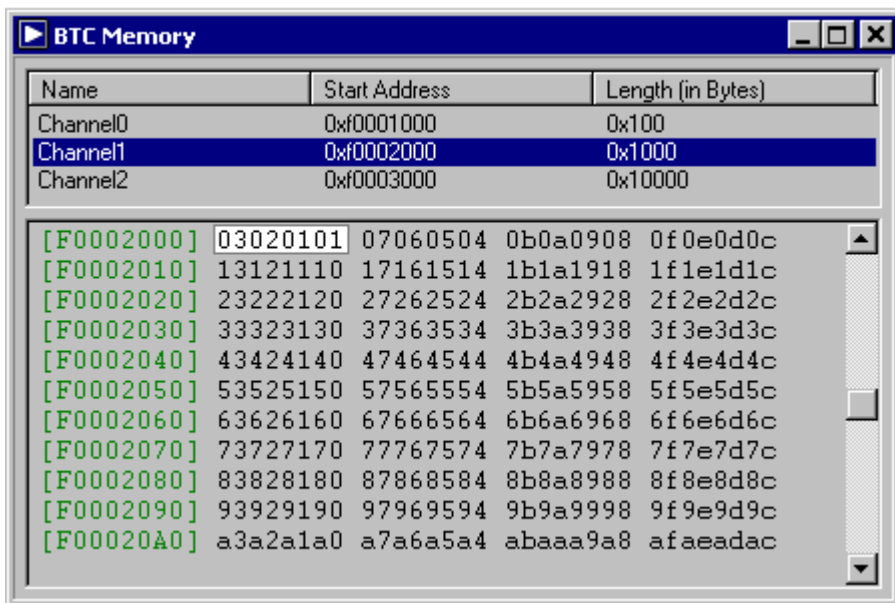


Figure 2-62. Viewing Defined Channels and Contents of a Selected Channel

Right-Click Menu

Table 2-25 describes the **BTC Memory** window's right-click menu.

Table 2-25. BTC Memory Window's Right-Click Menu

Command	Purpose
Go To	Opens the Go To dialog box, in which you specify an address. The specified address appears in the top-left corner of the display. The address must be within the range defined for the channel currently being displayed. Tip: Double-clicking in the address column also opens the Go To dialog box.
Show Map or Hide Map	Shows or hides a more informative map display of all the current channel definitions Show Map displays a channel list. Double-click a channel to display its contents in the lower portion of the window. Hide Map removes the list of channels. The selected channel remains in the display.
Lock Columns	Locks or unlocks the number of columns currently displayed in the window
Select Format	Specifies how to display data in the window. Choices include double words (32 bits), words (16 bits), and bytes (8 bits).
Refresh Rate	Specifies the refresh rate, which is used when Auto Refresh is chosen. The display is updated at the selected interval.
Auto Refresh	Enables the window to refresh itself at given intervals. The rate is specified by Refresh Rate. Auto Refresh mode is valid only while the processor is running.
Channel Timeout	Specifies the length of time to wait for any single response from the BTC. If the timeout value is exceeded, the current transaction ends.

Memory Map Windows

The **Memory Map** window (Figure 2-56) displays the memory map for the selected processor. Open this window by choosing **Memory** and **Memory Map**.

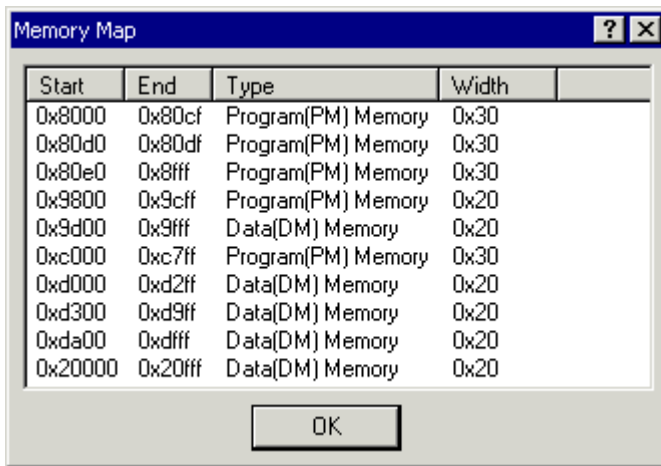


Figure 2-63. Memory Map Window

If no DSP program is loaded into the processor, the memory map displays all available memory in the processor.

If a program is loaded, the memory map is the map defined in the memory section of the program's .LDF file.

For each portion of memory, the window displays the start address, end address, and width.

Register Windows

Depending on your processor, you have access to various register windows from the **Register** menu.

Figure 2-64 shows the **Register** menu tree for SHARC processors.

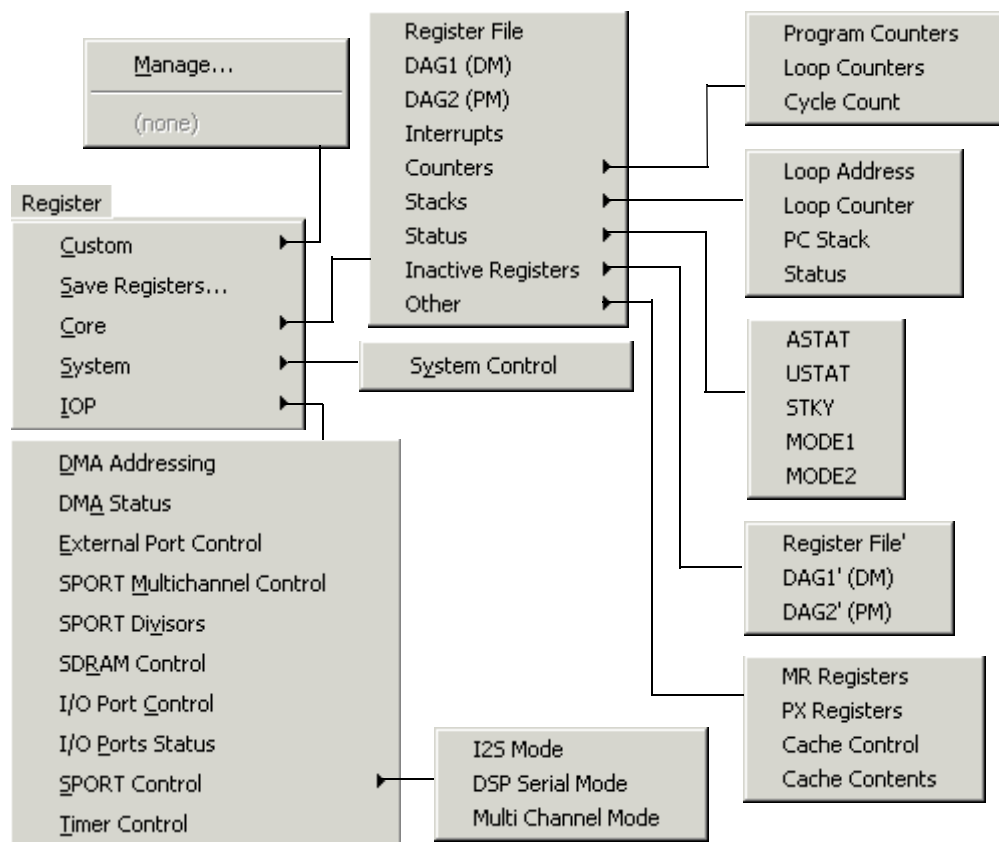


Figure 2-64. Register Windows Available for SHARC Processors

Figure 2-65 shows the **Register** menu tree for TigerSHARC processors.

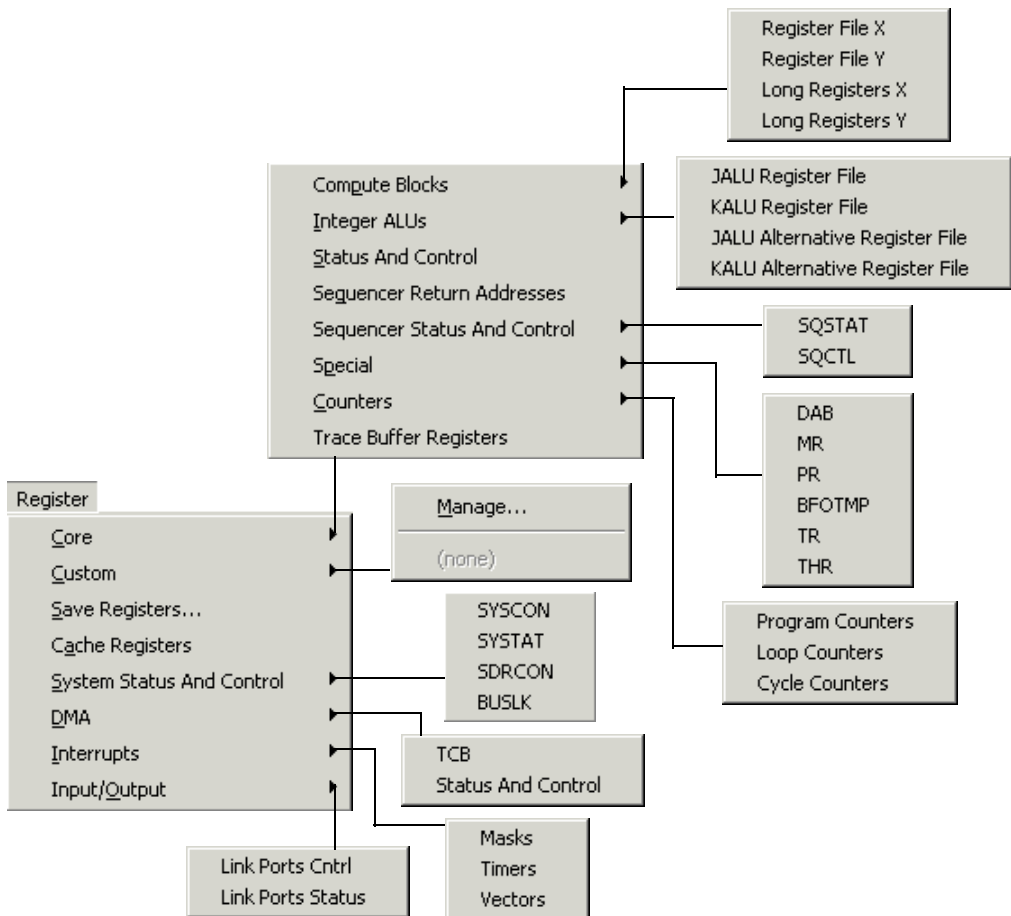


Figure 2-65. Register Windows Available for TigerSHARC Processors

Debugging Windows

Figure 2-66 shows an example of a data register file in a register window.

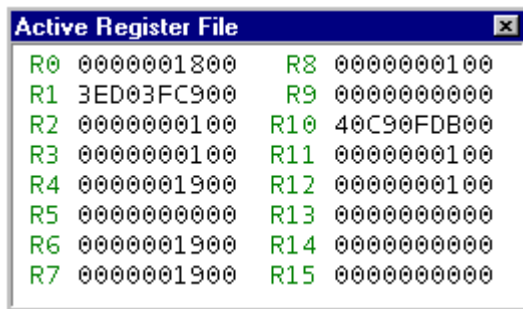


Figure 2-66. Register Window

A register window enables you to:

- View and change register contents
- Change the presentation (number format)

Register window number formats include standard formats, such as hexadecimal, octal, and binary. Depending on the processor, other formats are available.

You can change a register's data directly from a register window. The modified register content is used during program execution. Edits to data do not affect your source files. To make changes permanent, edit the source file and rebuild your project.

Stack Windows

Depending on your processor, you have access to various stack windows, including:

- PC Stack
- Counter Stack
- Loop Stack
- Status Stack

For more information about your processor's stack windows, consult the online Help.

Custom Registers Window

While debugging, you can configure and display a **Custom Registers** window. To create a **Custom Registers** window, choose **Register**, **Custom**, and **Manage**. Then add the registers that you want to display. The **Custom Registers** window appears immediately after it's created.

Each **Custom Registers** window displays a title that you specify and only the registers that you choose to monitor. The **Custom Registers** window shown in [Figure 2-67](#) displays the contents of five registers.

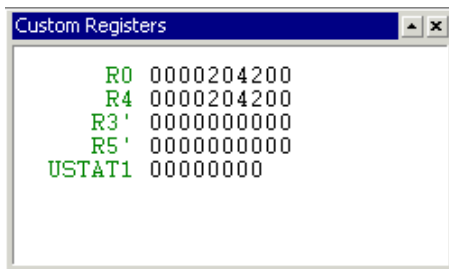


Figure 2-67. Custom Registers Window

Multiprocessor Window

 The SHARC and TigerSHARC simulators do not support multiprocessor (MP) debugging. Multiprocessing is available in emulator sessions only.

Use the **Multiprocessor** window ([Figure 2-68](#)) to select and control the different processors in a multiprocessor debug session.

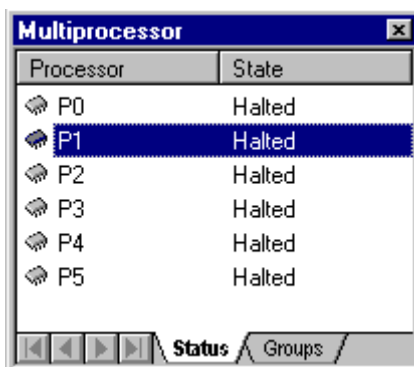


Figure 2-68. Multiprocessor Window

The window consists of two pages, **Status** and **Groups**.

Multiprocessor Window Pages

The **Multiprocessor** window has two tabbed pages, **Status** and **Groups**.

Status Page

The **Status** page ([Figure 2-69](#)) shows the status of each processor in a multiprocessor system. The processor with focus is highlighted with a horizontal bar.

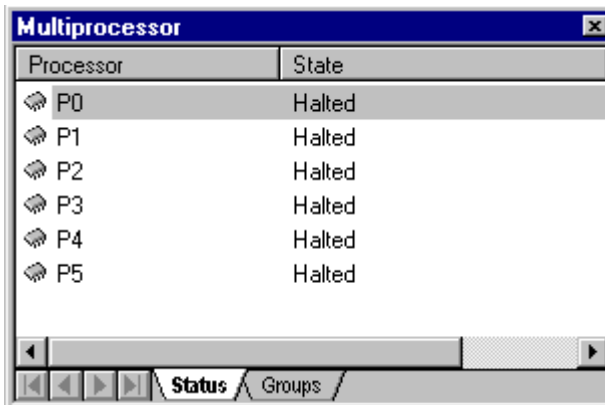


Figure 2-69. Multiprocessor Window – Status Page

You can change focus by clicking on a processor in the list.

Debugging Windows

Groups Page

The **Groups** page (Figure 2-70) shows the current list of multiprocessor groups. A **Default** group is created with each new multiprocessor session. The members of the **Default** group are the processors that you checked off under **Multiprocessor System** in the **New Session** dialog box.

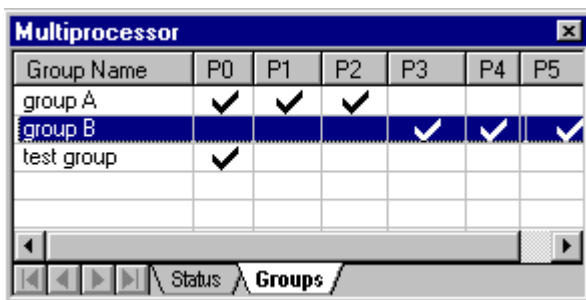


Figure 2-70. Multiprocessor Window – Groups Page

From the **Groups** page, you can assign one or more processors to a group to apply a multiprocessor operation (MP **Run**, MP **Halt**, MP **Step**, MP **Reset**, and MP **Restart**) to only the processors in the currently selected group.

Right-clicking on the **Group** page displays a context menu that lets you add or remove a group.

Multiprocessor Groups

If a session contains three processors (A, B, and C) and a group is created that contains A and C, the MP **Run** command runs A and C only, and B remains unaffected.

Focus

Processor focus changes, depending on the window currently selected. To move focus among the processors, click on a processor listed in the **Multiprocessor** window (Figure 2-68).

You can pin a register window, a memory window, or **Disassembly** window to a specific processor. Select the processor in the **Multiprocessor** window and right-click in the window that you want to pin. Then choose **Pin to Processor** to lock the window to the selected processor. A window pinned to a processor always displays data from that processor, regardless of the currently focused processor.

For example, if a register window is pinned to Processor 1 and a memory window is pinned to Processor 2, selecting the register window moves the focus to Processor 1. Selecting the memory window moves the focus to Processor 2. The **Multiprocessor** window's **Status** page reflects the change in focus.

Right-Click Menu

The **Multiprocessor** window's right-click menu offers these commands:



Figure 2-71. Multiprocessor Window's Right-click Menu

Pipeline Viewer Window

The Pipeline Viewer window (Figure 2-72) enables you to view instructions in the pipeline and event details. Open this window by choosing View, Debug Windows, and Pipeline Viewer.

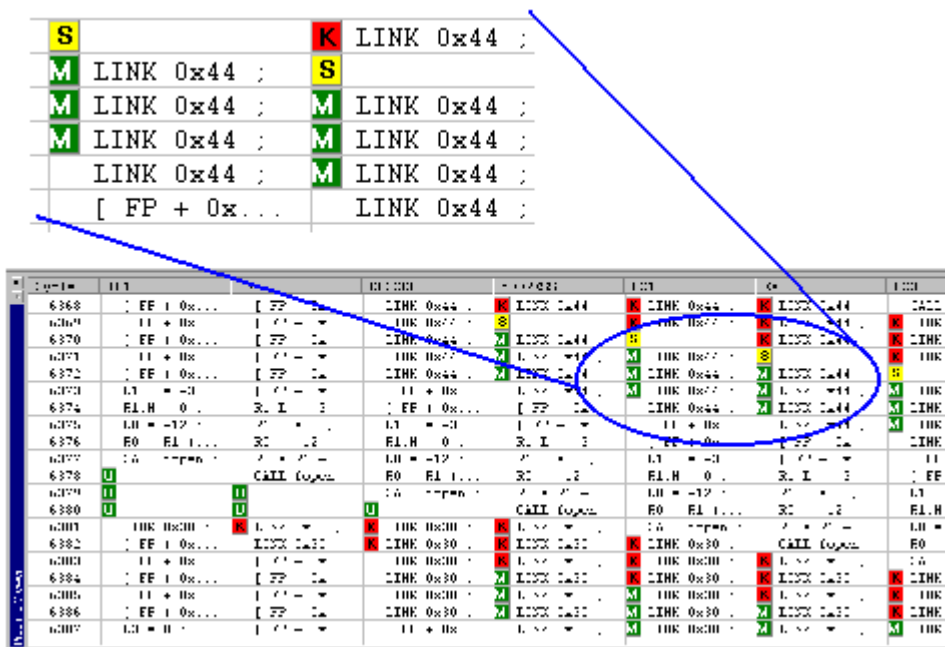


Figure 2-72. Pipeline Viewer Window

i Pipeline Viewer is available only for TigerSHARC ADSP-TS101 and ADSP-TS20x simulation targets. For SHARC processors, the Disassembly window displays symbols (F, D, or E) to indicate an instruction's pipeline stage.

Column headings refer to pipeline stages for the processor's core registers. Refer to your processor's Hardware Reference for details.

Right-Click Menu

The **Pipeline Viewer** window's right-click menu provides the commands described in [Table 2-26](#).

Table 2-26. Pipeline Viewer Right-Click Menu

Item	Purpose
Enabled	Enables and disables collection of pipeline data while running or stepping
Clear	Clears the current sample buffer
Display Format	Controls the display format of data Address shows the hexadecimal-formatted address of the pipeline stage (for example, 0x1234). Use this format to follow a particular address route through the pipeline. Disassembly disassembles the instruction at that address and shows the opcode mnemonic, similar to a Disassembly window. Use this format to determine why a particular event is occurring. Opcode format is the hexadecimal representation of the disassembly mnemonic.
Save	Opens the Save As dialog box, where you export the collected data to a text file
Properties	Opens the Pipeline Viewer Properties dialog box, where you view and specify properties (buffer and display depth, display format, column widths, grid lines, and the appearance of stages) for the Pipeline Viewer window. You can also modify window colors.

Pipeline Viewer Properties Dialog Box

The **Pipeline Viewer Properties** dialog box enables you to specify how (amount and format) the **Pipeline Viewer** window displays pipeline events. [Table 2-27](#) describes the Pipeline Viewer properties.

Table 2-27. Pipeline Viewer Properties

Property Item	Purpose
Buffer depth	Specifies the total number of pipeline samples to retain at any time. When this buffer overflows, the oldest data shifts out to make room for new samples. The default is 100.
Display depth	Specifies the number of samples to display. Adjust this number to meet your performance needs. The lower the depth, the faster the target can run. This option cannot be set greater than the Buffer depth . The default is 20.
Display format	Specifies the data's format Address includes the hexadecimal-formatted address of the pipeline stage (for example, 0x1234). Disassembly includes the opcode mnemonic, similar to the format displayed in a Disassembly window. Opcode format is the hexadecimal representation of the disassembly mnemonic.
Show gridlines	Toggles the display of gridlines in the window. The default is On .
Auto-size columns	Automatically sizes all columns to have the same width as samples are collected. The default is On .
Stages to view	Specifies the stages to appear in the window. Note that all stages are collected, but you view only the stages that you select to appear.

The **Colors** tab lets you specify the colors displayed in the **Pipeline Viewer** window. The current color appears under **Current Color**. Click a color in the color palette or click **Other** to specify a custom color. Use the **Reset** button to restore the default colors.

Pipeline Viewer Window Event Icons

Table 2-29 shows the **Pipeline Viewer** window icons that indicate pipe stage events for ADSP-TS101 and ADSP-TS20x processors.

Table 2-28. Event Icons

Icon	Event	Description
	Abort	A stage contains an instruction that has been aborted.
	Invalid Instruction in fetch pipe	A stage contains a placeholder representing a result of an invalid fetch. This condition occurs when an instruction alignment buffer is full; or the fetch pipe was flushed because of an abort in the execution pipeline.
	Stall	A stall has been generated at a stage of the pipeline.
	Wait	An instruction at a stage of the pipeline waits to be executed (because of a stall down the pipeline).
	Bubble	The pipeline stage contains an invalid instruction as a result of a stall up the pipeline.
	Hit	An instruction at a stage is a Branch Target Buffer Hit. The address of the last slot of the instruction line was found in the Branch Target Buffer.

The icons in the above table are listed in descending priority. When more than one event occurs at a certain stage at a certain cycle, only one icon is displayed—the icon with highest priority. For example, if an instruction that was a Branch Target Buffer hit is aborted, the **Abort** icon  appears.

Pipeline Instruction Event Details

In the Pipeline Viewer window, moving the cursor over a Pipeline Viewer event icon displays pipeline event details in a tool tip (message) box, shown in [Figure 2-73](#).

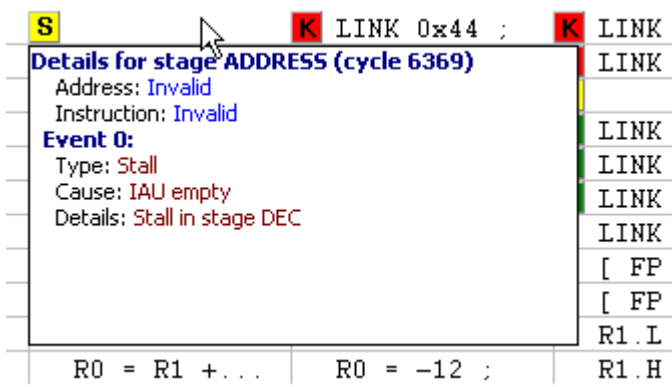


Figure 2-73. Tool Tip Box Showing Pipeline Event Details

A pipeline event can include the details described in [Table 2-29](#).

Table 2-29. Pipeline Event Details

Item	Displays
Address	Address of the pipeline stage at that cycle (if valid)
Instruction	Assembly instruction of that address (if valid)
Type	Type of event
Cause	Cause of the event condition
Details	Further explanation of the cause of the event (if applicable)

Cache Viewer

The VisualDSP++ Cache Viewer provides a means to visualize a processor's cache and locate problem areas. The tool shows how instructions are being executed. Use this valuable information to boost your application's performance.

The **Cache Viewer** window (Figure 2-74) provides a view of each instruction's execution characteristics. Cache Viewer information indicates the type of event and describes the cause of the event. Each instruction that executes from cache is marked with an **H** (hit) or an **M** (miss). Hits represent cache instructions executed without a stall. Misses identify instructions fetched from slower parts of memory because they were not found in cache.

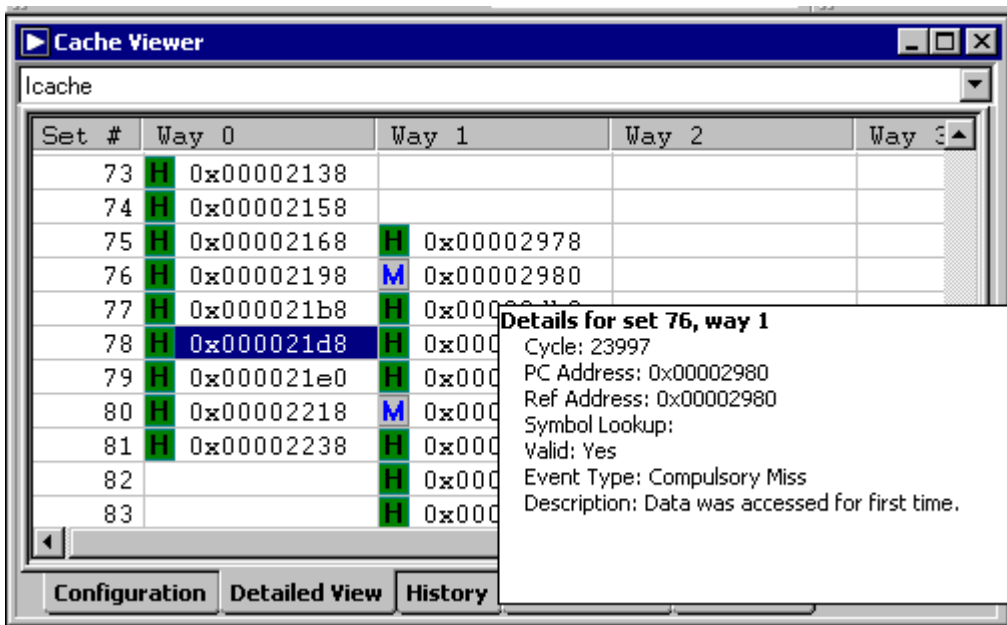


Figure 2-74. Viewing a Cache Event's Details in the Cache Viewer

Debugging Windows

Use the hit or miss information to increase an application's performance by locating instructions in the cache when they are needed. Ensuring that no cache misses are in frequently executed areas of an application (as highlighted by the profiler utility) is a critical step in optimizing your application's software performance.

As shown in [Figure 2-74](#), the **Cache Viewer** window enables you to view the details of any cache event. These descriptive details help you understand the cause of the cache event. Use this information to isolate areas where performance can be improved.

For example, based on cache event details, you might:

- Modify an application's layout in memory to avoid cache thrashing
- Prefetch instructions to avoid compulsory misses
- Lock down ways in the cache to avoid a conflict miss with a frequently accessed instruction

Open the **Cache Viewer** window by choosing **View, Debug Windows**, and **Cache Viewer**.

The Cache Viewer consists of several tabbed pages, described in [Table 2-30](#).

Table 2-30. Cache Viewer Pages

Page	Displays
Configuration	Cache configuration information
Detailed View	Location (set and way) of cache events
History	List of cache events
Performance	Cache performance metrics
Histogram	A plot of cache activity
Address View	Cache events on an Address vs. Cycle plot

The **Cache Viewer** window's right-click menu (described in [Table 2-31](#)) enables you to read, write, and step an *event log*, a file that records cache events.

Table 2-31. Cache Viewer Window's Right-Click Menu

Menu Option	Description
Enabled	Enables and disables collection of cache data while the target is running or stepping
Clear	Clears all displays and deletes all stored cache data
Map References	Opens the Map References dialog box, where you specify the cache reference map (start address and end address)
Event Log→Read	Opens the File Open dialog box, where you select and open a cache event log file. The log file data is used by the Cache Viewer window's Configuration view.
Event Log→Write	Opens the File Save dialog box, where you save a cache event log file. Events are written to this log file.
Event Log→Step	Executes one cache event at a time from the cache event log file. The event displays in the Detailed View, History, and Histogram pages of the Cache Viewer window. By default, this option is enabled when a cache log file is opened for reading.
Properties	Opens the Cache Viewer Properties dialog box, where you specify the Cache Viewer window's appearance



The event log file does not include icons. Thus, the **Cache Viewer** window's **Detailed View** page does not display icons.

Stepping enables you to execute one cache event at a time from the cache events log file. The event is displayed on the **Detailed View**, **History**, and **Histogram** pages. When stepping is configured, a check mark appears next to **Step** on the right-click menu. By default, this option is enabled when a cache events log file is opened for reading.

Configuration Page

The **Configuration** page (Figure 2-75) displays configuration information for configured cache.

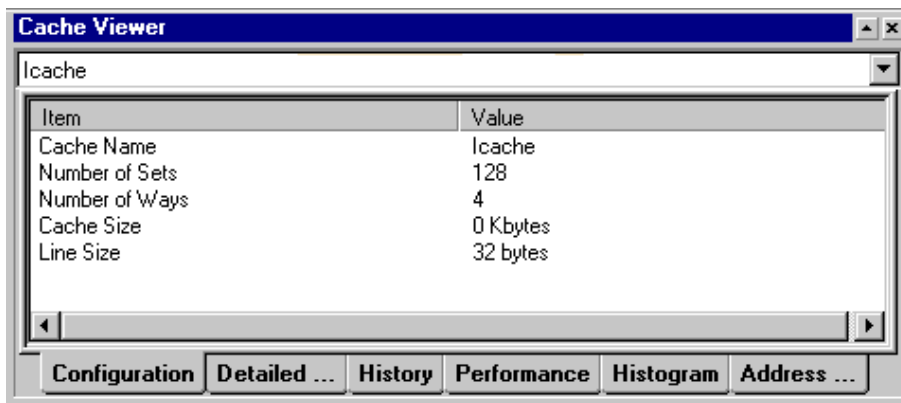


Figure 2-75. Configuration Page

The **Cache Selection** pull-down box (top of dialog box) lists cache displays. If more than one cache is configured, you can use this list to change cache displays.

The **Cache Configuration** list box (below the **Cache Selection** pull-down box) displays a list of items and their values. The first three items (Cache Name, Number of Sets, and Number of Ways) are required. The target may display additional items, such as Cache Size and Line Size. The list of items depends on the selection in the **Cache Selection** pull-down box.

Detailed View Page

The Detailed View page (Figure 2-76) displays a grid depicting cache sets (rows) and cache ways (columns).

Set #	Way 0	Way 1	Way 2	Way 3
10	H 0xf0000558			
11	H 0xf0000178	H 0xf0000560		
12	H 0xf0000198	H 0xf0000598		
13	H 0xf00001b8	H 0xf00005b8		
14	H 0xf00001d8	H 0xf00005d8		
15	H 0xf00001f8			
16	H 0xf0000218			
17	H 0xf0000228			
18	H 0xf0000a58			
19	H 0xf0000a78			
20	M 0xf0000a80			
21				
22				
23				

Figure 2-76. Detailed View Page

Data received from a cache event is placed in the cell corresponding to the cache set and way. The most recent events are highlighted.

Each cell has an icon and text entry. The icon indicates the type of cache event that occurred (hit or miss). Depending on the objects you choose to display, you can view details, such as reference address, PC address, cycle count, event type, event description, and so on.

Debugging Windows

You can also view tool tips showing details for the most recent cache event. The appearance of a lock icon in the column header indicates that the cache way is locked.

A reference map icon in the Set # column indicates the results of the reference mapper function. Double-clicking on a cell switches the display to the history view (**History** page) for the selected cell.

History Page

The **History** page (Figure 2-77) displays detailed information for each cache event that occurred in the selected set and way.

Index #	Set #	Way #	Cycle	PC Address	Ref Address	Symbol L...	Va
2	5	0	13...	0xf00004b8	0xf00004b8		Ye
1	5	0	13...	0xf00004b0	0xf00004b0		Ye
0	5	0	13...	0xf00004a8	0xf00004a8	_init_argv	Ye
8	6	0	13...	0xf00004d8	0xf00004d8		Ye
7	6	0	13...	0xf00004d0	0xf00004d0		Ye
6	6	0	13...	0xf00004c8	0xf00004c8		Ye
5	6	0	13...	0xf00004c0	0xf00004c0		Ye
4	6	0	13...	0xf00004d8	0xf00004d8		Ye
3	6	0	13...	0xf00004d8	0xf00004d8		Ye
2	6	0	13...	0xf00004d0	0xf00004d0		Ye
1	6	0	13...	0xf00004c8	0xf00004c8		Ye
0	6	0	13...	0xf00004c0	0xf00004c0		Ye
2	7	0	13...	0xf00008f8	0xf00008f8		Ye
1	7	0	13	0xf00008f0	0xf00008f0		Ye

Figure 2-77. History Page

Select the set and way from the pull-down control or by double-clicking a cell on the **Detailed View** page.

You can specify the number of cache events stored. Sort the rows by clicking on any particular column heading. An up arrow in a column heading indicates an ascending sort order, and a down arrow indicates a decending sort order.

[Table 2-32](#) describes the history information for cache events.

Table 2-32. History Information for Cache Events

Item	Description
Index #	Shows the order in which the cache events were received. The index starts at zero and increments each time an event is received.
Set #	Displays the set number where the cache event occurred
Way #	Displays the way number where the cache event occurred
Cycle	Displays the cycle count when the cache event occurred
PC Address	Displays the PC address of the cache event
Ref Address	Displays the reference address of the cache event
Symbol Lookup	Displays the symbol name when the reference address resolves to a symbol in memory
Valid	Displays the cache line valid flag (Yes or No)
Event Type	Displays the cache event type, such as Hit or Miss
Description	Displays the cache event's description

Debugging Windows

Performance Page

The **Performance** page (Figure 2-78) shows a list of performance metrics (items and values), which are determined by the target.

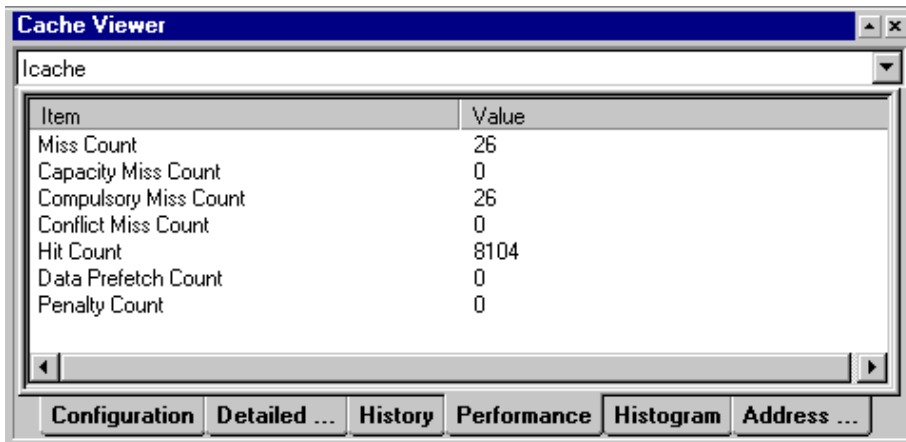


Figure 2-78. Performance Page

The target updates this list. The update rate, however, is not predetermined.

Histogram Page

The **Histogram** page (Figure 2-79) shows a plot of the total number of cache events that occurred in each cache set.

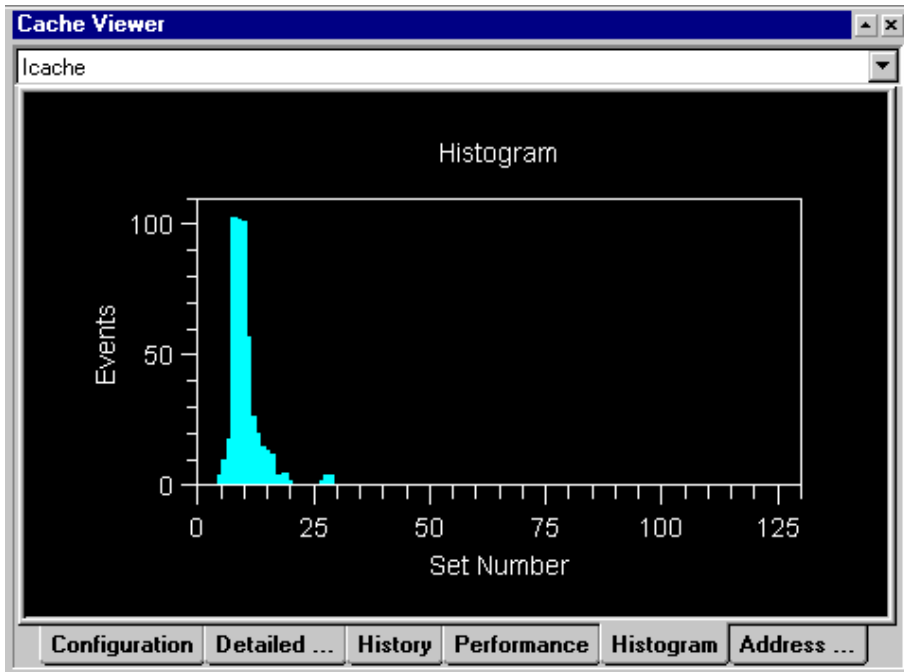


Figure 2-79. Histogram Page

A vertical line is displayed for each cache set. The line starts at zero and ends at the total number of events. Use this plot to identify the most active cache sets.

Address View Page

The **Address View** page (Figure 2-80) displays cache events on an Address versus Cycle plot. Use this view to display the cache events for the specified addresses over time.

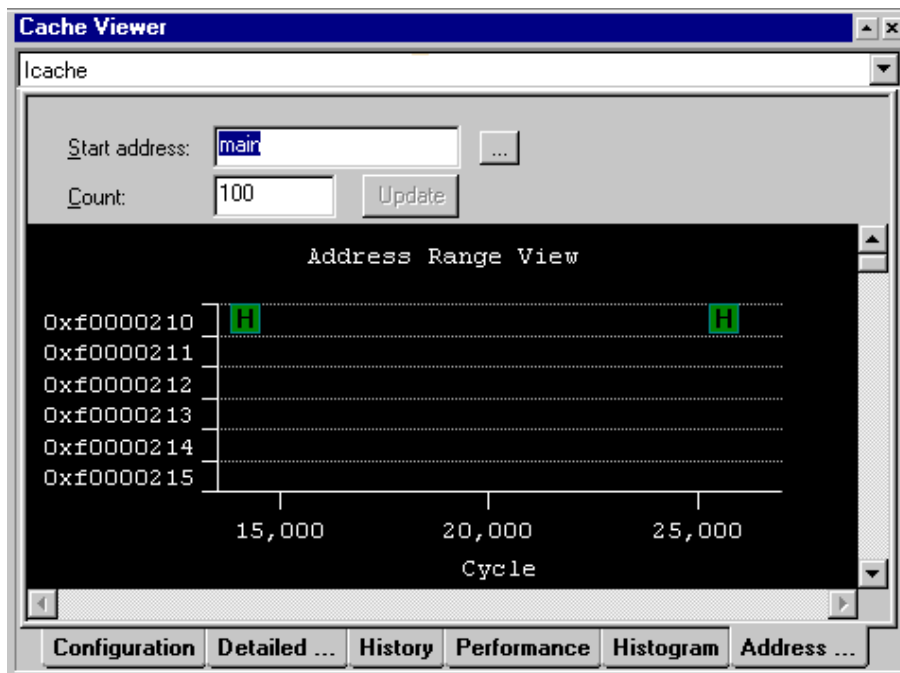


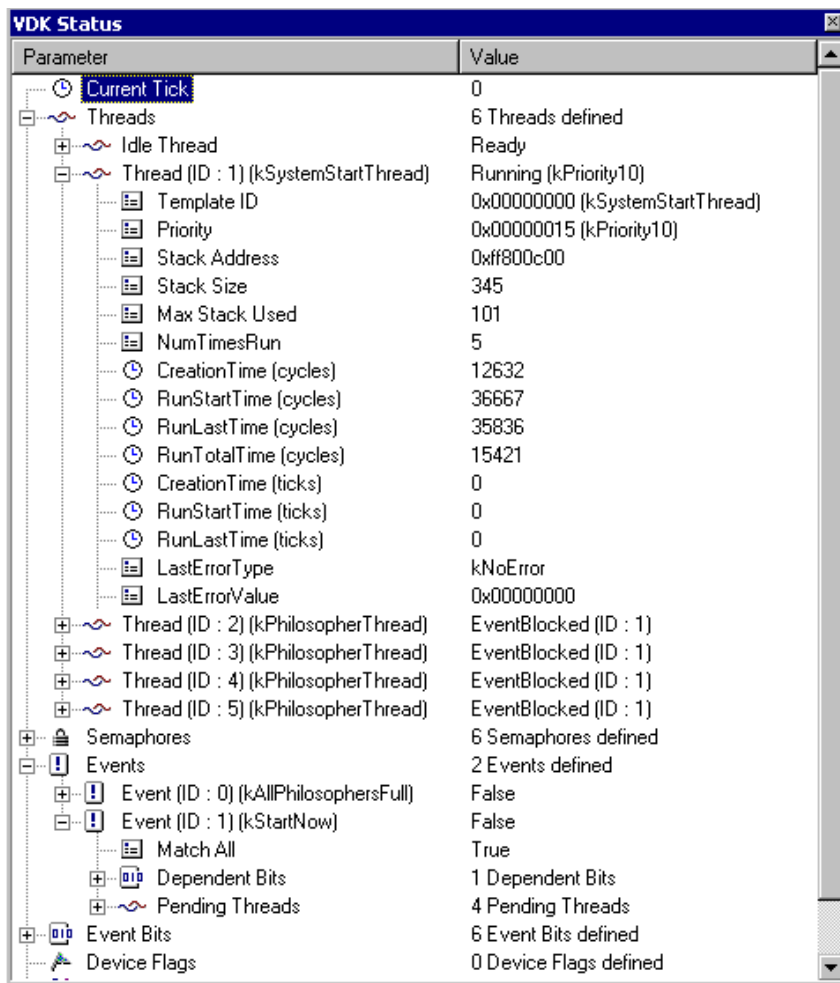
Figure 2-80. Address View Page – Address Range View

Events are displayed as icons, identical to the icons used in the detailed view. A start address and count are required. You can enter the start address as a hexadecimal value or a symbol. Click the browse (...) button to browse for a symbol.

The count determines the number of addresses displayed. After entering a start address and count, click **Update** to display the event data. Horizontal and vertical scroll bars enable you to scroll the view.

VDK Status Window

The VDK Status window (Figure 2-81) is available when an executable file is built with VDK support enabled. Open this window by choosing View, VDK Windows, and Status.



Parameter	Value
Current Tick	0
Threads	6 Threads defined
Idle Thread	Ready
Thread (ID : 1) (kSystemStartThread)	Running (kPriority10)
Template ID	0x00000000 (kSystemStartThread)
Priority	0x00000015 (kPriority10)
Stack Address	0xff800c00
Stack Size	345
Max Stack Used	101
NumTimesRun	5
CreationTime (cycles)	12632
RunStartTime (cycles)	36667
RunLastTime (cycles)	35836
RunTotalTime (cycles)	15421
CreationTime (ticks)	0
RunStartTime (ticks)	0
RunLastTime (ticks)	0
LastErrorType	kNoError
LastErrorValue	0x00000000
Thread (ID : 2) (kPhilosopherThread)	EventBlocked (ID : 1)
Thread (ID : 3) (kPhilosopherThread)	EventBlocked (ID : 1)
Thread (ID : 4) (kPhilosopherThread)	EventBlocked (ID : 1)
Thread (ID : 5) (kPhilosopherThread)	EventBlocked (ID : 1)
Semaphores	6 Semaphores defined
Events	2 Events defined
Event (ID : 0) (kAllPhilosophersFull)	False
Event (ID : 1) (kStartNow)	False
Match All	True
Dependent Bits	1 Dependent Bits
Pending Threads	4 Pending Threads
Event Bits	6 Event Bits defined
Device Flags	0 Device Flags defined

Figure 2-81. VDK Status Window

Debugging Windows

When the execution of a VDK program is halted, VisualDSP++ reads data for threads, semaphores, events, event bits, device flags, memory pools, and messages and displays the state and status data in this window. When one of the above VDK entities is created, it is added to the display. An entity is removed from the display when it is destroyed.

Initially, information is displayed in a collapsed state, which shows only the name of the entity and, in some cases, its current state. When a thread is in the Ready state, its priority is displayed.

Clicking the plus sign () next to the name of an entity expands the view.

The possible thread states are as follows.

- Running
- Ready
- SemaphoreBlocked
- EventBlocked
- DeviceFlagBlocked
- MessageBlocked
- SemaphoreBlockedWithTimeout
- EventBlockedWithTimeout
- DeviceFlagBlockedWithTimeout
- MessageBlockedWithTimeout
- Sleeping
- Unknown

See the *VisualDSP++ 3.5 Kernel (VDK) User's Guide for 32-Bit Processors* for details.

VDK State History Window

VDK state history is available only for DSP executable files with VDK support. During execution of a VDK-enabled program, thread and event data are collected in a history buffer if **Full Instrumentation** is specified for the project. When a running program is halted, the history buffer data is plotted in the **VDK State History** window, described in [Figure 2-82](#). Some features become available only when the data cursor is enabled. Open this window by choosing **View, VDK Windows, and History**.

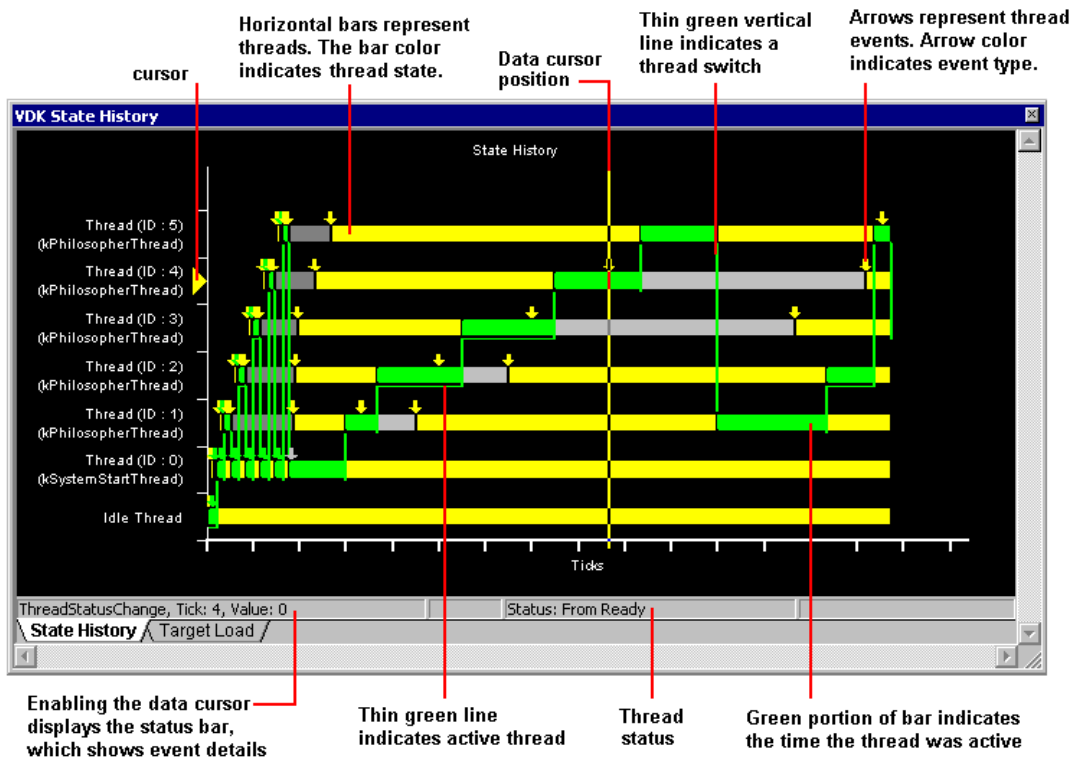


Figure 2-82. VDK State History Window

Debugging Windows

Each thread appears as a horizontal bar (thread status bar). The ThreadID and the name of the thread type appear to the left of the bar. When a thread is destroyed, the name of the thread type is no longer displayed. Each thread event appears as an arrow above a thread.

Thread Status and Event Colors

Threads and events are coded by color, based on thread status and event type. The colors appear in the horizontal bars (threads) and colored arrows (events) used throughout the plot. Events of the same type are drawn in the same color.

Right-click on the plot and choose **Legend** to display legends that define each color in the **VDK State History** window. To customize colors, right-click on the plot and choose **Properties**.

Trace thread-switched history by following the thin green line, which winds through the display, passing under threads to indicate the running thread at any particular time. When a context switch occurs and changes the running thread, a vertical green line is drawn from the previously running thread to the next running thread.

When you use the data cursor, a yellow triangle to the left of a thread name identifies the currently running thread.

Window Operations

The status bar (at the bottom of the plot) on the **State History** page shows the event's details and thread status when the data cursor is enabled. Event details include the event type, the tick when the event occurred, and an event value. The value for a thread-switched event indicates the thread being switched in or out.

Right-click on the plot and choose **Data Cursor** to activate the data cursor, which is used to display event and thread status details. Based on the event that occurred, the thread status changes. Press the keyboard's right arrow key or left arrow key to move to the next or previous event. When the data cursor hits a thread-switched event, it moves to the thread being switched in. The yellow triangle to the left of the thread name indicates the currently active thread.

You can zoom in on a region to examine that area in more detail. Hold the left mouse button down while dragging the mouse to create a selection box. Then release the mouse button to expand the plot. To restore the plot to its original scale, right-click on the plot and choose **Reset Zoom**.

Right-Click Menu

The **VDK State History** window's right-click menu provides easy access to operations that can be performed from the state history plot.

Target Load Window

Clicking the **Target Load** tab from the **VDK State History** window displays the **Target Load** window. A target load plot ([Figure 2-83](#)) shows the percentage of time that the target spent in the Idle thread.

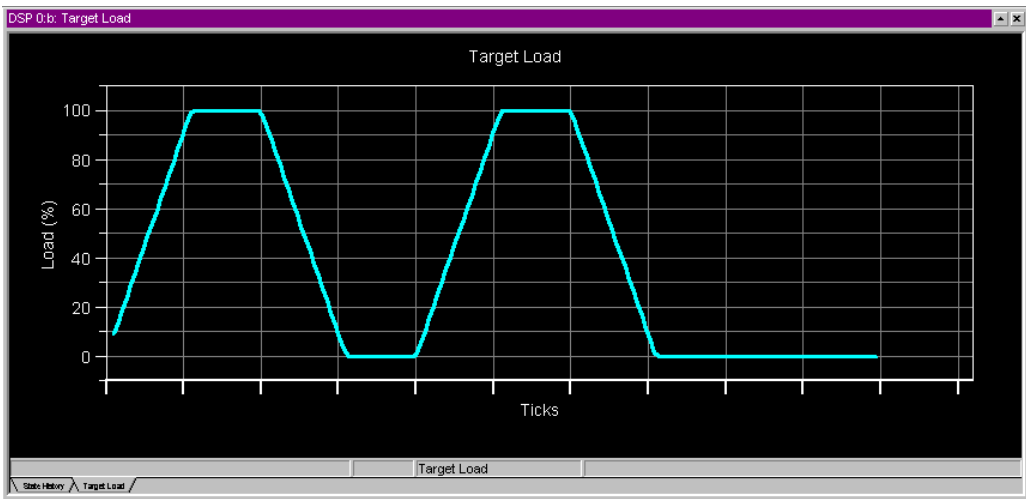


Figure 2-83. Target Load Window Plot

A load of 0% indicates that VDK spent all of its time in the Idle thread. A load of 100% indicates that the target did not spend any time in the Idle thread.

Load data is processed by means of a moving window average.

Plot Windows

Use a plot window to display a *plot*, which is a visualization of values obtained from processor memory. You can display one or multiple plot windows by choosing **View, Debug Windows, Plot, and New**.

In the **Plot Configuration** dialog box, specify the contents of a plot. In the **Plot Settings** dialog box, specify the presentation of a plot. You can modify a plot's configuration and immediately view the revised plot.

Figure 2-84 shows an example of a plot window.

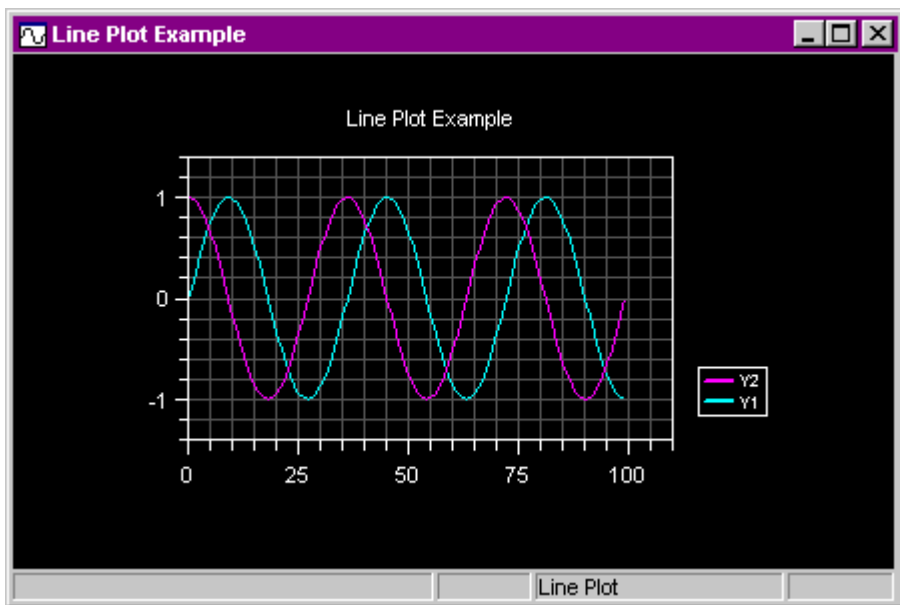


Figure 2-84. Plot Window

From a plot window, you can zoom in on a portion of a plot or view the values of a data point.

Debugging Windows

You can print a plot, save the plot image to a file, or save the plot's data to a file. For details, refer to the online Help in VisualDSP++.

Plot Window Features

Plot windows include a status bar, tool bar, and a right-click menu.

Status Bar

The status bar, located at the bottom of the plot window, displays the plot type and other information, depending on the plot type and other settings.

The following examples show different plot information displayed on the status bar.



Figure 2-85. Status Bar Information for Plots

In a waterfall plot, the status bar indicates the azimuth and elevation viewing angles. If you zoom in on a region, the status bar indicates that zoom is enabled. When you use the data cursor, the status bar shows the selected point's data value.

When a plot window's auto-refresh mode is enabled in BTC mode, the status bar indicates current buffer capacity (for example, 89%) and data logging status.

Buffer capacity, which dynamically changes between 0 and 100%, indicates the portion of the buffer currently in use. The ideal size is a little below 100%. Readings of 100% indicate lost data.

[Table 2-33](#) describes the data logging status indicators in a plot window.

Table 2-33. Data Logging Status Indicators in a Plot Window

Status	Indicates
Record	Real-time data being displayed is also being saved (logged) to a .BIN file.
Live	Data is being displayed in real time.
Playback	A previously saved data (log) file is being viewed.

Tool Bar

The plot window's toolbar, shown in [Figure 2-86](#), provides buttons for recording and playing back streaming data and a box for specifying streamed data (.BIN) file names.

Debugging Windows

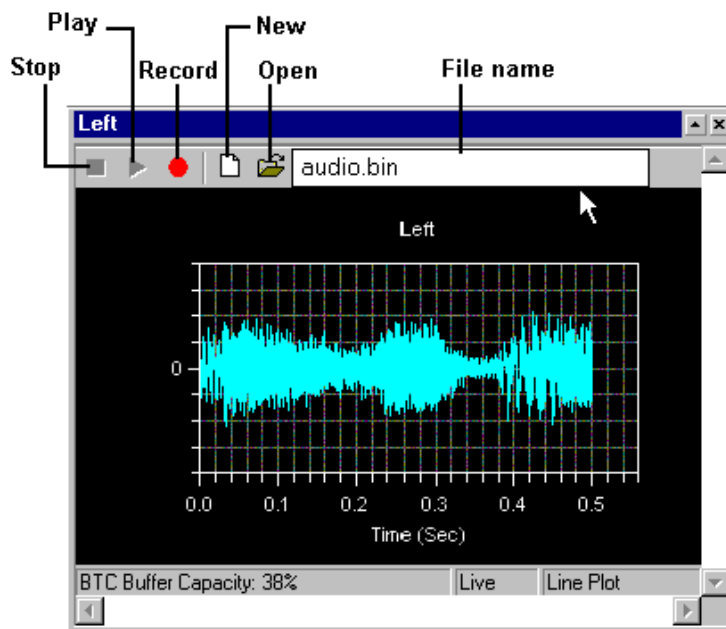


Figure 2-86. Plot Window's Toolbar

Right-Click Menu

The plot window's right-click menu is shown in [Figure 2-87](#).

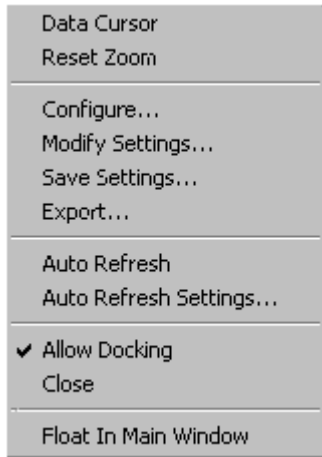


Figure 2-87. Plot Window's Right-Click Menu

This menu provides access to the standard window options (docking, closing, and floating in the main window) and to the plot window features described in [Table 2-34](#).

Table 2-34. Plot Window Operations

Feature	Description
Data Cursor	Displays the data value associated with the position of the plot window's data cursor. View the value on the left side of the plot window's status bar. Press the keyboard's arrow keys to move around the graph.
Reset Zoom	Resets the plot window to its initial full-scale display
Configure	Opens the Plot Configuration dialog box, where you add, remove, or modify data sets. You can also change the plot type and rename the plot.
Modify Settings	Opens the Plot Setting dialog box, where you customize the plot's appearance. You can specify plot settings (grids, colors, margins, fonts, axes, and so on) and settings for each data set (data processing).
Save Settings	Saves plot configuration settings for future use. The configuration is stored, but not the data. You can retrieve settings (.VPS file) and load new plot data.
Export	Exports the plot image to various destinations including the Windows clipboard. Save the plot image as a file (JPG, GIF, TIF, EPS, TXT, or DAT format) or print a hard copy.
Auto Refresh	Enables a plot window to refresh automatically based on settings that you specify. The auto-refresh timer starts. Streaming data is read and displayed. When this option is deselected, the timer is stopped and streaming data is not processed.
Auto Refresh Settings	Enables you to configure options that control auto-refresh settings for plot windows. These settings determine the method in which memory is transferred.

Plot Window Statistics

View various statistics (mean, standard deviation, signal-to-noise ratio (SNR), minimum data value, and maximum data value) while displaying a plot. Note that statistics apply only to the portion of data that is visible. When the plot is zoomed, the statistics are recalculated only for the visible area.

Figure 2-88 shows statistics displayed for a portion of audio data.

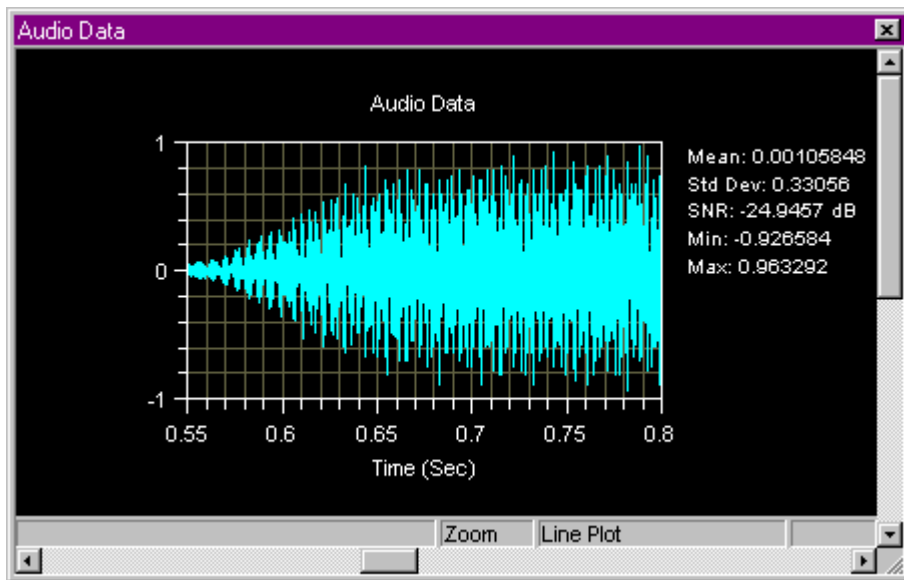


Figure 2-88. Statistics Displayed for a Portion of Audio Data

For details about viewing statistics, refer to the VisualDSP++ online Help.

Plot Configuration

A plot configuration comprises two parts: data values and presentation (configuration) settings.

A plot window must contain at least one *data set*, a series of data values in processor memory. Create data sets in the **Plot Configuration** dialog box, shown in [Figure 2-89](#).

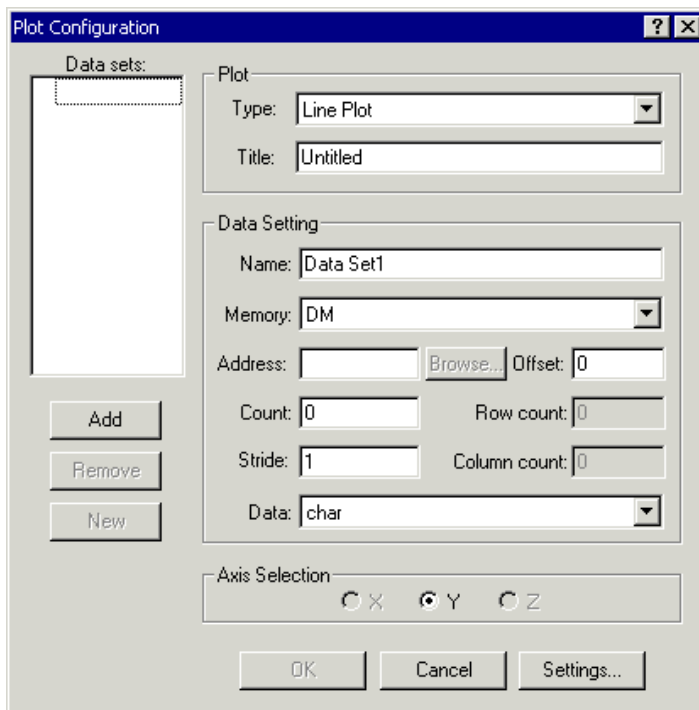


Figure 2-89. Plot Configuration Dialog Box

Specify the type of plot (for example, waterfall), the memory location, the number of values, the axis associated with each data set, and other options that identify the data. Note that 3-D plots require additional specifications for row and column counts.

The **Settings** button enables you to configure presentation options (such as titles, grids, fonts, colors, and axis presentation) for each data set. You can recall a plot from a saved settings file (.VPS). VisualDSP++ uses these settings and reads processor memory to display a plot window.

Plot Window Presentation

Customize the presentation of a plot window to fit your needs. Configure presentation settings from the **Plot Settings** dialog box, which you can invoke as follows.

- Right-click from within a plot window
- Click the **Settings** button in the **Plot Configuration** dialog box

The **Plot Settings** dialog box provides the tabs shown in [Figure 2-90](#).

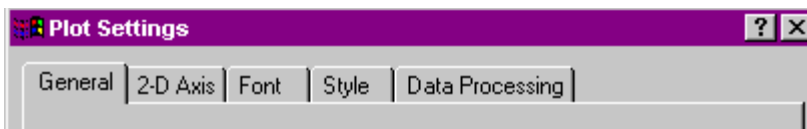


Figure 2-90. Tabs in the Plot Setting Dialog Box

Options on the tab pages enable you to configure the plot window's presentation. On the **Style** page, for example, you can easily specify symbols for a data set as well as line type and width, as shown in [Figure 2-91](#).

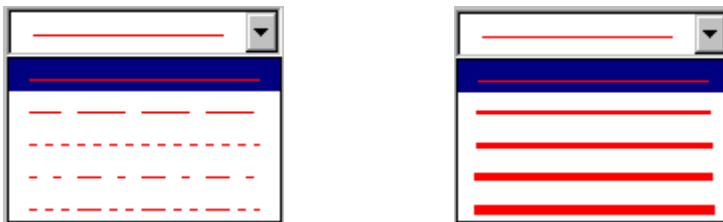


Figure 2-91. Line Styles

Debugging Windows

In addition to the many presentation options, you can select a rectangular area, as shown in [Figure 2-92](#), and zoom in on it.

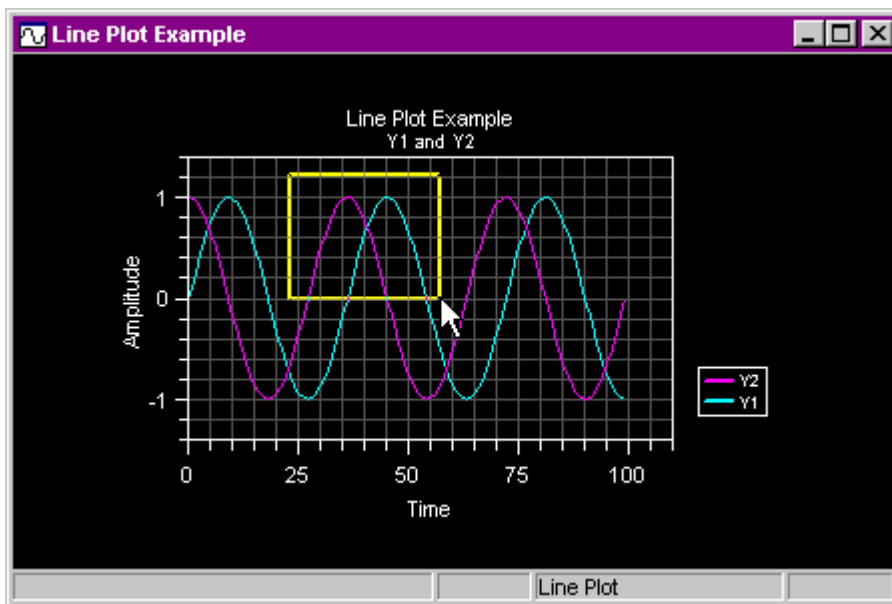


Figure 2-92. Zooming in on a Selected Area

Plot Presentation Options

Depending on the type of plot, many plot presentation options are available.

In the **Plot Settings** dialog box, these options are grouped by function on tabbed pages, described in [Table 2-35](#).

Table 2-35. Plot Settings Options by Page

Page	Options That You Can Specify
General	Title and subtitle, grid lines, margins, background colors, and legend
2-D Axis	For X-axis and Y-axis: axis titles, start and increment values, scales
3-D Axis	For X-axis, Y-axis, and Z-axis: axis titles, Z-axis settings, step sizes, scale multipliers, color and mesh
Font	Font name, color, and size
Style	For a data set: line type, width, color; symbol and type
Data Processing	For a data set: data processing algorithm, sample rate, and triggering

You can specify a plot's presentation options before generating the plot (while configuring the plot) or after generating the plot.

Image Viewer

The VisualDSP++ **Image Viewer** window enables you to perform these operations:

- View an image. You can display BMP, JPEG, PPM, or MPEG data from processor memory or from a file on your PC.
- Configure image attributes such as number of pixels, bits per pixel, and image format
- Correct the gamma attributes of an image. For a color image, you can adjust the red, green, and blue pixel values. On a gray-scale image, you can adjust darkness only.
- Copy an image to the Windows clipboard
- Print an image or save it to a file
- Export an image

You select the image source (from processor memory or a file on your PC) and specify image attributes. If the image is located in processor memory, you must specify the image's address, size, and format.

To open the **Image Viewer** window, choose **View, Debug Windows, and Image Viewer**.

The **Image Viewer** window, shown in [Figure 2-93](#), renders the image and provides scroll bars and buttons for zooming in and out.

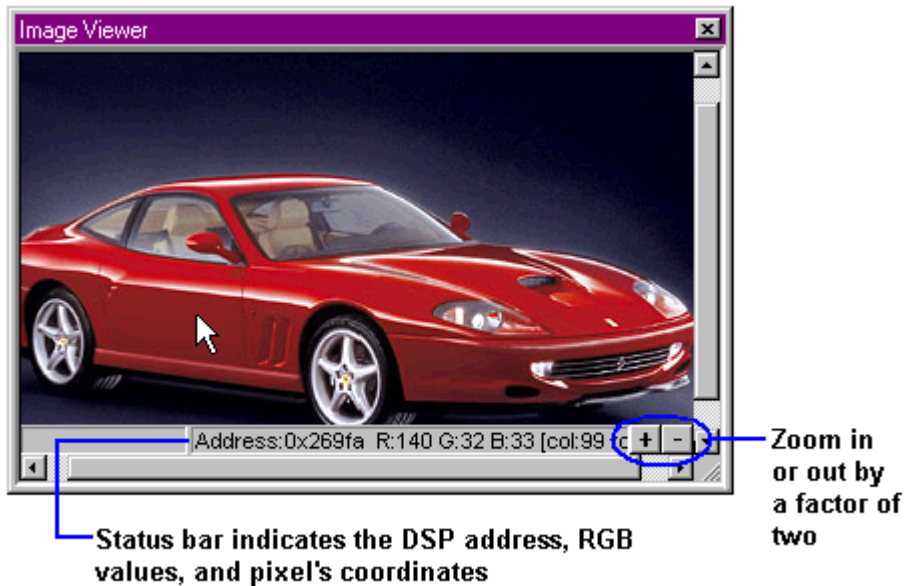


Figure 2-93. Image Viewer Window

As you move the mouse over the image, the status bar indicates:

- Processor address where the selected pixel is located
- Red, green, and blue (RGB) pixel values for color images, intensity values for gray-scale images
- Pixel coordinates (column and row)



Pixel color depth is 24 bits for color images and 8 bits for gray-scale images.

Right-Click Menu

Figure 2-94 shows the **Image Viewer** window's right-click menu.



Figure 2-94. Image Viewer Window's Right-Click Menu

Table 2-36 describes the menu commands.

Table 2-36. Right-Click Menu Commands

Command	Purpose
Configure	Opens the Image Configuration dialog box, where you specify image attributes
Update Now	Reads the image data from processor memory
Reset Zoom	Displays the image in its original size
Export	Opens the Export Image dialog box, where you copy or print the image
Gamma Adjust	Opens the Gamma Correction dialog box, where you adjust image color
Play Video	Plays an MPEG video clip
Stop Video	Ends the playing of a video clip

Image Configuration Dialog Box

When using the Image Viewer, you must configure specifications for the image. [Table 2-37](#) describes the buttons and fields in the **Image Configuration** dialog box.

Table 2-37. Buttons and Fields in the Image Configuration Dialog Box

Item	Purpose
DSP Memory	Specifies Image or Video
File	Specifies a file on your PC. You then specify the file name and path. Clicking Browse opens the Select Image Import File dialog box, from which you navigate to the file.
Memory selection	Specifies the memory
Image start address (hex)	Specifies the first location of the image data
Horizontal pixels	Specifies the number of horizontal pixels
Vertical pixels	Specifies the number of vertical pixels
Bits per pixel	Specifies the number of bits per pixel. For color images, only 24 bits per pixel are currently allowed. For gray-scale images, only 8 bits per pixel are currently allowed.
Stride	Specifies the skip count. The default is one.
Image format	Specifies the format (RGB or Gray Scale)
Video bytes	Specifies the number of bytes (video images only)
Pack data	Specifies the storage of continuous bytes of RGB pixel data in target memory. When this option is not selected, each component of the pixel data is stored in a separate memory location.

Gamma Correction Dialog Box

When using the Image Viewer, you can adjust the image gamma. For color images, you can adjust red, green, and blue independently or in tandem. For gray-scale images, you can adjust only the black-white balance.

Debugging Windows

[Table 2-38](#) describes the buttons and fields in the **Gamma Correction** dialog box.

Table 2-38. Buttons and Fields in the Gamma Correction Dialog Box

Item	Purpose
Red	Specifies the red value
Green	Specifies the green value
Blue	Specifies the blue value
Link	Adjusts the red, green, and blue values at the same time (not for video images)
Gray	Specifies the black value (gray-scale images only)

Export Image Dialog Box

When using the Image Viewer, you can export an image to the Windows clipboard, a file, or to the printer.

The **Export Image** dialog box contains the buttons and fields described in [Table 2-39](#).

Table 2-39. Buttons and Fields in the Export Image Dialog Box

Item	Purpose
Clipboard	Copies the image to the Windows clipboard
File	Specifies a file name and path. The file name and path appear in the text box. Click Browse to navigate your system.
Printer	Sends the image to the default printer

3 DEBUGGING

This chapter describes VisualDSP++ debugging tools used during single-processor and multiprocessor debug sessions. The topics are organized as follows.

- “Debug Sessions” on page 3-2
- “Code Analysis Tools” on page 3-7
- “Program Execution Operations” on page 3-11
- “Simulation Tools” on page 3-17
- “Image Viewer” on page 3-19
- “Plots” on page 3-19
- “Flash Programmer” on page 3-28

Debug Sessions

You run the DSP projects that you develop as *sessions* (debug sessions).

A session is defined by the elements described in [Table 3-1](#).

Table 3-1. Specifying a Debug Session

Element	Description
Debug target	The debug target is the software module that controls a type of debug target (a simulator or emulator). The <i>simulator</i> is software that mimics the behavior of a DSP chip. Simulators are used to test and debug processor code before a DSP chip is manufactured. An <i>emulator</i> is software that “talks” to a hardware board that contains one or more actual DSP chips.
Platform	For a given debug target, several platforms may exist. For a simulator, the platform defaults to the identically named DSP simulator. When the debug target is an EZ-ICE® board, the platform is the board in the system on which you want to focus. When the debug target is a JTAG emulator, the platforms are the individual JTAG chains.
Processor	Multiple processors can exist for a given debug target and platform. When you create an executable file, the processor is specified by the Linker Description File (LDF) and other source files.

When setting up a session, set the focus on a series of more specific elements.

The target platform and processor settings specify the debug session. A default session name is automatically generated. You can further identify the session by modifying the default name, choosing a more meaningful name.



A well-chosen name can prevent confusion later.

Debug Session Management

You can run several debug sessions at once and can dynamically switch between sessions.

The typical reasons for running multiple debug sessions are:

- To write different versions of your program to compare their operating efficiencies
- To debug completely different programs without having to run multiple instances of VisualDSP++

Simulation Versus Emulation

When connected to a simulator session, you may open as many sessions as your system's memory can handle.

When connected to actual hardware through an emulator, you can have only **one** debug session connected to one emulator at any time. If multiple emulators are installed and are connected to multiple target boards, one debug session may be connected to each individual emulator.

 In a JTAG emulator session, only one debug session may be connected to each physical target/emulator combination. Otherwise, contention issues may arise. Upon switching to a different session, VisualDSP++ detaches from the old session before attaching to the new session.

Breakpoints

In a simulator session, a breakpoint can be set at any address in your executable program's memory. Program execution halts at the address where the breakpoint is located.

 *Hardware breakpoints* can be used in emulator debug sessions only.

Watchpoints

Watchpoints are like breakpoints that trap on a specified condition. You can set watchpoints on registers, stacks, and memory ranges. Reaching the condition halts program execution and updates all windows.

 Watchpoints are available only during simulation.

Multiprocessor (MP) Debugging

Often, performance-based products require two or more processors. A system built with multiple processors is called a multiprocessor system, and a system built with a single processor is called a single-processor system.

 Multiprocessor debugging is available only during emulation. The SHARC and TigerSHARC simulators do not support this feature.

Setting Up a Multiprocessor Debug Session

The first step in setting up a multiprocessor debug session is to develop a multiprocessing project by using the multiprocessing capabilities of the linker and an `.LDF` file to describe the multiprocessing system.

See the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*, especially the sections about `SHARED_MEMORY{ }` and `MPMEMORY{ }` commands.

The second step is to use the VisualDSP++ Configurator utility to describe the hardware to the VisualDSP++ software if you are running a JTAG emulator session. VisualDSP++ uses this description when you set up your debug session. Refer to your processor's Hardware Reference for information about the VisualDSP++ Configurator.

When running a multiprocessor simulator debug session, select the desired configuration from the **Platforms** list in the **New Session** dialog box. After specifying your hardware system, build your project.

The first time that you launch VisualDSP++ for a new project, the **New Session** dialog box opens to enable you to configure the MP session. The next time VisualDSP++ is launched, the debug session is automatically configured for you.

Debugging a Multiprocessor System

Debugging a multiprocessor system requires that you synchronously run, step, halt, and observe program execution operations in all the processors at once.

The following capabilities help to speed a multiprocessor debug session.

- Multiprocessor debug commands that operate like the commands used to debug a single processor. The only difference is that MP commands work synchronously on *all active processors* in the currently selected MP group

- **Multiprocessor** window

The **Status** page displays the status of each processor and lets you switch processor focus.

The **Group** page enables you to group processors into multiple, logical units to which all MP commands are applied.

- Window pinning. Note that you can use pinning and the processor status items in the **Multiprocessor** window with single-processor debug commands to debug individual processors in an MP session.
- Window color specification

Focus and Pinning

In a multiprocessor debug session, you often have to examine the behavior of a single processor to better understand its interaction with the other processors on the target.

Debug Sessions

When you debug a single processor in an MP session, the processor being debugged has the *focus*.

By *pinning a window* to a processor, you dedicate the window, such as a memory window, to a particular processor in a multiprocessor group. Pinning statically associates a window to a specific processor.

 Before debugging, open and pin the register windows and memory windows that you plan to use. If these windows are not pinned, they display information for any processor that has focus.

When a window is pinned to a processor, a pin icon appears in the window's upper-left corner.

For example:



Window Title Bar Information

Figure 3-1 shows a pinned window in a multiprocessor debug session.

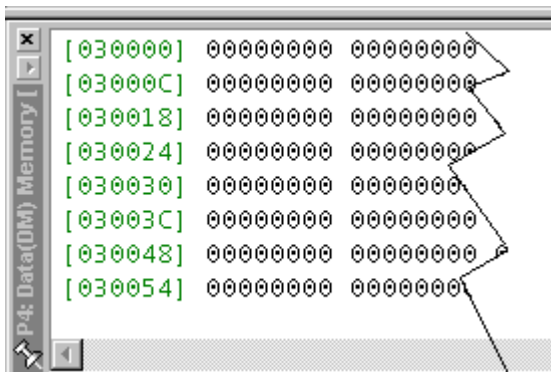


Figure 3-1. Pinned Window in a Multiprocessor Debug Session

The title bar of a pinned window shows:

- Processor name
- Pushpin icon () to indicate a pinned window
- Window title
- Number format, such as hexadecimal (for windows that support multiple formats)

Additional Focus Indication

If configured, VisualDSP++ shades unfocused windows with a specified color. You can specify the background color of focused and unfocused windows.

Code Analysis Tools

You use code analysis tools to examine your code's behavior and locate areas that may be optimized for better performance.

VisualDSP++ provides these code analysis tools:

- Statistical profiles and linear profiles
- Traces
- Processor memory plots

Statistical Profiles and Linear Profiles

VisualDSP++ provides two profiling methods that measure program performance by sampling the target's `Program Counter (PC)` register to collect data. During program development use linear profiling with simulator targets, and use statistical profiling with emulator targets.

Code Analysis Tools

The **Linear Profiling Results** window and **Statistical Profiling Results** window display the data collected by these two profiling methods and indicate where the application is spending its time.

The window's title (**Linear Profiling Results** or **Statistical Profiling Results**) depends on whether this tool is used during simulation or emulation.

Simulation

Linear profiling with the simulator is not statistical because the simulator samples *every* PC executed. This feature provides an accurate and complete picture of program execution.

Linear profiling is much slower than statistical profiling. Simulator targets support linear profiling but do not support statistical profiling.

Emulation

A statistical profile measures the performance of a DSP program by sampling the target's PC register at random intervals while the target is running the DSP program. Most of the execution time in the program is in the areas where most of the PC registers are concentrated.

Statistical profiling provides a more generalized form of profiling that is well suited to JTAG emulator debug targets. Emulator targets do not support linear profiling.

JTAG sampling is completely non-intrusive, so the process does not incur additional run-time overhead.

Traces

A trace captures a history of processor activity during program execution. Run a trace (also called an execution trace or a program trace) to analyze the run-time behavior of your DSP program, enable I/O capabilities, and simulate source to target data streaming.



TigerSHARC processors do not support traces.

A trace includes the following information.

- Buffer depth (instruction lines)
- Cycle count
- Instructions executed such as memory fetches, program memory writes, and data/memory transfers

Viewing the disassembled instructions that were performed can also help in analyzing code behavior.

DSP Memory Plots

You can display processor memory as a *plot* in a plot window (Figure 3-2).

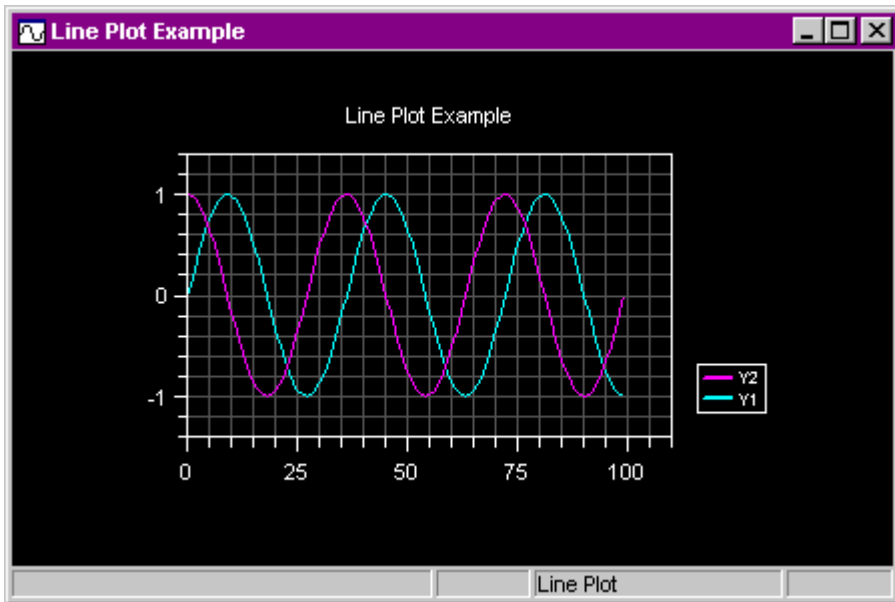


Figure 3-2. Plot Window Displaying Processor Memory

When plotting processor memory, you can:

- Choose from multiple plot types and specify the plot's data and presentation
- Apply a data processing algorithm to the processor's memory data
- Modify a plot's configuration and immediately view the results
- Zoom in on a plot area or view data point values, in a plot window
- Print a plot, save the plot image to a file, or save the plot's data to a file

Program Execution Operations

When VisualDSP++ starts up, it attaches to the previous session by default. You can override this behavior, and instead, force VisualDSP++ to start a new session.

When loading and running your program, use VisualDSP++ features to step, break, and halt the program.

Selecting a New Debug Session at Startup

If you had a problem, such as a corrupted workspace, in your last debug session, use the following procedure to force a fresh session at startup.

Note: VisualDSP++ must be closed before performing the following procedure.

1. Hold down the keyboard's **Ctrl** key.



Do not release the **Ctrl** key until the **Session List** dialog box appears, as described in the next step.

2. Invoke VisualDSP++ as you normally do.

Typical methods include using the Windows **Start** button sequences, clicking desktop icons, or using Windows Explorer.

The **Session List** dialog box appears.

3. Specify and activate a debug session.

When launching VisualDSP++ in standalone mode, ensure that the session is configured correctly *before* you load your program.

Loading the DSP Executable Program

Once you specify the debug session, begin the session by loading the DSP executable program.

After a successful build of the target executable program, VisualDSP++, if configured, loads the program automatically to the current session when the session processor type matches the project's processor. When the current session processor does not match the project's processor type, you are prompted to choose another session.

If automatic load is not configured, VisualDSP++ does not try to load the executable program automatically after a successful build.



The target must be an executable (.EXE) file.

This debugging feature saves time, as you do not have to load the executable target manually, and you can start to debug right after a successful build of the project.

Using Program Execution Commands

You can run program execution commands from the **Debug** menu or from the toolbar.

Executable files run until an event such as a breakpoint, watchpoint, or user-issued **Halt** command stops execution. When program execution halts, all windows are updated to current addresses and values.

Multiprocessor (MP) commands operate like single-processor commands with one exception—they perform an action on *all* processors in the MP group. MP **Run**, MP **Halt**, and MP **Step** are synchronous operations, which means that all processors in the currently selected group execute on exactly the same clock cycle. Some commands have keyboard shortcuts.

Use the commands described in [Table 3-2](#) to control program execution.

Table 3-2. Commands Used to Control Program Execution

Command	Description
Run	Runs an executable program. The program runs until an event stops it, such as a breakpoint or user intervention. When program execution halts, all windows update to current addresses and values.
Halt	Stops program execution. All windows are updated after the processor halts. Register values that changed are highlighted, and the status bar displays the address where the program halted.
Run to Cursor	Runs the program to the line where you left your cursor. You can place the cursor in editor windows and Disassembly windows.
Step Over	(C/C++ code only in an editor window) Single-steps forward through program instructions. If the source line calls a function, the function executes completely, without stepping through the function instructions.
Step Into	(editor window or Disassembly window) Single-steps through the program one C/C++ or assembly instruction at a time. Encountered functions are entered.
Step Out Of	(C/C++ code only in an editor window) Performs multiple steps until the current function returns to its caller, and stops at the instruction immediately following the call to the function.

Restarting the Program

You can set the `Program Counter` to the first address of the interrupt vector table.

Performing a Restart During Simulation

In the simulator, restart works like a reset; however, the target's memory does not change. All registers are reset to their initial values.



Memory is not reset. Thus, C and assembly global variables are *not* reset to their original values. Your program may behave differently after a restart. To reinitialize these values, reload your `.DXE` file.

Program Execution Operations

Performing a Restart during Emulation

In the emulator, restart works exactly like a reset. Only registers with default reset values are affected. All other registers remain unchanged.

Using Breakpoints

An enabled breakpoint halts program execution at a specific instruction or address. You can enable and disable breakpoints as well as add and delete breakpoints.

A disabled breakpoint is set up, but not turned on. A disabled breakpoint does not stop program execution. It is dormant and may be used later.

A *break* occurs when the conditions that you specify are met.

You can quickly place an unconditional breakpoint at an address in a **Disassembly** window or editor window by using one of these options:

- Select the address and click the **Toggle Breakpoint** button .
- Double-click on the line in the **Disassembly** or editor window.

Symbols in the left margin of a **Disassembly** window or editor window indicate breakpoint status, as shown in [Table 3-3](#).

Table 3-3. Breakpoint Status Symbols

Symbol	Indicates
	An enabled (set) breakpoint
	A disabled breakpoint (recognized, but cleared)

Using Unconditional and Conditional Breakpoints

A breakpoint configured to occur when the `Program Counter` reaches a specific address is an *unconditional breakpoint*. The breakpoint occurs when it is reached.

A breakpoint configured to occur when various conditions (criteria) are met is called a *conditional breakpoint*. The conditions may include:

- A user-defined *expression* that must evaluate to TRUE
- A *skip* (count) that specifies the number of times to skip over the breakpoint before finally halting

If both an expression and skip are set, execution stops when the breakpoint is reached “*n*” times when the expression is TRUE, where *n* represents the skip count. When the expression is empty, execution stops when the breakpoint is reached “*n*” times.

Using Watchpoints

Similar to breakpoints, watchpoints stop program execution when user-specified conditions are satisfied. Watchpoints, however, are used to set a *condition*, such as a memory read or stack pop, for halting events.



Use watchpoints only during simulation.

Watchpoints, unlike breakpoints, are not attached to a specific address. A watchpoint halts anywhere in your program once the watchpoint conditions are satisfied.

Using Hardware Breakpoints

Similar to simulator watchpoints, hardware breakpoints enable you to set breaks on instructions or data transfers within a user-defined memory range.

 Hardware breakpoints can be used only in TigerSHARC ADSP-TS101 and ADSP-TS201 emulator sessions.

Choosing **Hardware Breakpoints** from the **Settings** menu opens the **Hardware Breakpoints** dialog box, where you can configure the following attributes for three breakpoints (**BP0**, **BP1**, or **BP2**).

- **Mode** – type of breakpoint (Disabled, Single Address, In Range, Out of Range)
- **Start Address** and **End Address** – address range in which to break. Specified addresses are inclusive.
- **Access** – type of memory access to which the breakpoint responds (Read Only, Write Only, or In Range)
- **Counter** – number of breakpoint events ignored before the processor stops. Each time this breakpoint condition occurs, its count is decremented. When the count reaches zero (0), the processor processes the breakpoint and halts.
- **Restore Counter on Break** – resets the counter to the counter value each time the breakpoint is processed
- **AND Breakpoints, OR Breakpoints** – AND decrements the counter each time all breakpoints find a match in the same cycle. OR decrements the counter when one breakpoint finds a match.

Latency

Hardware breakpoints do not assert until one (1) or two (2) instruction cycles after the actual break condition occurs. Note that the program counter is not placed on the instruction that caused the break.

Restrictions

When using hardware breakpoints, do not place breaks at any address where a JUMP, CALL, or IDLE instruction would be illegal.

Do not place breaks in the last few instructions of a DO LOOP or in the delay slots of a delayed branch. For more information on these illegal locations, refer to your processor's Hardware Reference or User's Guide.

Simulation Tools

Before you have the processor, you can use interrupts and data streams within VisualDSP++ to simulate the processor's behavior.

Interrupts

Use interrupts to simulate external interrupts in your program. When you use interrupts with watchpoints and streams, your program simulates real-world operation of your DSP system.

Input/Output Simulation (Data Streams)

In many products, processors exist as part of a larger system where they can act as a host or a slave. They can drive other devices or take part in processing a subset of data. Because of their extensive I/O capabilities, Analog Devices processors excel in these roles.

Use data streams to transmit data between:

- A device and a file
- A device and a device
- A device in one processor and a device in another processor in a multi-processor system

Simulation Tools

Using background telemetry channel (BTC) technology, VisualDSP++ permits the streaming of data from a target processor without halting the processor. This capability applies to SHARC emulation targets only.

The plot window receives and displays a stream of data from processor memory. If the target supports background telemetry, the plot window reads memory and updates the display without halting the target. Otherwise, the plot window halts the processor, reads memory, updates the plot, and resumes the processor.

The plot window allows data to be streamed to (or from) a binary data file. The data file can be converted into ASCII format for input to other applications such as MATLAB and Excel.

The DSP application may collect and transfer data in four different ways:

- Sampling a test point over time
- Transferring a data array over BTC at a specified point in the DSP application (SHARC emulation only)
- Using `GetMem()` directly
- Periodically halting the target to read memory

For information about using BTC, see the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors*.

Image Viewer

The VisualDSP++ Image Viewer enables you to:

- View an image. You can display BMP, JPEG, PPM, or MPEG data from processor memory or from a file on your PC.
- Correct the gamma attributes of an image. For a color image, you can adjust the red, green, and blue pixel values. On a gray-scale image, you can adjust darkness only.
- Copy an image to the Windows clipboard
- Print an image or save it to a file
- Export an image

You select the image source (from processor memory or a file on your PC) and specify image attributes. If the image is located in processor memory, you must specify the image's address, size, and format.

For more information about Image Viewer, see [“Image Viewer” in Chapter 2, Environment](#).

Plots

VisualDSP++'s data plotting capability helps you visualize the data in the processor's memory.

Plot Types

Specify a plot as one of the plot types described in [Table 3-4](#).

The X, Y, and Z values are read from processor memory.

Plots

Table 3-4. Available Plot Types

Plot Type	Description	Requires
Line	Displays points connected by a line	Y value for each data point
X-Y	Similar to a line plot, but also uses X-axis data	X value and Y value for each data point
Constellation	Displays a symbol at each data point	X value and Y value for each data point
Eye diagram	Typically used to show the stability of a time-based signal	Y value for each data point
Waterfall	3-D plot typically used to show the change in frequency content of signal over time	Z value for each data point
Spectrogram	2-D plot displays amplitude data as a color intensity	Z value for each data point

Line Plots

A line plot (shown in [Figure 3-3](#)) displays a range of processor memory values connected by a line. The values read from processor memory are assigned to the Y-axis. The corresponding X-axis values are automatically generated.

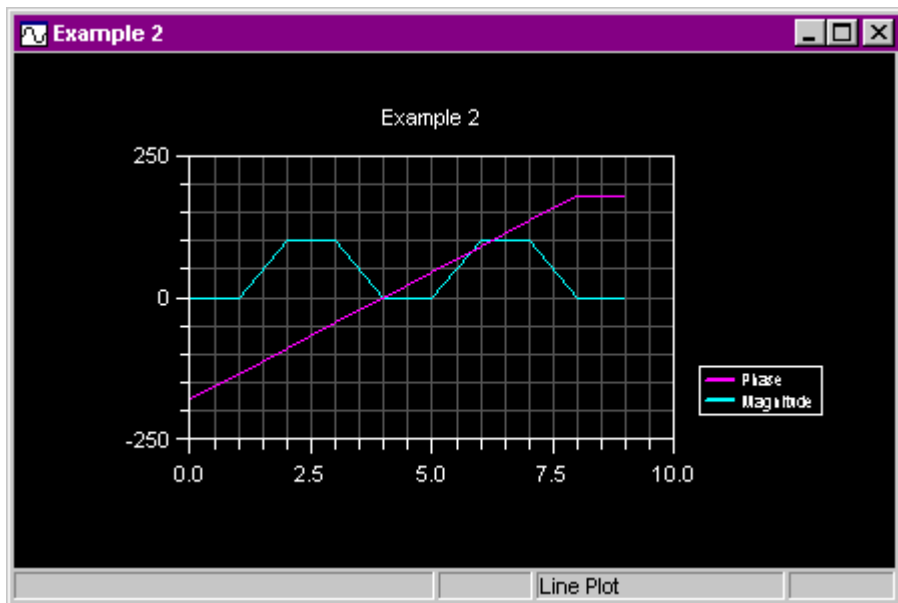


Figure 3-3. Line Plot

Multiple data sets can be plotted on a single graph.

Plots

X-Y Plots

An X-Y plot (shown in [Figure 3-4](#)) requires an X value and a Y value for each data point. Unlike a line plot, an X-Y plot requires the X-axis data.

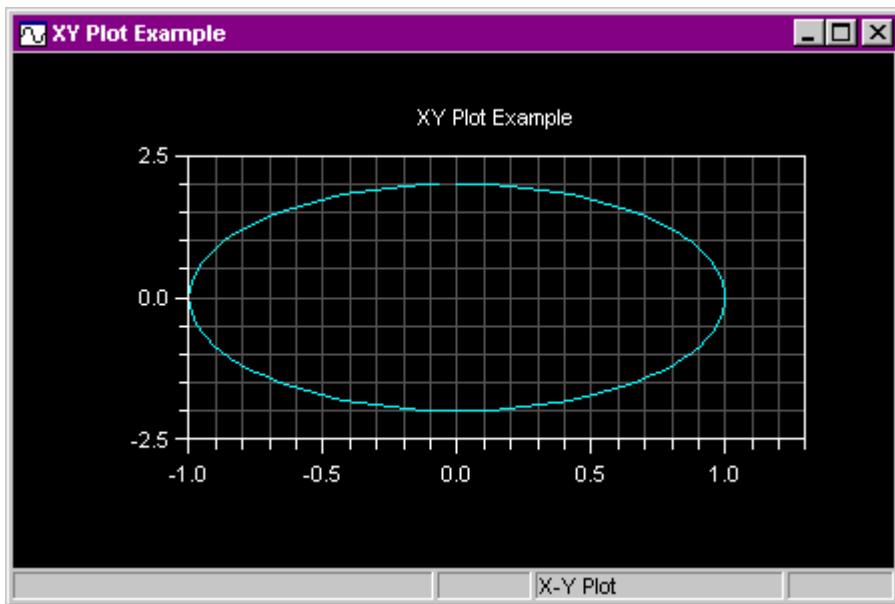


Figure 3-4. X-Y Plot

The X and Y data are specified separately in a user-defined memory location. The number of X and Y points must be equal.

Constellation Plots

A constellation plot (shown in [Figure 3-5](#)) displays a symbol at each (X,Y) data point.

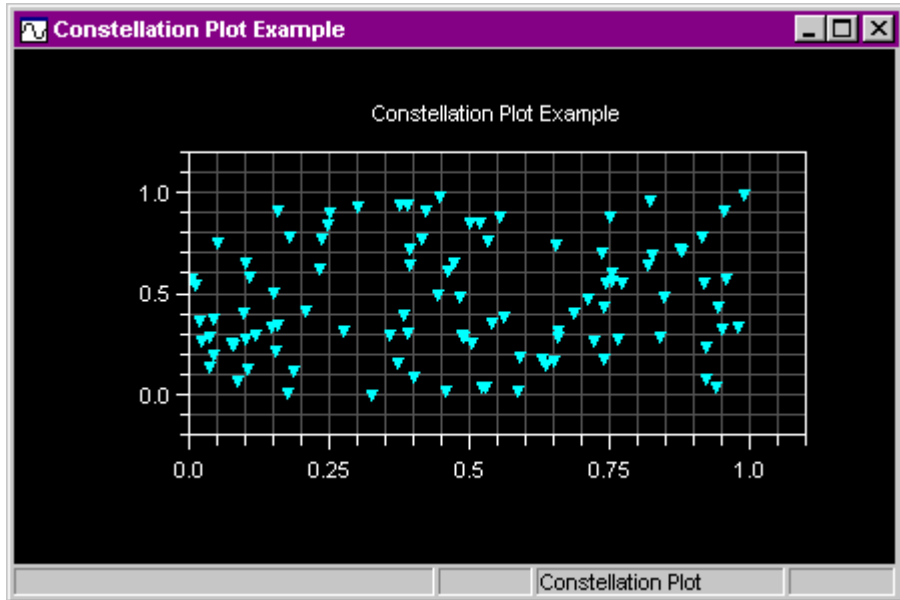


Figure 3-5. Constellation Plot

The X and Y data are specified separately in a user-defined processor memory location. The number of X and Y points must be equal.

Eye Diagrams

An eye diagram plot (shown in [Figure 3-6](#)) is typically used to show the stability of a time-based signal. The more defined the eye shape, the more stable the signal.

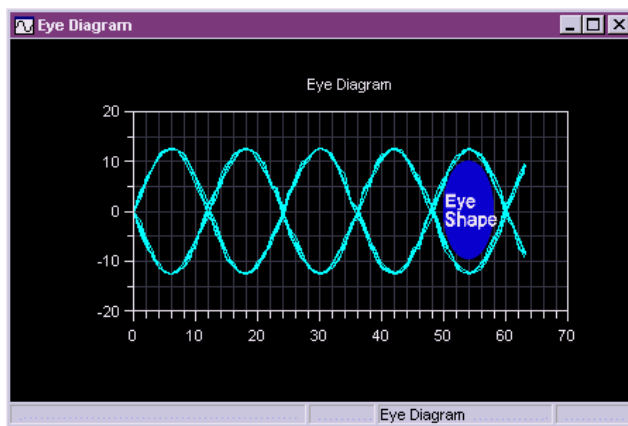


Figure 3-6. Eye Diagram Plot

This plot works like a storage oscilloscope by displaying an overlapped history of a time signal. The eye diagram plot processes the input data and optionally looks for a threshold crossing point (default is 0.0). A trace is plotted when the threshold crossing is reached. Plotting continues for the remainder of the trace data.

When a breakpoint occurs (or a step is performed), the plot data is updated and a new trace is plotted. The eye diagram uses a data shifting technique that stores the desired number of traces in a plot buffer (default is ten traces). When the number of traces is exceeded, the first trace shifts out of the buffer and the new trace shifts into the last buffer location. This technique is referred to as *first in, first out* (FIFO).

You can modify options for threshold value, rising trigger, falling trigger, and the number of overlapping traces.

Waterfall Plots

A waterfall plot (shown in [Figure 3-7](#)) is typically used to show the change in frequency content of signal over time.

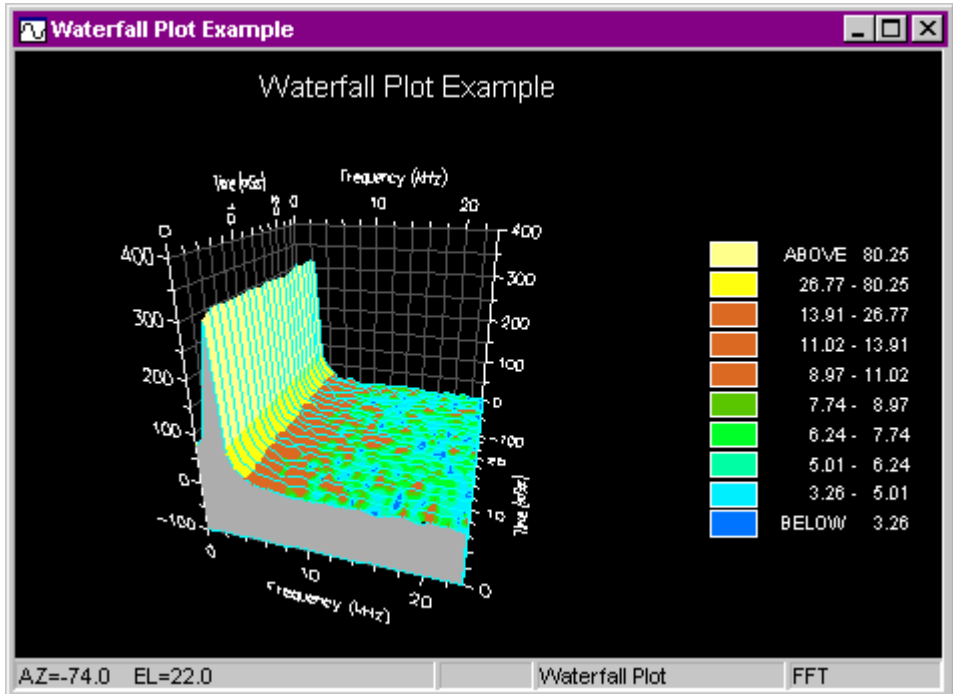


Figure 3-7. Waterfall Plot

The plot comprises multiple line plot traces in a 3-D view. Each line plot trace represents a slice of the waterfall plot.

The easiest way to create a waterfall plot is to define a 2-D array in C code (a grid). The first array dimension is the number of rows in the grid, and the second dimension is the number of columns in the grid. The number of columns is equal to the number of data points in each line trace.

Plots

A time-based signal is sampled at a predefined sampling rate and stored as a slice in the grid (row 0, columns 0– N).

Figure 3-8 shows a grid of sampled data.

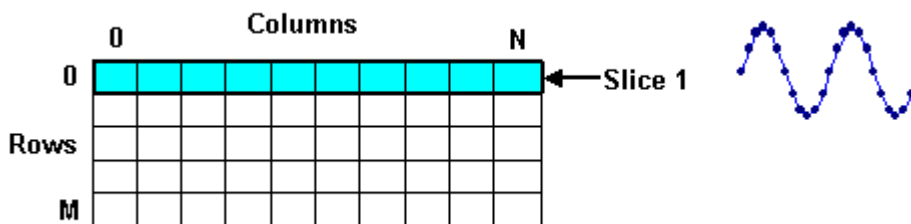


Figure 3-8. Grid of Sampled Data

The next time, a signal is sampled and stored (in row 1, columns 0– N). This process continues until all the rows are filled.

By default, an FFT performed on each slice results in a frequency output display. Use a color map on the **3-D Axis** page of **Color Settings** dialog box to enhance the display. Each color corresponds to a range of amplitude values.

The plot output displays a legend showing each color and associated range of values.

You can rotate the waterfall plot to any desired azimuth and elevation by using the keyboard's arrow keys.

Spectrogram Plots

A spectrogram plot (shown in [Figure 3-9](#)) displays the same data as a 3-D waterfall plot, except in 2-D format.

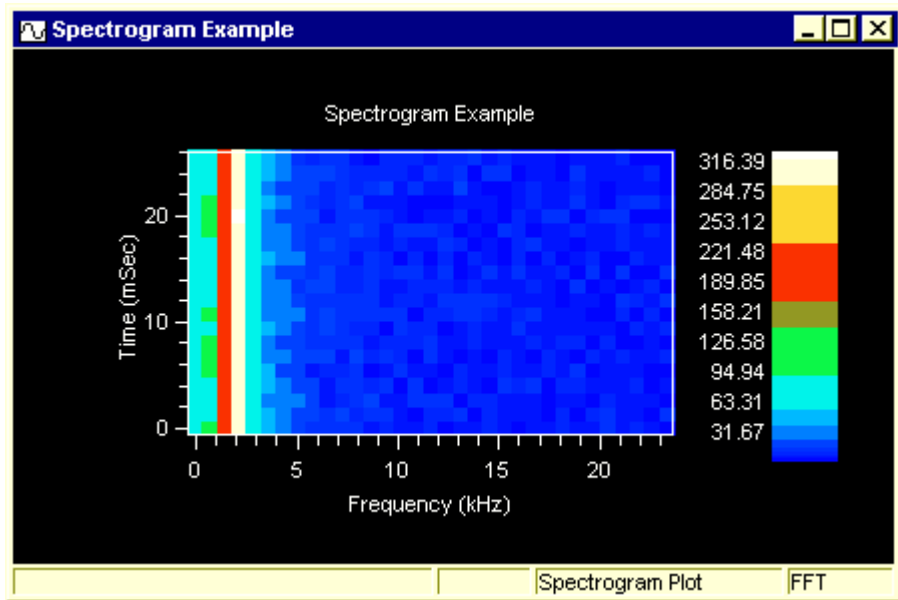


Figure 3-9. Spectrogram Plot

Each (X,Y) location displays as a color representing the amplitude of the data. By default, an FFT performed on each slice results in a frequency output display. A legend displays the colors and associated range of values.

Flash Programmer

The VisualDSP++ Flash Programmer provides a convenient, generic interface to numerous processors and flash memory devices. This utility simplifies the process of changing data values on a flash device and modifying its memory. You no longer have to remove the flash memory from the board, use a separate Flash Programmer, and then replace the flash.

Flash Devices

Flash memory parts are non-volatile memories that can be read, programmed, and erased. In most applications, flash devices store:

- Boot code that the processor loads at startup
- Data that must persist over time and through the loss of power

Flash device programming is typically performed with a device programmer at the factory or by the application developer. When a flash device is wired appropriately to the processor, the processor can program the flash device.

Flash Programmer Functions

Use the Flash Programmer to:

- Load a flash algorithm (driver) program onto the processor at any time
- Obtain the flash manufacturer and device codes
- Reset the flash
- Program the flash from an Intel Hex data file
- Fill portions of flash memory with a value and quickly “punch-in” data
- Erase the entire flash

- Erase a single sector
- Send custom commands to the driver for batch processes or user-defined behavior

The utility stores the most recently used information in the registry for retrieval when the utility is next started up. A status indicator shows the utility's current state.

Flash Driver

To use the Flash Programmer utility, you must first load a flash driver onto the processor. The driver is a DSP application that interfaces with the Flash Programmer and performs all the interaction with the flash device. Analog Devices supplies sample drivers for use on certain EZ-KIT Lite evaluation systems.

Flash Programmer Window

Figure 3-10 shows the Flash Programmer window.

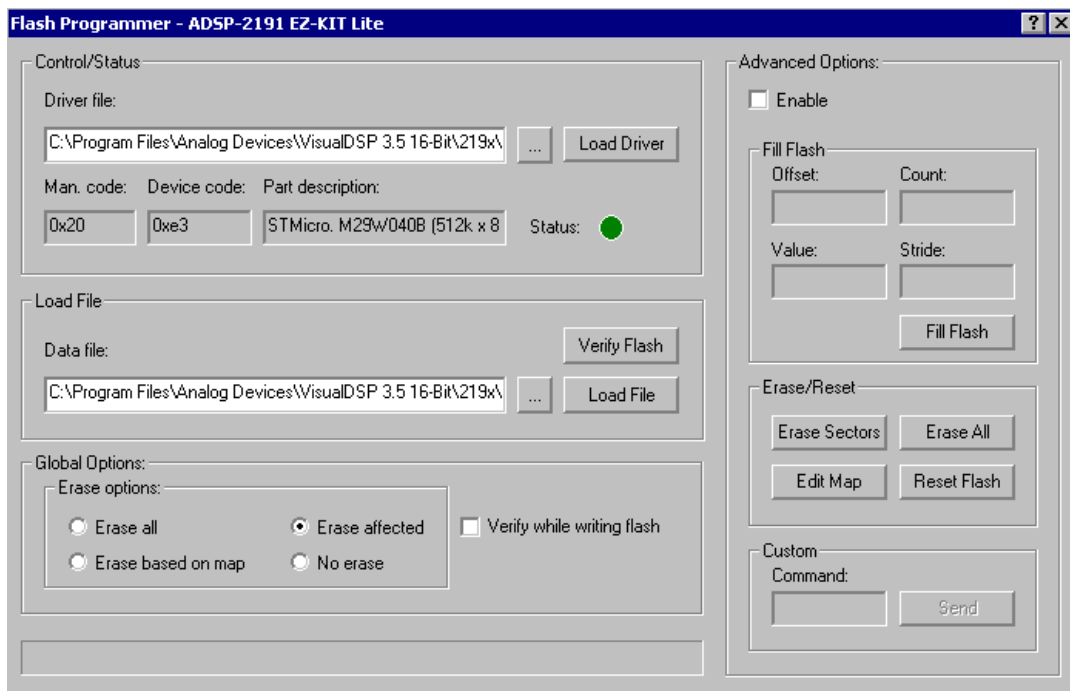


Figure 3-10. Flash Programmer Window

Table 3-5 describes the fields and buttons in the **Flash Programmer** window.

Table 3-5. Flash Programmer Window Controls

Control	Description
Control/Status	
Driver file	Specifies the path and name of the driver file. Type the path and file name or browse to select the driver.
Load Driver	Loads the specified driver onto the processor
Man. code	Specifies the flash memory's manufacturer code. Load the driver to view this data.
Device code	Displays the flash memory's device code. Load the driver to view this data.
Part description	Displays the flash memory's part description. Load the driver to view this data.
Status	Displays the utility's current status Red – The utility is not ready. Load a driver. Green – The utility is ready to process a command. Yellow – The utility is busy processing a command.
Load File	
Data file	Specifies the data file. Type the path and file name or browse to select the file. Note: Only valid Intel Hex files may be used. The VisualDSP++ loader produces files in this format.
Verify Flash	Compares the flash's current contents against an Intel Hex data file. This procedure assumes that a driver has been loaded.
Load File	Loads the specified data file onto the flash memory device
Global Options	
Erase all	Erases all of the device's memory
Erase based on map	Erases based on a sector map
Global Options (Cont'd)	
Erase affected	Erases only the portion of flash affected by the write

Table 3-5. Flash Programmer Window Controls (Cont'd)

Control	Description
No erase	Does not erase flash
Verifying while writing flash	Verifies each performed write by ensuring that what was written matches what is read back from flash
Advanced Options	
Enable	Enables advanced features Selecting this control enables the fields and buttons below it. Clearing this control disables (grays out) the fields and buttons.
Offset	Specifies the first address at which to place data
Value	Specifies the data value to be written
Count	Specifies the number of locations to be written
Stride	Specifies the number of locations to skip between each write. Typically, this is 0x1. Entering 0x2 specifies every other location.
Fill Flash	Loads the specified data value onto the flash memory device
Erase Sector	Erases the specified sector from the flash device
Edit Map	Opens the Sector Map dialog box, which enables you to select the sector (or sectors) you want to erase. This function assumes that a driver has been loaded. Note: Erasure is constricted by the selected Global Erase option.
Erase All	Erases the flash device's entire memory
Reset Flash	Resets the internal state of the flash device and places it into read mode without modifying its contents
Command	Specifies the custom command to be run
Send	Sends the specified custom command to the driver. The value entered in Command is interpreted as a hexadecimal value; for example, 10 is interpreted as 10 hexadecimal or 16 decimal.

A REFERENCE INFORMATION

This appendix contains reference information to help you understand VisualDSP++ and speed up DSP program development. The information is organized as follows.

- [“Glossary” on page A-2](#)
- [“File Types” on page A-24](#)
- [“Keyboard Shortcuts” on page A-27](#)
- [“IDDE Command-Line Parameters” on page A-33](#)
- [“Extensive Scripting” on page A-34](#)
- [“Toolbar Buttons” on page A-38](#)
- [“Text Operations” on page A-43](#)
- [“Online Help Features and Operations” on page A-48](#)

Glossary

The following terms are important toward understanding VisualDSP++.

Application Programming Interface (API) functions

A set of functions available to an applications programmer. These functions, which are part of an application, can be accessed by other applications. For VDK, API refers to a library of C/C++ functions and assembly macros that define VDK services. These services are essential for kernel-based application programs. The services include interrupt handling, thread management, and semaphore management.

archiver

The VisualDSP++ archiver, `elfar.exe`, combines object (`.OBJ`) files into library (`.DLB`) files, which serve as reusable resources for project development. The linker searches library files for routines (library members) that are referred to by other objects, and links them in your executable program.

breakpoint

User-defined halt in an executable program. Toggle breakpoints (turn them on or off) by double-clicking on a location in a **Disassembly** window or editor window.

break condition

Hardware condition under which the target breaks and returns control of the target back to the user. For example, a break condition could be set up to occur when address `0x8000` is read from or written to.

build

Performing a build (or project build) refers to the operations (pre-processing, assembling, and linking) that VisualDSP++ performs on projects and files. During a build, VisualDSP++ processes the files in the project that have been modified (or depend on files that have been modified) since the previous build. A build differs from a rebuild all. During a rebuild all, VisualDSP++ processes all the files in the project, regardless of whether they have been modified.

build type

Replaced by “configuration”

channel

A transmission path between two communicating locations, usually the smallest subdivision of a transmission system. For VDK, channel refers to a FIFO queue into which messages sent to a thread are placed. Each thread has 15 channels. Messages are received in priority order from the lowest numbered channel to the highest.

COFF

Common Object File Format. VisualDSP++ does not support COFF formatted files.

configuration (or project configuration)

A project is developed in stages (configurations). By default, a project includes two configurations: Debug and Release. A configuration refers to the collection of options (tool chain and individual options for files) specified for the configuration. You can add a configuration to your project at any time. You can delete a customized configuration that you created, but you cannot delete the Debug or Release configurations.

context switch

A process of saving/restoring the processor's state. The scheduler performs the context switch in response to the system change.

A hardware interrupt can occur and change the state of the system at any time. Once the processor's state has changed, the currently running thread may be swapped with a higher-priority thread. When the kernel switches threads, the entire processor's state is saved and the processor's state for the thread being switched in is restored.

critical region

A sequence of instructions whose execution cannot be interrupted or swapped out. Suspending all interrupt service routines (ISRs) before calling the critical region ensures that the execution of a critical region is not interrupted. Once the critical region routine concludes, ISRs are enabled.

CROSSCORE®

Analog Devices DSP development tools, which provide easier and more robust methods for engineers to develop and optimize systems by shortening product development cycles for faster time-to-market. CROSSCORE components include the VisualDSP++ software development environment and EZ-KIT Lite evaluation systems and emulators for rapid on-chip debugging.

current directory

Directory where the .DPJ file is saved. The build tools use the current directory for all relative file path searches. See also “default directories.”

data set

A series of data values in processor memory used as input to a plot. You can create data sets and configure the data for each data set. You specify the memory location, the number of values, and other options that identify the data. Additional specifications for row and column counts are required for 3-D plots.

Debug configuration

For a debug configuration, you can accept the default options or specify your own options and save them. The configuration refers to the specified options for all the tools in the tool chain. See also “configuration.”

debug session

The combination of a target and a platform. For example, a session can be a JTAG emulator target connected to a platform consisting of five processor. Another example of a debug session is an ADSP-21262 EZ-KIT Lite target connected to an ADSP-21262 EZ-KIT Lite board.

The DSP projects being developed are run as debug sessions. The two types of sessions are hardware and software. The processor, target, and platform define the session. When setting up a session, set the focus on a series of more specific elements.

debug target

The communication channel between VisualDSP++ and a processor (or group of processors). Targets include simulators, emulators, and EZ-KIT Lite evaluation systems. Several targets may be installed on your system. Simulator targets, such as the ADSP-TS101 cycle-accurate simulator, differ from emulator targets in that the processor exists only in software.

Glossary

The Summit-ICE emulator communicates with one or more physical devices over the host PC's PCI bus. The Apex-ICE emulator communicates with a device through the PC's USB port.

default intermediate and output file directories

These file directories (folders) are `\Debug` (for the debug configuration) and `\Release` (for the release configuration). By default, VisualDSP++ creates these directories as children of the directory where the `.DPJ` file is saved, which is called the project's current directory. See also "current directory."

dependencies

VisualDSP++ uses dependency information to determine which files, if any, are updated during a build. If an included header file is modified, VisualDSP++ builds the source files that include (`#include`) the header file, regardless of whether the source files have been modified since the previous build.

dependency files

Usually user files or system header (`*.H`) files, these files are referenced from a source file by a preprocessor `#include` command.

device

A single processor. With regard to JTAG emulation and the JTAG EZ-ICE Configurator, a device refers to any physical chip in the JTAG chain.

device driver

A user-written model that abstracts the hardware implementation from the application code. User code accesses device drivers through a set of device driver API functions.

DWARF-2

Format for debugging source-level assembly code via improved line and symbol information

editor window

A document window that displays a source file for editing. When an editor window is active, you can move about within the window and perform typical text editing activities such as searching, replacing, copying, cutting, pasting, and so on.

ELF

Executable Linking Format

emulator

Hardware used to connect a PC to a processor target board. This hardware allows application software to be downloaded and debugged from within the VisualDSP++ environment. Emulator software performs the communications that enable you to see how your DSP code affects processor performance.

event

A signal (similar to a semaphore or message) used to synchronize multiple threads in a system. An event is a logical switch, having two binary states (available/true and unavailable/false) that control thread execution. When an event becomes available, all pending (waiting) threads in the wait list are set to a ready-to-run state. When an event is available and a thread pends on it, the thread continues running and the event remains available.

To facilitate error handling, threads can specify a timeout period when pending on an event.

Glossary

An event is a code object of global scope, so any thread can pend on any event. Event properties include the EventBit mask, Event-Bit value, and combination type. Events are statically allocated and enumerated at run time. An event cannot be destroyed, but its properties can be changed (see Blueelem text).

event bit

A flag set or cleared to post the event. The event is posted (available) when the current values of the system Event Bits match the event bit's mask and event bits' values defined by the event's combination type.

A system has only one Event Bits word, the size of a data word minus one. For ADSP-21xxx and ADSP-TSxxx processors, the size is 31 bits.

executable file

A file or program written and built in VisualDSP++

EZ-KIT Lite

A development board, software, and cable for evaluating a particular processor. The kit includes fundamental debugging software to facilitate architecture evaluations via a PC-hosted tool set. Use the kit to evaluate Analog Devices processors, learn about DSP applications, simulate and debug applications, and prototype applications.

focus

Refers to the active processor in a multiprocessor (MP) debugging session

ICE

In-Circuit Emulator. Analog Devices offers emulators that provide non-intrusive target-based debugging of DSP systems. An emulator can single-step or execute a processor at full speed to facilitate viewing or altering a processor's register and memory contents.

IDDE

Integrated Development and Debugging Environment for Analog Devices DSP development tools

interrupt

An external or internal condition detected by the hardware interrupt controller. In response to an interrupt, the kernel processes a subroutine call to a predefined interrupt service routine (ISR).

Interrupts have the following specifications.

Latency – interrupt disable time. The period between the interrupt occurrence and the first ISR's executed instruction.

Response – interrupt response time. The period between the interrupt occurrence and a context switch.

Recovery – interrupt recovery time. The period needed to restore the processor's context and to start the return-from-interrupt (RTI) routine.

interrupt service routine (ISR)

A routine executed as a response to a software interrupt or hardware interrupt. VDK supports nested interrupts, which means that the kernel recognizes other interrupts, services interrupts, or both with higher priorities while executing the current ISR. For VDK, the ISRs are written in assembly language. VDK reserves the timer and the lowest priority (reschedule) interrupt.

JTAG

Joint Test Action Group. This committee is responsible for implementing the IEEE boundary scan specification, enabling in-circuit emulation of ICs.

JTAG ICE configurator

See “VisualDSP++ configurator.”

kernel

The main module of a real-time operating system. The kernel loads first and permanently resides in the main memory and manages other modules of the real-time operating system. Typical services include context switching and communication management between OS modules.

keyboard shortcuts

The keyboard provides a quick means of running the commands used most often, such as simultaneously typing the keyboard’s **Ctrl** and **G** keys (indicated with the symbols **Ctrl+G**) to go to a line in a file.

librarian

A utility that groups object files into library files. When linking your program, specify a library file and the linker automatically links any file in the library that contains a label used in your program. Source code is provided so you can adapt the routines to your needs.

library files

The VisualDSP++ archiver, `elfar.exe`, combines object (`.DOJ`) files into library (`.DLB`) files, which serve as reusable resources for project development. The linker searches library files for routines (library members) that are referred to from other objects, and links them into your executable program.

linear profiling

A debugging feature that samples the target's PC register at every instruction cycle. Linear profiling gives an accurate picture of where instructions were executed, since every PC value is collected. The trade-off, however, is that linear profiling is much slower than statistical profiling. A display of the resulting samples appears in the **Linear Profiling Results** window, which graphically indicates where the application is spending its time. Simulator targets support linear profiling. See also "Statistical profiling."

linker

The linker creates executable files, shared memory files, and overlay files from separately assembled object and library files. It assigns memory locations to code and data in accordance with a user-defined `.LDF` file, which describes the memory configuration of the target system.

Linker Description Files (LDFs)

LDFs describe the target system and map your program code within the system memory and processors. The `.LDF` file creates an executable file using the target system memory map and defined segments in your source files.

loader

A utility that transforms an executable file into a boot file. The loader creates a small kernel, which is booted into internal memory at chip reset. A program of arbitrary size can then be loaded into the processor's internal and external memory.

makefile

VisualDSP++ can export a makefile (make rule file), based on your project options. Use a makefile (.MAK) to automate builds outside of VisualDSP++. The output make rule is compatible with the gnu-make utility (GNU Make V3.77 or higher) or other make utilities.

memory pool

An area of memory containing a specified number of uniformly sized blocks of memory available for allocation and subsequent use in an application. The number and size of the blocks in a particular memory pool are defined at pool creation.

message

For VDK, a signal (similar to an event or semaphore) used to synchronize two threads in a system or to communicate information between threads. A message is sent to a specified channel on the recipient thread (and can optionally pass a reference to a payload to facilitate the transfer of data between threads). Posting a message takes a deterministic amount of time and may incur a context switch.

mixed mode

One of the two editor window display formats (the other being source mode). Mixed mode displays assembled code after the line of the corresponding C code.

multiprocessor system

A system built with multiple processors. Often, performance-based products require two or more processors. A system built with a single processor is called a *single-processor system*. Debugging a multiprocessor system requires that you synchronously run, step, halt, and observe program execution operations in all the processors at once. The SHARC simulator does not support this capability.

non-bootable PROM-image file

Splitter output, consisting of PROM files that cannot be used to boot-load a system

outdated file

A file that has been edited since the last build

payload

For VDK, an arbitrary amount of data associated with a message. A reference to the payload can be passed between threads as part of a message to enable the recipient thread to access the data buffer that contains the payload.

pinning a window

A technique that statically associates a window to a specific processor

pipelining

A feature that helps you analyze and tune your code for optimal performance. For TigerSHARC processors, VisualDSP++ provides a simulation-only debugging window (**Pipeline Viewer**) to help you visualize the pipeline by displaying pipeline stalls and aborts. For SHARC processors, the **Disassembly** window displays symbols (F, D, or E) to indicate an instruction's pipeline stage.

platform

The device with which a target communicates. For simulation, a platform is typically one or more processors of the same type. For emulation, you specify the platform with the VisualDSP++ configurator, and the platform can be any combination of devices.

The platform represents the hardware upon which one or more devices reside. You typically define a platform for a particular target. For example, if three emulators are installed on your system, a platform selection might be emulator two.

Several platforms may exist for a given debug target. For a simulator, the platform defaults to the identical DSP simulator. When the debug target is a JTAG emulator, the platforms are the individual JTAG chains. When the debug target is an EZ-KIT Lite board, the platform is the board in the system on which you wish to focus.

pre-emptive kernel

A priority-based kernel in which the currently running thread of the highest priority is pre-empted, or suspended, to give system resources to the new highest-priority thread

processor

An individual chip contained on a specific platform within a target. When you create the executable file, the processor is specified in the Linker Description File (.LDF file) and other source files.

profile-guided optimization (PGO)

A process that involves setting up and executing data sets to produce an optimized application. A *data set* is the association of zero or more input streams with one .pgo output file.

profiling

A technique used during simulation to examine program execution within selected ranges of code. Profiling helps you determine: percentage of time spent executing instructions, number of clock cycles spent executing instructions, number of instructions executed, and the number of times memory is read or written.

The profiler is non-intrusive. It does not report on execution within a called function (daughter function). Use profiling to monitor program memory. By watching one or more profile ranges, you can find areas of code that may be optimized for better performance. A profile session must include one memory range at a minimum. For each range, specify a start and end address. You can use symbols or hexadecimal numbers to represent addresses.

project

This term refers to the collection of source files and tool configurations used to create a DSP program. Through a project, you can add source files, define dependencies, and specify build options related to producing your output executable program. A project (.DPJ) file stores your program's build information.

VisualDSP++ helps you manage projects from start to finish in an integrated user interface. Within the context of a DSP project, you define project and tool configurations, specify project-wide and individual file options for debug or release modes of project builds, and create source files. VisualDSP++ facilitates easy movement among editing, building, and debugging activities.

project configuration

This configuration includes all of the settings (options) for the tools used to build a project.

Glossary

Project file tree display

See “Project window.”

Project window

This window displays your project’s files in a *tree view*, which can include folders to organize your project files. Right-clicking on an icon (the project itself, a folder, or a file) opens a menu of actions that you can perform on the selected item. Double-clicking on the project icon or a folder icon opens or closes the tree list.

Double-clicking a file icon opens the file in an editor window.

real-time operating system (RTOS)

A software executive that handles DSP algorithms, peripherals, and control logic. The RTOS comprises these components: kernel, communication manager, support library, and device drivers. An RTOS enables structured, scalable, and expandable DSP application development while hiding OS complexity.

rebuild all

See “build.”

registers

For information on available registers, see the corresponding processor documentation or view the associated online Help.

Release configuration

You can accept the default set of options, or you can specify the options you want and save them. The configuration refers to the specified options for all the tools in the tool chain. See also “configuration.”

reset

This command resets the processor to a known state and clears processor memory.

restart

This command sets your program to the first address of the interrupt vector table. Unlike a reset, a restart does not reload memory.

right-click

This action opens a right-click menu (sometimes called a context menu, pop-up menu, or shortcut menu). The commands that appear depend on the context (what you are doing). Right-click menus provide access to many commonly used commands.

round-robin scheduling

For VDK, a scheduling scheme whereby all threads at a given priority are given processor time automatically in fixed duration intervals. Round-robin priorities are specified at build time.

scheduler

For VDK, a kernel component responsible for scheduling system threads and interrupt service routines. VDK is a priority-based kernel in which the highest-priority thread is executed first.

scripting

You can interact with the IDDE by using a single command or a script file. Scripting languages include VBScript, JavaScript, and Tcl. Output displays in the **Console** view of the **Output** window. The output is also logged to the `VisualDSP_log.txt` file.

Glossary

semaphore

For VDK, a signal (similar to an event or message) used to synchronize multiple threads in a system. A semaphore is a data object whose value is zero or a positive integer (limited by the maximum setup at creation time). The two states (available/greater than zero and unavailable/zero) control thread execution. Unlike an event, whose state is automatically calculated, a semaphore is directly manipulated. Posting a semaphore takes a deterministic amount of time and may incur a context switch.

serial port data

You can automatically transfer serial port (SPORT) data to and from on-chip memory by using DMA block transfers. Each serial port offers a time division multiplexed (TDM) multichannel mode.

session

See “debug session.”

session name

Although the choice of target, platform, and processor define the session, you may want to further identify the session. To prevent confusion later, modify the default session name when you first create the debug session. A session name can be any string and can include space characters. There is no limit to the number of characters in a session name, but the **Session List** dialog box can display about 32 characters.

shortcuts

See “keyboard shortcuts.”

signal

For VDK, a method of communicating between multiple threads. VDK supports four types of signals: semaphores, events, messages, and device flags.

simulator

The simulator is software that mimics the behavior of a DSP chip. Simulators are often used to test and debug DSP code before the DSP chip is manufactured.

The way a simulator runs an executable program in software is similar to the way a processor does in hardware. The simulator also simulates the memory and I/O devices specified in the `.LDF` file. VisualDSP++ lets you interactively observe and alter the data in the processor and in memory. The simulator reads executable files. A simulator's response time is slower than that of an emulator.

source files

The C/C++ language and assembly language files that make up your project. Other source files that a project uses, such as the `.LDF` file, contain command input for the linker and dependency files (data files and header files). View source files in editor windows.

source mode

One of the two editor window display formats (the other being mixed mode). Source mode displays C code only.

splitter

A PROM splitter utility that transforms an executable file into a non-boot-loadable image. This file is loaded onto external processor memory.

statistical profiling

A debugging feature that provides a more generalized form of profiling that is well suited to JTAG emulator debug targets. With statistical profiling, VisualDSP++ randomly samples the target processor's program counter (PC) and presents a graphical display of the resulting samples in the **Statistical Profiling Results** window. This window graphically indicates where the application is spending time.

JTAG sampling is completely non-intrusive, so the process does not incur additional run-time overhead. See also "linear profiling."

stepping

A technique for moving through source or assembly code to observe instruction execution

streams

A debug tool used during simulation to drive other devices or take part in processing a subset of data. Use streams to simulate data input and output.

symbols

Labels for sections, subroutines, variables, data buffers, constants, or port names. For more information, refer to the related build tool documentation.

system configurator

For VDK, the system configuration control is accessible from the **Kernel** page of the **Project** window. The **Kernel** page provides a graphical representation of the data contained in the `vdk.h` and `vdk.cpp` files.

target

See “Debug target.”

threads

For VDK, a kernel system component that performs a predetermined function and has its own share of system resources. VDK supports multithreading, a run-time environment with concurrently executed independent threads.

Threads are dynamic objects that can be created and destroyed at runtime. Thread objects can be implemented in C, C++, or assembly language. A thread's properties include an ID, priority, and current state (wait, ready, run, or interrupted). Each thread maintains its own C/C++ stack.

ticks

The system-level timing mechanism. Every system tick is a timer interrupt.

tool chain

The collection of tools (utilities) used to build a project configuration

trace

Provides a history of program execution. A trace is sometimes called an execution trace or a program trace. Trace results show how the program arrived at a certain point and show program reads, writes, and memory fetches. TigerSHARC processors do not support traces.

unscheduled regions

For VDK, a sequence of instructions whose execution can be interrupted, but cannot be swapped out. The kernel acknowledges and services interrupts when an unscheduled region routine is running.

VDK

See “VisualDSP++ Kernel (VDK).”

VisualDSP++

An Integrated Development and Debugging Environment (IDDE) for Analog Devices DSP development tools

VisualDSP++ Configurator

Previously called JTAG ICE Configurator, use this utility to describe the hardware to VisualDSP++ when connecting to a JTAG emulator session. VisualDSP++ requires this description to set up the debug session.

VisualDSP++ Kernel (VDK)

The RTOS kernel from Analog Devices, a software executive between DSP algorithms, peripherals, and control logic. The kernel is integrated with the Integrated Development and Debugging Environment (IDDE), assembler, compiler, and linker programs into the DSP development tool chain.

VDK is supported on the SHARC and TigerSHARC processors. Refer to the *VisualDSP++ 3.5 Kernel (VDK) User's Guide for 32-Bit Processors* for details.

watchpoints

For simulation only. Similar to breakpoints, watchpoints stop program execution. Unlike breakpoints, which are attached to specific addresses, watchpoints are attached to user-defined conditions, such as memory reads or stack pops. The program halts when the conditions are met. SHARC processors do not support watchpoints.

workspace

You can open multiple windows, arrange them in any configuration, and save the layout as a workspace setting that can be recalled (loaded) at a later time. Each debug session's default workspace is automatically saved when you close the session and is automatically restored when you load that session.

By default, each session includes two workspaces. You can create any number of workspaces and switch among them. Suggested workspaces include an edit workspace, a debug workspace, and a plot workspace.

File Types

Table A-1 describes the files used to build a project.

Table A-1. Files Used With VisualDSP++

Extension	Name	Purpose
.ASM	Assembly source file	Source file comprising assembly language instructions
.C	C source file	Source file comprising ANSI standard C code and Analog Devices extensions
.CPP .CXX .HPP .HXX	C++ source file	Preprocessed compiler files that are inputs to the C/C++ compiler. These files comprise ANSI standard C++ code.
.DPJ	Project file	Contains a description of how your source files combine to build an executable program
.LDF	Linker Description File	Linker command source file is a text file that contains commands for the linker in the linker's scripting language
.IS .PP .S	Intermediate files	Preprocessed assembly files generated by the preprocessor
.DOJ	Assembler Object file	Binary output of the assembler
.DLB	Archiver file	Archiver's binary output in ELF format
.H	Header file	Dependency file used by the preprocessor, and a source file for the assembler and compiler
.H	Loader output	For ADSP-2192-12 processors, the C-language header file output of the boot loader utility (BLU)
.DAT	Data file	Dependency file used by the assembler for data initialization

Table A-1. Files Used With VisualDSP++ (Cont'd)

Extension	Name	Purpose
.DLO .DXE .OVL .SM	Debugging files	Binary output files from the linker in ELF/DWARF format
.MAP	Linker Memory Map file	Optional output for the linker. This text file contains memory and symbol information for executable files.
.TCL .TC8	Tools Command Language files	Tcl scripting language files used to script work
.OBJ	Assembled Object file	(Previous releases only, replaced by .DOJ) Output of the assembler
.LST	Listing file	Optional file output by the assembler
.BNM .H .LDR	Loader format files	The loader's output in ASCII format. Different varieties exist. Used to create boot PROMS.
.H_# .S_# .STK	PROM format files	The loader's output in ASCII format. Different varieties exist. Used to create boot PROMS.
.ACH	Architecture file	(Previous releases only, replaced by .LDF)
.TXT	Linker Command-Line file	(Previous releases only, replaced by .LDF) ASCII text file that contains command-line input for the linker
.EXE	Debugging file	(Used in previous releases, replaced by .DXE)
.EXE	Compiled simulation file	Enables faster execution speed compared to a standard .DXE program
.VDK	VisualDSP++ Kernel Support file	Enables VDK support
.JS .VBS	Script files	Enable you to script work for test applications. Scripting languages let you access the Automation API to interact with the IDDE.

File Types

Table A-1. Files Used With VisualDSP++ (Cont'd)

Extension	Name	Purpose
.DSP	Assembly source file	Source file comprising assembly language instructions
.MAK .MK	Makefiles	The output make rule file is used for project builds
.DPG	Project group	An .XML file containing information about projects
.IDL	VCSE input	VCSE Interface Definition Language (VIDL) specification
.XML	Manifest	Used by New Component Package Wizard when packaging a component for distribution

Keyboard Shortcuts

VisualDSP++ includes keyboard shortcuts (also called shortcut keys) for commonly used operations. These keyboard shortcuts appear in the tables below. You can also run commands by:

- Choosing a command from a drop-down menu on the menu bar
- Clicking a toolbar button
- Right-clicking from a particular context, such as from the **Project** window
- Clicking a configured user tool ()
- Clicking a button within a dialog box
- Running a Tcl script (from the **File** menu or **Output** window)
- Choosing a command from the application's control menu

Working With Files

When working with files, use the keyboard shortcuts listed in [Table A-2](#).

Table A-2. Keyboard Shortcuts for Working With Files

Action	Key(s)
Open a new file	Ctrl+N
Open an existing file	Ctrl+O
Save a file	Ctrl+S
Print a file	Ctrl+P
Go to the next window	F6
Go to the previous window	Shift+F6

Moving Within a File

To move within a file, use the keyboard shortcuts listed in [Table A-3](#).

Table A-3. Keyboard Shortcuts for Moving Within a File

Action	Key(s)
Move the cursor to the left one character	Left Arrow (←)
Move the cursor to the right one character	Right Arrow (→)
Move the cursor to the beginning of the file	Ctrl+Home
Move the cursor to the end of the file	Ctrl+End
Move the cursor to the beginning of the line	Home
Move the cursor to the end of the line	End
Move the cursor down one line	Down Arrow (↓)
Move the cursor up one line	Up Arrow (↑)
Move the cursor one page down	Page Down
Move the cursor one page up	Page Up
Move the cursor right one tab	Shift
Move the cursor left one tab	Shift+Tab
Move the cursor left one word	Ctrl+Left Arrow (←)
Move the cursor right one word	Ctrl+Right Arrow (→)
Move to the matching brace character within a file	Ctrl+B
Go to the next bookmark	F2
Go to a line	Ctrl+G
Find text	Ctrl+F
Find the next occurrence of text	F3

Cutting, Copying, Pasting, Moving Text

To edit text, use the keyboard shortcuts listed in [Table A-4](#).

Table A-4. Keyboard Shortcuts for Editing Text

Action	Key(s)
Copy text	Ctrl+C or Ctrl+Insert
Copy text	Select with cursor and Ctrl +drag
Cut text	Ctrl+X or Shift+Delete
Delete text	Delete (selection or forward)
Delete text	Backspace (selection or backward)
Move text	Select with cursor and drag
Move selected text right one tab	Tab
Move selected text left one tab	Shift+Tab
Paste text	Ctrl+V or Shift+Insert
Undo the last edit	Ctrl+Z or Alt+Backspace
Redo an edit command	Shift+Ctrl+Z
Replace text	Ctrl+H or Ctrl+R

Selecting Text Within a File

To select text within a file, use the keyboard shortcuts listed in [Table A-5](#).

Table A-5. Keyboard Shortcuts for Selecting Text Within a File

Action	Key(s)
Select all text in a file	Ctrl+A
Select the character on the left	Shift+Left Arrow (←)
Select the character on the right	Shift+Right Arrow (→)

Keyboard Shortcuts

Table A-5. Keyboard Shortcuts for Selecting Text Within a File (Cont'd)

Action	Key(s)
Select all text to the beginning of the file	Shift+Ctrl+Home
Select all text to the end of the file	Shift+Ctrl+End
Select all text to the beginning of the line	Shift+Home
Select all text to the end of the line	Shift+End
Select all text to the line below	Shift+Down Arrow (↓)
Select all text to the line above	Shift+Up Arrow (↑)
Select all text to the next page	Shift+PgDn
Select all text to the above page	Shift+PgUp
Select the word on the left	Shift+Ctrl+Left Arrow (←)
Select the word on the right	Shift+Ctrl+Right Arrow (→)
Select by column	Place cursor, press and hold down Alt and drag the cursor (selects by column-character instead of by line-character)

Working With Bookmarks in an Editor Window

When working with bookmarks in an editor window, use the keyboard shortcuts listed in [Table A-6](#).

Table A-6. Keyboard Shortcuts for Bookmarks

Action	Key(s)
Toggle a bookmark	Ctrl+F2
Go to next bookmark	F2

Building Projects

To build projects, use the keyboard shortcuts listed in [Table A-7](#).

Table A-7. Keyboard Shortcuts for Building Projects

Action	Key(s)
Build the current project	F7
Build only the current source file	Ctrl+F7

Using Keyboard Shortcuts for Program Execution

For program execution, use the keyboard shortcuts listed in [Table A-8](#).

Table A-8. Keyboard Shortcuts for Program Execution

Action	Key(s)
Load a Program	Ctrl+L
Reload a Program	Ctrl+R
Dump to File	Ctrl+D
Run	F5
Multiprocessor Run	Ctrl+F5
Run to Cursor	Ctrl+F10
Halt	Shift+F5
Step Over	F10
Step Into	F11
Multiprocessor Step	Ctrl+F11
Step Out Of	Alt+F11
Halt a Script	Ctrl+H

Keyboard Shortcuts

Working With Breakpoints

When working with breakpoints, use the keyboard shortcuts listed in [Table A-9](#).

Table A-9. Keyboard Shortcuts for Breakpoints

Action	Key(s)
Open the Breakpoints dialog box	Alt+F9
Enable/disable a breakpoint	Ctrl+F9
Toggle (add or remove) a breakpoint	F9

Obtaining Online Help

To obtain online Help, use the keyboard shortcuts listed in [Table A-10](#).

Table A-10. Keyboard Shortcuts for Obtaining Online Help

Action	Key(s)
View online Help for the selected object	F1
Obtain context-sensitive Help for controls (buttons, fields, menu items)	Shift+F1

Miscellaneous

For windows and workspaces, use the keyboard shortcuts listed in [Table A-11](#).

Table A-11. Miscellaneous Keyboard Shortcuts

Action	Key(s)
Refresh all windows	F12
Select workspace 1 through 10	Alt+1 ... Alt+0

IDDE Command-Line Parameters

VisualDSP++ can be invoked from a DOS command line.

Syntax:

```
idde.exe [-f script_name]
         [-s session_name]
         [-p project_name]
```



Note: Specify the full path to `idde.exe`.

Table A-12 describes the `idde.exe` command-line parameters.

Table A-12. `idde.exe` Command-Line Parameters

Item	Description
<code>-f script_name</code>	Loads and executes the Tcl script specified by <code>script_name</code> . Use this parameter to automate regression tests. You can also manipulate VisualDSP++ by running a Tcl script from a library of common Tcl commands that you create. If an error is encountered while executing this script, VisualDSP++ automatically exits.
<code>-s session_name</code>	Specifies the session to which VisualDSP++ connects when it starts. The session must already exist. This parameter is useful when you are debugging more than one target board. Having multiple shortcuts to <code>idde.exe</code> allows you to run a different session. This overrides VisualDSP++'s default behavior of always connecting to the last session.
<code>-p project_name</code>	Specifies the project to load at startup. The project must already exist.

Examples:

```
idde.exe -f "c:\\scripts\\myscript.tcl"
```

```
idde.exe -s "My BF535 JTAG Emulator Session"
```

```
idde.exe -p "c:\\projects\\myproject.dpj"
```

Extensive Scripting

You can issue script commands from a command line, from the **Output** window's **Console** view, from a menu, from an editor window, or from a user tool.

- From a command line

Load a script from a command window with an `idde` command by typing:

```
idde -f <filename>
```

Optionally, add `-s` and the session name to specify a previously created session. When no session name is specified, the last session is used.

If the script encounters an error during execution, VisualDSP++ automatically exits.

- From the **Output** window

Load a script from the **Output** window's **Console** view by typing one of the following commands.

For the Microsoft ActiveX script engine, type:

```
Idde.LoadScript (filename)
```

For Tcl, type:

```
source filename
```

As in C/C++, use a backslash (`\`) as an escape character. If you specify paths in the Windows environment, you must escape the escape character, as shown in this example:

```
c:\\my_dir\\my_subdir\\my_file.tcl
```

For Tcl only, you can also use forward slashes to delimit directories in a path, as shown in this example:

```
source c:/my_dir/my_subdir/my_file.tcl
```

Command execution is deferred until a line is typed without a trailing backslash. This feature permits the entry of an entire block of code (or entire procedure) for the script interpreter to evaluate at once.

Use the built-in `Idde` object to easily access the properties and methods of the VisualDSP++ Automation API when using a Microsoft ActiveX script engine. For example:

```
Idde.ActiveSession.ActiveProcessor
```

Evaluate expressions by using the “?” when a Microsoft ActiveX script engine is selected. For example:

```
? Idde.FullName
```

- From a menu

You can quickly issue frequently used scripts. From the **File** menu, choose **Recent Scripts** and then select the script.

Extensive Scripting

- From an editor window

In an open editor window that contains a script, right-click and choose **Load Script**, as shown in [Figure A-1](#).

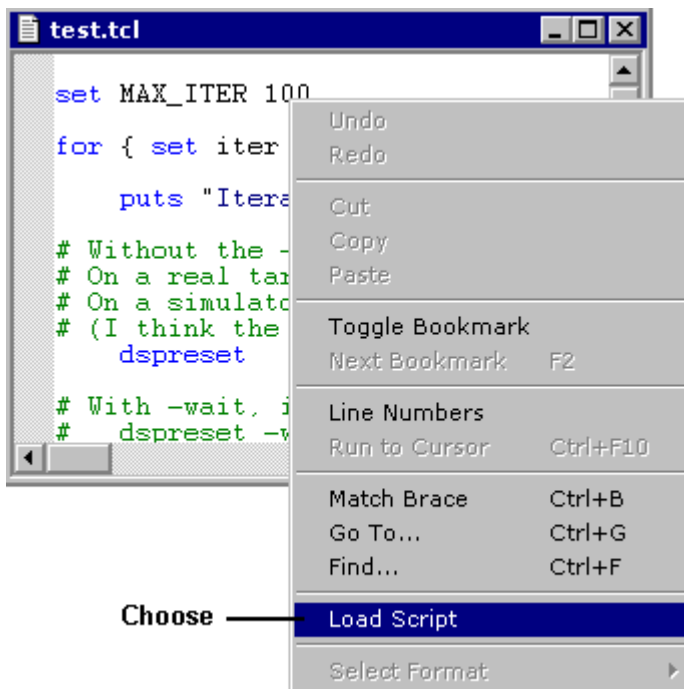


Figure A-1. Running a Script From an Editor Window

- From a user tool

From a toolbar, click a user tool or choose a user tool from the **Tools** menu.

You can invoke a script (such as `.js` or `.vbs`) automatically when launching VisualDSP++ from a shortcut set up on your Window's desktop or **Start** button. Right-click on the shortcut and select **Properties** and the **Shortcut** tab. Then append `-f` and the name of the script file to the executable file in the **Target** text box.

The example shown in [Figure A-2](#) runs `myscript.js` automatically when `idde.exe` is launched.

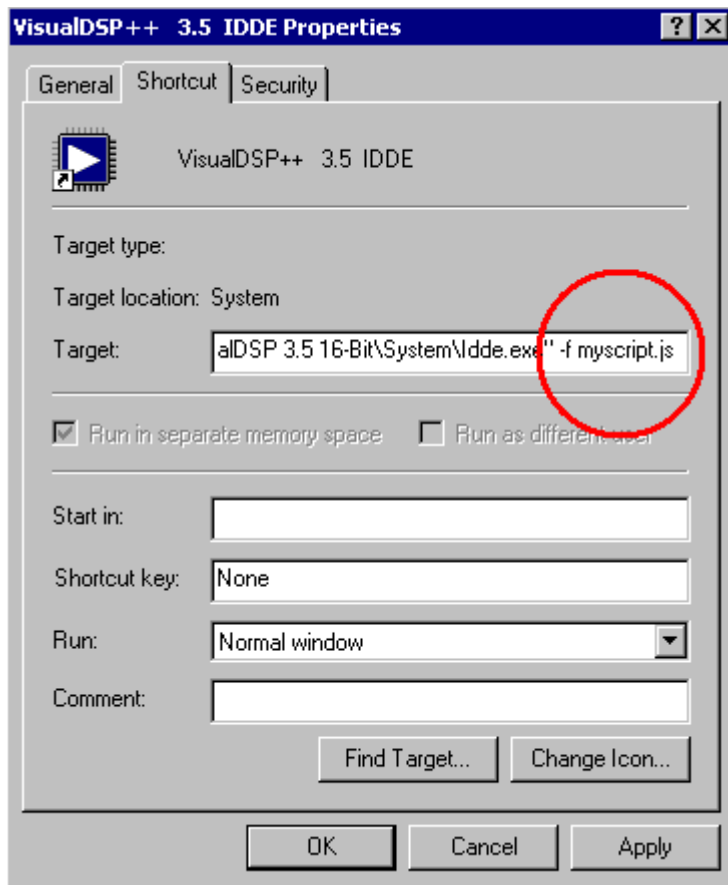


Figure A-2. Loading a Script From a Shortcut

Toolbar Buttons

The toolbar, which comprises separate tool buttons, provides quick mouse access to commands.

The toolbar is a Windows docking bar. You can move it to different areas of the screen by dragging it to the selected location.

Table A-13. Toolbar Buttons

Button	Purpose
	Creates a new document
	Opens an existing document
	Saves the active document or template with the same name
	Saves all open files that have been modified, including files not in the current project
	Prints the active document
	Loads a program into the target
	Reloads the most recent program into the target
	Cuts selected data from the document and store it on the clipboard
	Copies the selection to the clipboard
	Pastes the contents of the clipboard at the insertion point

Table A-13. Toolbar Buttons (Cont'd)

Button	Purpose
	Undoes previous edit command (multilevel undo)
	Redoes the command undone by the previous Undo command (multilevel redo)
	Finds a text block in an editor window
	Finds again or repeats the previous find command
	Replaces the selected text with other text
	Searches through files for text or regular expressions
	Goes to or moves to the specified location
	Displays the current source file
	Toggles the bookmark at selected line in the active editor window
	Goes to the next bookmarked line in the editor window
	Goes to the previous bookmarked line in the editor window
	Clears all bookmarks in the editor window
	Opens the online Help to the Search page

Toolbar Buttons

Table A-13. Toolbar Buttons (Cont'd)

Button	Purpose
	Provides context-sensitive Help for a button command or portion of VisualDSP++
	Opens the About VisualDSP++ dialog box
	Adds a source file to the project
	Removes a selection from the project
	Opens an existing project
	Saves the open project
	Opens the Project Options dialog box, where you specify project options
	Builds the selected source file
	Builds the project (update outdated files)
	Builds all files in the project
	Stops the current project build
	Arranges windows as tall non-overlapping tiles
	Arranges windows as wide non-overlapping tiles

Table A-13. Toolbar Buttons (Cont'd)

Button	Purpose
	Arranges windows so they overlap
	Closes all open windows
	Refreshes all the debugging windows
	Runs (starts or continues) the current program
	Restarts the current program
	Stops the current program
	Resets the target
	Toggles a breakpoint for the current line
	Clears all current breakpoints
	Enables or disables one breakpoint
	Disables all breakpoints
	Steps one line
	Steps over the current statement

Toolbar Buttons

Table A-13. Toolbar Buttons (Cont'd)

Button	Purpose
	Steps out of the current function
	Runs the program to the line containing the cursor
	Opens the Expressions window
	Opens the Locals window
	Opens the Call Stack window
	Opens the Disassembly window
	Runs the command associated with the user tool (one of ten)
	Opens the associated workspace (one of ten)

Text Operations

VisualDSP++ allows the use of regular expressions and tagged expressions in find/replace operations and comments in your code.

Regular Expressions Versus Normal Searches

Normally, when you search for text, the search mechanism scans for an exact, character-by-character match of the search string, which does not have to be an entire word. Every character in the search string is examined. If there are embedded spaces, for instance, the exact number is matched.

Regular expression matching provides much more flexibility and power than a normal search. A regular expression can be a simple string, which yields the same matches as normal searches. Some characters in a regular expression string, however, have special interpretations, which provide greater flexibility.

For example, with regular expression matching, you can find the following.

- All occurrences of either `hot` or `cold`
- Occurrences of `for` followed by a left parenthesis, with any number of intervening spaces
- A semicolon (`;`) only when it is the last character on a line
- The string `ADSP` followed by a sequence of digits

Using a regular expression as the search pattern for replacement provides ways to identify and recover the variable portions of the matched strings.

Text Operations

Specific Special Characters

Regular expressions assign special meaning to the following characters.

If you have to match on one of these characters, you must escape it by preceding it with a backslash (\). Thus, \[^] matches the [^] character, yet [^] matches the beginning of the line.

Table A-14. Special Search Characters

Character	Description
[^]	A caret matches the beginning of the line.
\$	A dollar sign matches the end of the line.
.	A period (.) matches any character.
[abc]	A bracketed sequence of characters matches one character, which may be any of the characters inside the brackets. Thus, [abc] matches an a, b, or c.
[0-9]	This shorthand form is valid within the sequence brackets. It specifies a range of characters, from first through last, exactly as if they had been written explicitly. Ranges may be combined with explicit single characters and other ranges within the sequence. Thus, [-+.0-9] matches any constituent character of a signed decimal number; and [a-zA-Z0-9_] matches a valid identifier character, either lowercase or uppercase. Ranges follow the ordering of the ASCII character set.
[^abc] [^0-9]	A caret (^) that is the first character of a sequence matches all characters except for the characters specified after the caret.
(<i>material</i>)	The material inside the parentheses can be any regular expression. It is treated as a unit, which can be used in combination with other expressions. Parenthesized material is also assigned a numerical tag, which may be referenced by a replace operation.

Special Rules for Sequences

The normal special character rules of regular expressions do not apply within a bracketed sequence. Thus, `[*&]` matches an asterisk or ampersand.

Certain characters have special meaning within a sequence. These include `^` (not), `-` (range), and `]` (end of sequence). By placing these characters appropriately, you can specify these characters to be part of the sequence.

To search for a right bracket character, place `]` as the first character of the search string. To search for a hyphen character, place `-` as the first character of the search string after `]`, if present. Place a caret anywhere in the search string except at the front, where it means “not.”

Repetition and Combination Characters

Each character described in [Table A-15](#) extends the meaning of the item that immediately precedes it. This item may be a single character, a sequence in braces, or an entire regular expression in parentheses.

Table A-15. Match Characters

Character	Description
*	An asterisk matches the preceding any number of times, including none at all. Thus, <code>ap*le</code> matches <code>apple</code> , <code>aple</code> , <code>appppple</code> and <code>ale</code> . For example, <code>^ *void</code> matches only when <code>void</code> occurs at the beginning of a line and is preceded by zero or more spaces.
+	A plus character matches the preceding any number of times, but at least one time. Thus, <code>ap+le</code> matches <code>apple</code> and <code>aple</code> , but does not match <code>ale</code> .
?	A question mark matches the preceding either zero or one time, but not more. Thus, <code>ap?le</code> matches <code>ale</code> and <code>aple</code> , but nothing else.
	The pipe character (<code> </code>) matches either the preceding or following item. For example, <code>(hot) (cold)</code> matches either <code>hot</code> or <code>cold</code> . Spaces are characters. Thus, <code>(hot) (cold)</code> matches “hot “or” cold”.

Text Operations

Match Rules

If multiple matches are possible, the *, +, and ? characters match the longest candidates. The | character matches the left-hand alternative first.

For more information, see the many reference texts available on this topic, such as *Mastering Regular Expressions*, *Powerful Techniques for Perl and Other Tools* by Jeffrey E. F. Friedl, (c) 1997 O'Reilly & Associates, Inc.

Tagged Expressions in Replace Operations

Use a tagged expression as part of the string in the **Replace** field for a replace operation.

A tagged expression must be enclosed between parentheses characters. In the **Replace** field, the operators in [Table A-16](#) represent tagged expressions from the **Find** field.

Table A-16. Using Tagged Expressions in Replace Operations

Find Field	Replace Field
Entire matched substring	\0
Tagged expressions within parentheses () from left to right	\1 \2 \3 \4 \5 \6 \7 \8 \9
Entire match expression	&

The replace expression can specify an ampersand (&) character, meaning that the & represents the substring that was found. For example, if the substring that matched the regular expression is “abcd”, a replace expression of “xyz&xyz” changes it to “xyzabcdxyz”. The replace expression can also be expressed as “xyz\0xyz”, where the “\0” indicates a tagged expression representing the entire matched substring. Similarly, you can have another tagged expression represented by “\1”, “\2”.



Although the tagged expression 0 is always defined, the tagged expressions 1, 2, and so on, are defined only when the regular expression used in the search has enough sets of parenthesis. Some examples are shown in [Table A-17](#).

Table A-17. Examples of Replace Operations

String	Search	Replace	Result
Mr.	(Mr)(\.)	\1s\2	Mrs.
abc	(a)b(c)	&\1-\2	abc-a-c
bcd	(a b)c*d	&\1	bcd-b
abcde	(.*)c(.*)	&\1-\2	abcde-ab-de
cde	(ab cd)e	&\1	cde-cd

Comment Start and Stop Strings

Use start comment strings and stop comment strings for comment highlighting colors. [Table A-18](#) describes the two types of comment strings that you can set for each file type.

Table A-18. Start and Stop Comment Strings

String	Purpose
!	Starts an assembly style, singleline comment
/*	Starts a C/C++ style, multiline comment
//	Starts a C/C++ style, singleline comment
Carriage return	Ends a singleline comment (C and Assembly)
*/	Ends a C/C++ style, multiline comment
(blank)	Ends a C/C++ style, singleline comment

Online Help Features and Operations

This section describes online Help features and explains how to:

- Use the Help window
- Invoke online Help
- View context-sensitive Help
- Use the Help window's navigation buttons
- Copy example code from Help
- Print Help
- Bookmark frequently used help topics
- Navigate in online Help
- Use the search features
- View and print online manuals
- Use the **About VisualDSP++** dialog box

Using the Help Window

The Help window comprises three parts:

- The *Navigation pane* provides tabbed pages (**Contents**, **Index**, **Search**, and **Favorites**) that show different navigational views.
- The *Viewing pane* displays the selected object (topic, Web page, video, .PDF file, application).
- *Toolbar buttons* let you navigate and specify options.

Figure A-3 shows the parts of the VisualDSP++ Help window.

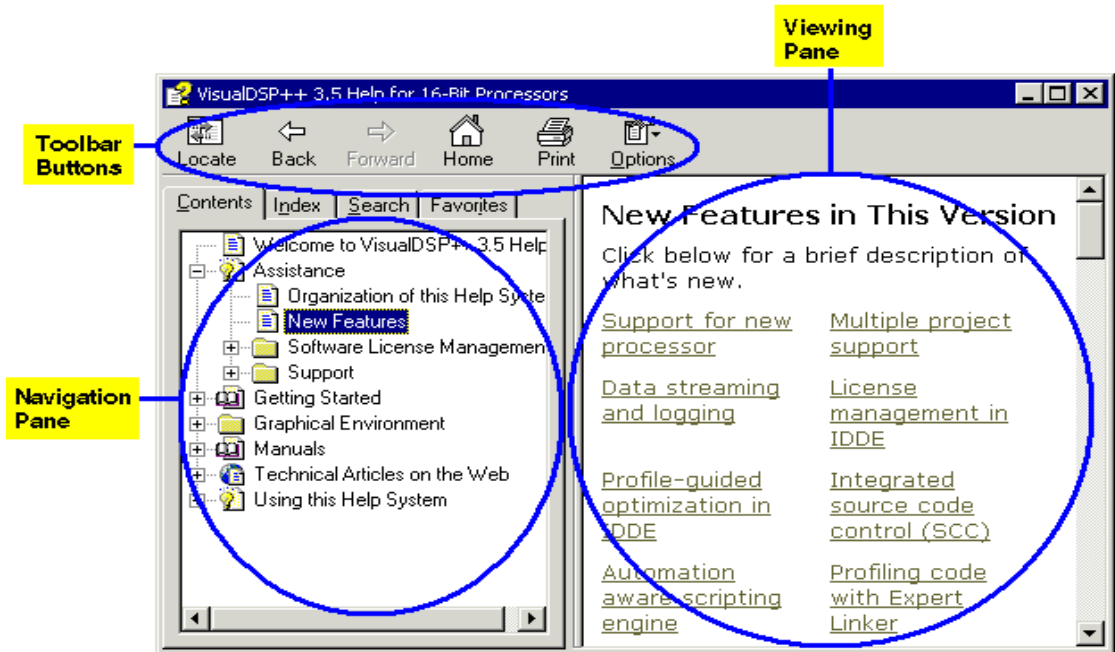


Figure A-3. Parts of the VisualDSP++ Help Window

Invoking Online Help

You can invoke online Help from VisualDSP++ or from the Windows **Start** button. You can also access Help manually via Windows Explorer.

To access online Help from the VisualDSP++ **Help** menu, choose **C**ontents, **S**earch, or **I**ndex.

To access online Help from the Windows **Start** button, click the **Start** button and choose **P**rograms, **A**nalog Devices, **V**isualDSP 3.5 for 16-Bit Processors, and **V**isualDSP++ **D**ocumentation.

Online Help Features and Operations

The Help function is programmed to look for the Help system in the VisualDSP++ Help folder.

By default, the VisualDSP++ software installation procedure places the complete set of Help files (except the *Getting Started Guide*) in the installation's Help folder.

If you receive an error message after invoking Help, the Help system:

- May not have been loaded onto your PC
- May have been deleted
- May reside in a directory other than the default directory

To locate the help (.CHM) files manually, use the Windows **Search** function as follows.

1. Record the Help file (.CHM) named in the error message.
2. From the Windows **Start** button, choose **Search** and **For Files or Folders**. Enter the name of the .CHM file from step 1.
3. After locating the file, launch it manually by clicking the file name from the **Search Results** window or from Windows Explorer.

Viewing Context-Sensitive Help

You can view context-sensitive Help (help pertinent to your current activity) for various items in VisualDSP++.

VisualDSP++'s context-sensitive Help is linked to toolbar buttons, menu commands, windows, and dialog box items.

Viewing Menu, Toolbar, or Window Help

Complete these steps to view menu, toolbar, or window Help:

1. Click the toolbar's Help button () or press **Shift+F1**.
The mouse pointer becomes a Help pointer ().
2. Move the Help pointer over a menu command, toolbar button, or window.
3. Click the mouse to open the Help window. A description of the object appears in the right panel.

Viewing Dialog Box Button or Field Help

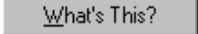
Perform one of these actions to view button or field Help in a dialog box:

- Select a field or button in a dialog box and press **F1** or **Shift+F1**.
- Click the question mark button () in the top-right corner of the dialog box.

The mouse pointer becomes a Help pointer ().

Move the Help pointer over a dialog box control (button or field) and click the mouse. A description of the object appears in a yellow pop-up window.

- Position the mouse pointer over a label or control (button or field) in a dialog box and right-click.

A **What's This** button () appears. Move the mouse pointer over the **What's This** button and click.



“What's This” Help is not configured for all items.

Viewing Window Help

Complete these steps to view window Help:

1. Click the window to make it active.
2. Press the F1 key to open the Help window.

A description of the window appears in the right panel.

Using Help Window Navigation Buttons

Move through the Help system and view Help topics by using the Help window's navigational aids, as shown in [Figure A-4](#).

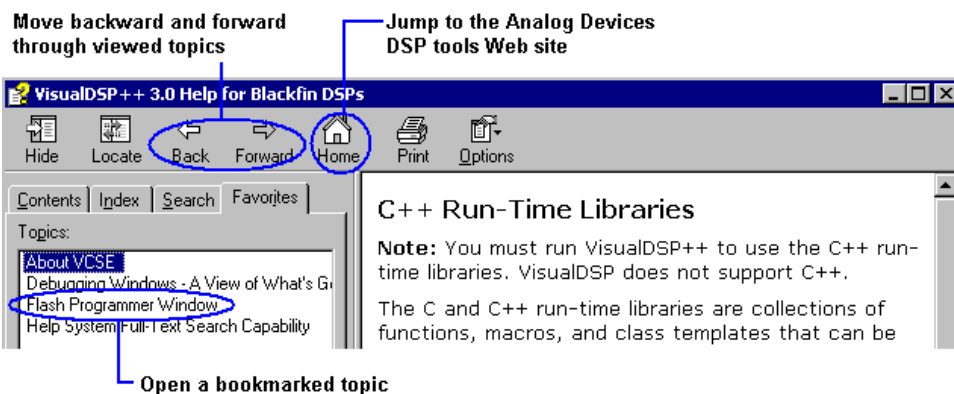


Figure A-4. Help Window Navigational Aids

Other standard Microsoft HTML Help buttons are described in [Table A-19](#).

Table A-19. Standard Microsoft HTML Help Buttons

Button	Purpose
 Hide	Hides the Help window's left pane. This button narrows the Help window.
 Show	Displays the Help window's left pane. This button restores a full view after you click Hide .
 Locate	Highlights the name of the current topic on the Contents page (left pane). After you jump around the Help system, this button shows the current topic's relation to other topics.

Copying Example Code From Help

You can copy code from the Help system and then paste it into your application. Be aware that the copied text may carry unwanted control codes. For example, if you copy a hyphen with a parameter, the actual code of the copied hyphen may be an ASCII 0x96 instead of an ASCII 0x2D. The hyphen may look OK, but it will cause an error when the command is run.

Printing Help

You can print a specific Help topic or multiple Help topics (an entire section of online Help).

Table A-20. How to Print Help Topics

To print	Do this
Current topic	Right-click within the help topic and choose Print .
Selected topic	On the Contents page: Right-click the topic () and choose Print .
Entire section of Help	On the Contents page: Right-click a book icon ( or ) and choose Print . Then choose Print the selected heading and all subtopics.

Tip: From the Help window's **Contents** page, click (), located at the top of the window.

Bookmarking Frequently Used Help Topics

Bookmarking a topic in online Help is just like bookmarking a page in a book. This feature is also called “setting up favorite places.”

Note: Each time a Microsoft HTML Help topic is bookmarked, a record is recorded in the file, `HH.DAT`. This file not only records VisualDSP++ Help bookmarks, but also the bookmarks placed in other application Help systems that use `.CHM` files.

Once a bookmark is placed onto a topic, you can view a list of bookmarked topics and quickly open one.

Placing a Bookmark at a Topic

To place a bookmark at a topic:

1. Display the topic.
2. On the left side of the Help window, click the **Favorites** tab.
3. Click **Add**.

Remove a bookmark by selecting the name and clicking **Remove**.

The Help system adds the topic and displays it in the alphabetized list.

Opening a Bookmarked Topic

To open a bookmarked topic:

1. On the left side of the Help window, click the **Favorites** tab.
2. Perform one of these actions:
 - Double-click the topic.
 - Select the topic and click **Display**.

Navigating in Online Help

To move around in the Help system, click the following.

- **A hyperlink within text.** The text is underlined and displayed in a color that is different from the regular black text.
- **A topic listed under a See Also heading.** The text is underlined and displayed in a color that is different from the regular black text.

Online Help Features and Operations

- A **mini button** or its associated **text**. The button is a small gray square and the underlined text is in a different color.
- A **topic name on the Contents page** (Figure A-5)

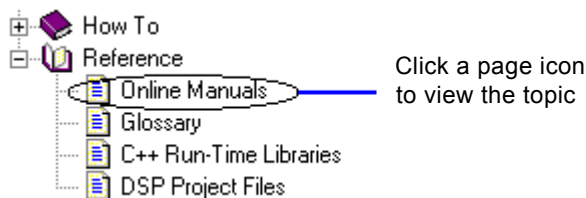


Figure A-5. Contents Page – Online Manual Topics

- An **index entry on the Index page** (Figure A-6)

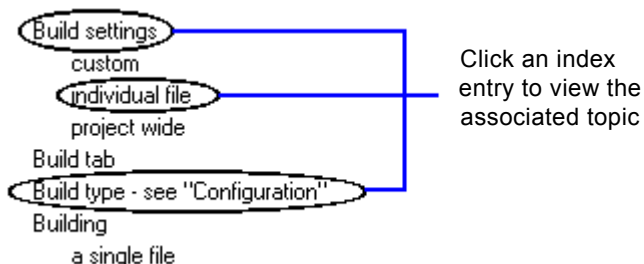


Figure A-6. Index Entries on the Index Page

- A **topic name on the Search page**. The bottom portion of the Search page displays the located topics (hits) that include your search string.

Using the Search Features

VisualDSP++ Help provides both full text and advanced search capabilities for finding information.

Help System Search Rules

Different rules apply for each type of search.

Rules for Full Text Searches

Observe these rules when formulating queries:

- Searches are not case sensitive. Type your search in uppercase or lowercase characters.

You can search for any combination of letters (a–z) and numbers (0–9).

- Searches ignore punctuation marks such as the period, colon, semi-colon, comma, and hyphen.
- Group the elements of your search by using double quotes or parentheses to set apart each element.
- You cannot search for quotation marks.

Note that if you are searching for a file name with an extension, group the entire string in double quotes, (“filename.ext”). Otherwise, the period breaks the file name into two separate terms. The default operation between terms is AND, which creates the logical equivalent to filename AND ext.

Online Help Features and Operations

Rules for Advanced Searches

These rules apply to advanced searches:

- Expressions in parentheses are evaluated before the rest of the query.
- If a query does not contain a nested expression, it is evaluated from left to right. For example, “folder NOT file OR project” finds topics containing the word “folder” without the word “file,” or topics containing the word “project.” The expression “folder NOT (file OR project)”, however, finds topics containing the word “folder” without either of the words “file” or “project.”
- You cannot nest expressions deeper than five levels.

Full Text Searches

A full text search locates every occurrence of a text string within the Help system. Specify a particular word or phrase to find only the topics that contain that word or phrase.

You can search previous results, match similar words, and search through the topic titles only.

A basic search consists of the word or phrase that you want to locate. Use similar word matches, a previous results list, or topic titles to further define your search.

You can run an advanced search, which uses Boolean operators and wild-card expressions to further narrow the search criteria. [Figure A-7 on page A-59](#) shows an example of a Boolean search for “new AND plot”.

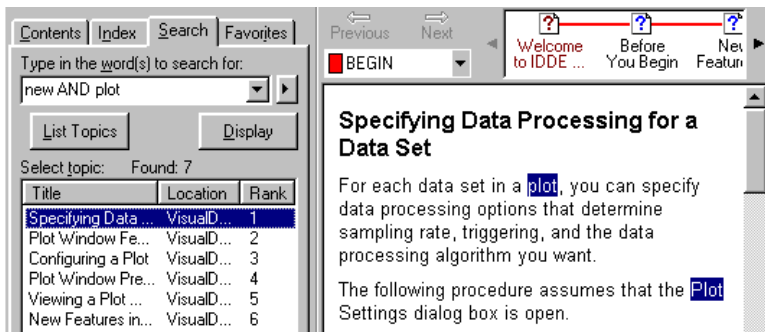


Figure A-7. Boolean Search for “new AND plot”

To find information with a full text search:

1. Click the Help viewer’s **Search** tab.
2. In **Type in the word(s) to search for**, type the word or phrase you want to find.
3. Select **Search previous results** to narrow the search.
4. Select **Match similar words** to find words that are similar to the search string.
5. Select **Search titles only** to search only the topic titles.
6. Click the **Options** button () at the top of the Help Viewer window to highlight all instances of search terms found in topic files. Then choose **Search Highlight On**.
7. Click **List Topics**, select the topic you want, and then click **Display**.

Note that you can sort the topic list by clicking the **Title**, **Location**, or **Rank** column heading.

Advanced Search Techniques

Use the following search techniques to narrow your searches for more precise results.

- Wildcard expressions
- Boolean operators
- Nested expressions

Using Wildcard Expressions

Wildcard expressions let you search for one or more characters by using a question mark or asterisk. [Table A-21](#) describes the results of these different kinds of searches.

Table A-21. How to Use Wildcard Expressions to Define a Search

Search Target	Example	Results
A single word	project	Locates topics that contain the word “project.” Other grammatical variations, such as “projects” are located.
A phrase	“project window” (note the quotation characters) project window	Locates topics that contain the literal phrase “project window” and all its grammatical variations. Without the quotation characters, the query is equivalent to specifying “project AND window,” which finds topics containing both of the individual words, instead of the phrase.
Wildcard expressions	link* -or- .C??	Locates topics that contain the terms “linker,” “linking,” “links,” and so on. The asterisk cannot be the only character in the term. Locates topics that contain the terms “.CPP” or “.CXX.” The question mark cannot be the only character in the term.

Using Boolean Operators

Use the Boolean **AND**, **OR**, **NOT**, and **NEAR** operators to precisely define your search by creating a relationship between search terms.

Insert a Boolean operator by typing the operator (**AND**, **OR**, **NEAR**, or **NOT**) or by clicking the arrow button.

Note that if you do not specify an operator, **AND** is used. For example, the query `call stack` is equivalent to `call AND stack`.

[Table A-22](#) describes the results of using Boolean operators to define a search.

Table A-22. How to Use Boolean Operators to Define a Search

Search Target	Example	Results
Both terms in the same topic	<code>new AND plot</code>	Locates topics that contain both the words “new” and “plot”
Either term in a topic	<code>new OR plot</code>	Locates topics that contain either the word “new” or the word “plot” or both
The first term without the second term	<code>new NOT plot</code>	Locates topics that contain the word “new”, but not the word “plot”
Both terms in the same topic, close together	<code>new NEAR plot</code>	Locates topics that contain the word “new” within eight words of the word “plot”

Do not use the `|`, `&`, or `!` characters as Boolean operators. You must use **OR**, **AND**, or **NOT**.

Online Help Features and Operations

Using Nested Expressions

Use nested expressions to create complex searches for information. For example, `new AND ((plot OR waterfall) NEAR window)` finds topics containing the word “new” along with the words “plot” and “window” close together, or topics containing “new” along with the words “waterfall” and “window” close together.

Viewing Online Manuals

VisualDSP++ includes three types of user documentation.

Table A-23. Types of User Documentation

Files	Purpose
.CHM	VisualDSP++ online Help system files and VisualDSP++ manuals are provided in Microsoft HTML Help format. Installing VisualDSP++ automatically copies these files to the <code>VisualDSP\Help</code> folder. Online Help is ideal for searching the entire tools manual set. Invoke Help from the VisualDSP++ Help menu or via the Windows Start button. The .CHM files require Internet Explorer 4.0 (or higher) or the installation of a component that provides a .CHM file viewer.
.PDF	Manuals and data sheets in Portable Documentation Format are located in the installation CD's <code>Docs</code> folder. Viewing and printing a .PDF file requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher). Running <code>setup.exe</code> on the installation CD provides easy access to these documents. You can also copy PDF files from the installation CD onto another disk.
.HTM or .HTML	Dinkum Abridged C++ library and FlexLM network license manager software documentation is located on the installation CD in the <code>Docs\Reference</code> folder. Viewing or printing these files requires a browser, such as Internet Explorer 4.0 (or higher). You can copy these files from the installation CD onto another disk.



The VisualDSP++ software installation procedure does not copy PDF versions of books and data sheets or supplemental reference documentation to the `VisualDSP` directory.

Printing Online Documents

You can print documents from the VisualDSP++ Tools Installation CD-ROM.

To print online documents:

1. Insert the VisualDSP++ Tools Installation CD-ROM in the CD-ROM drive.
2. Open the **Docs** folder by using one of these options:
 - From the **VisualDSP++ Tools Installation** main menu, click **View Documentation**. (If the main menu does not appear, run `setup.exe`.)
 - In Windows Explorer, select the CD-ROM drive (for example, **d:**) and open the **Docs** folder.
3. Open the folder where the document is located.

The `Data Sheets` folder contains copies of processor data sheets.

The `Hardware Manuals` folder contains copies of hardware manuals.

The `Reference` folder includes the `.HTML` files that comprise the Dinkum Abridged C++ library and the FlexLM network license documentation.

The `Tools Manuals` folder contains copies of VisualDSP++ tools manuals.

4. Double-click the document that you want to print. Selecting a `.PDF` file opens Adobe Acrobat Reader and displays the document. Selecting an `.HTML` file opens a browser and displays the document.
5. From the **File** menu, choose **Print** and specify the pages that you want to print (and other print options).

Using the About VisualDSP++ Dialog Box

Selecting the **About VisualDSP++** option from the **Help** menu opens the **About VisualDSP++** dialog box, which provides access to the following types of support information.

- Software versions

The **General** page ([Figure A-8](#)) provides information about your version of VisualDSP++. This information includes the name of the registered user and company, the version of IDDE and its build date, and the directory path in which VisualDSP++ is installed.

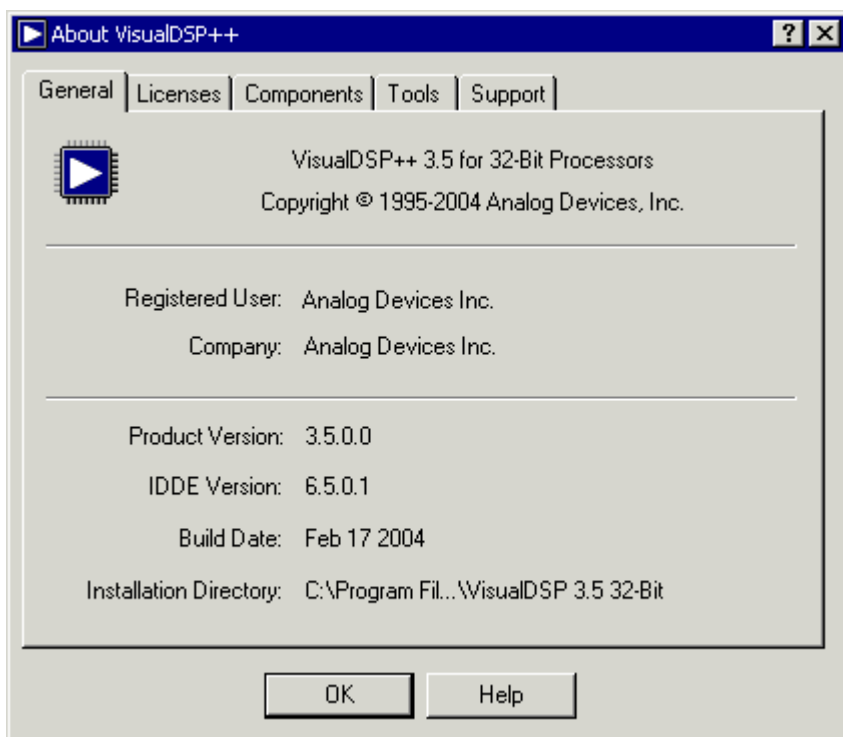


Figure A-8. General Page

- License management

The **Licenses** page (Figure A-9) provides a centralized view of your current licenses. You can view license status and perform all necessary licensing activities (installing, registering, and validating).

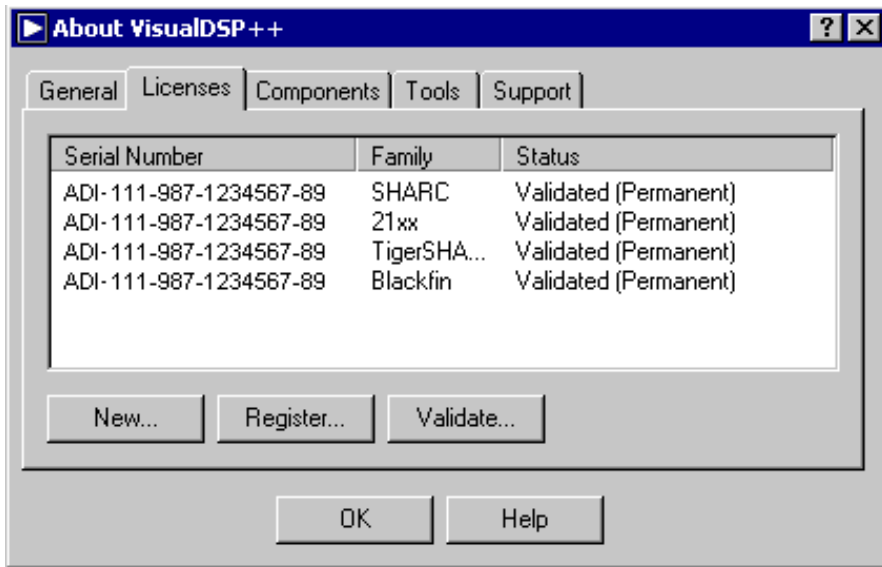


Figure A-9. Licenses Page

Online Help Features and Operations

- Component versions

The **Components** page (Figure A-10) displays a list of your system's components and provides information (name, version, provider) about your debug target, symbol manager, and processor library.

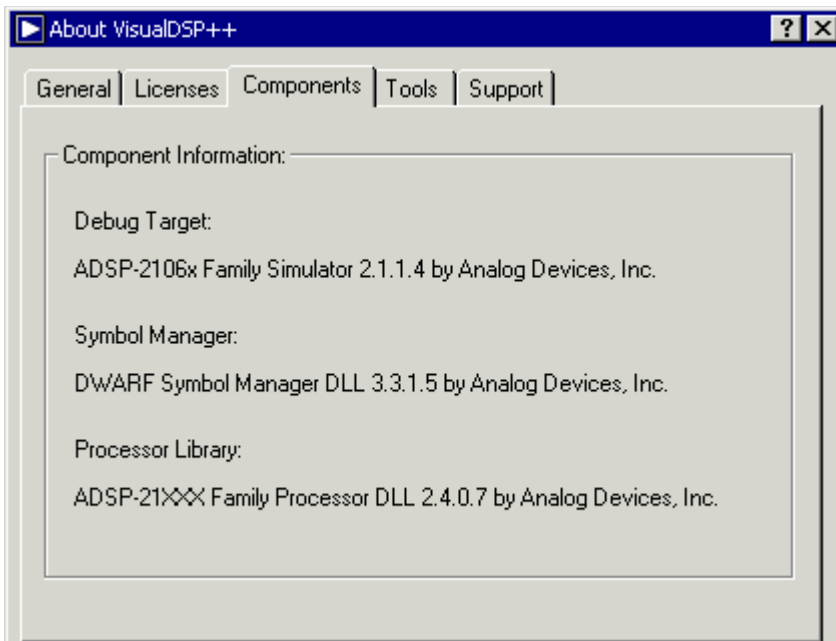


Figure A-10. Components Page

- Tools

The **Tools** page (Figure A-11) displays a list of your system's tools. Each tool includes a description, version number, and the name of the company that developed it.

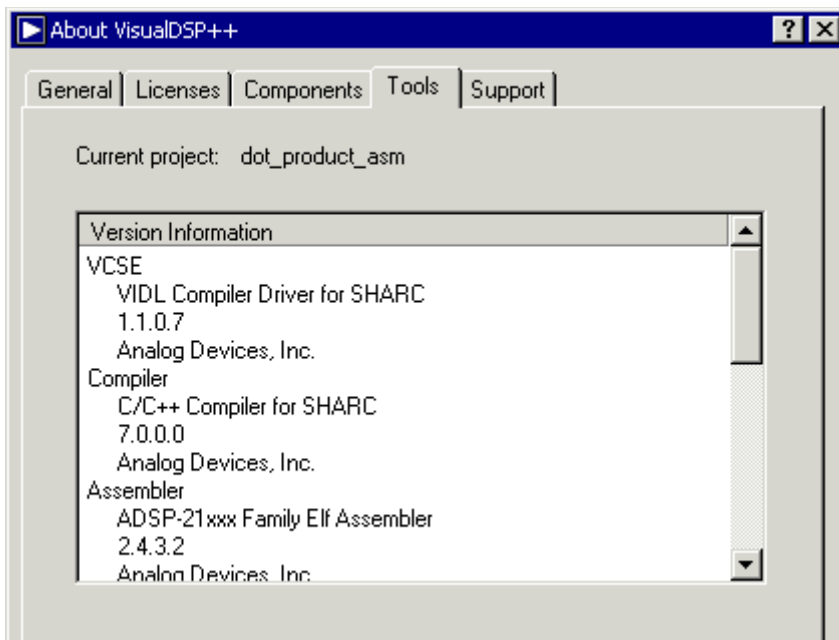


Figure A-11. Tools Page

Online Help Features and Operations

- Links to support on the Web

The **Support** page (Figure A-12) provides direct links to various Web pages that contain support information such as application notes, code examples, the DSP Knowledgebase, IC and tools anomalies and workarounds, manuals and datasheets, product comparisons, tools updates, and more. You can also generate the body of an email that automatically contains your system's description.

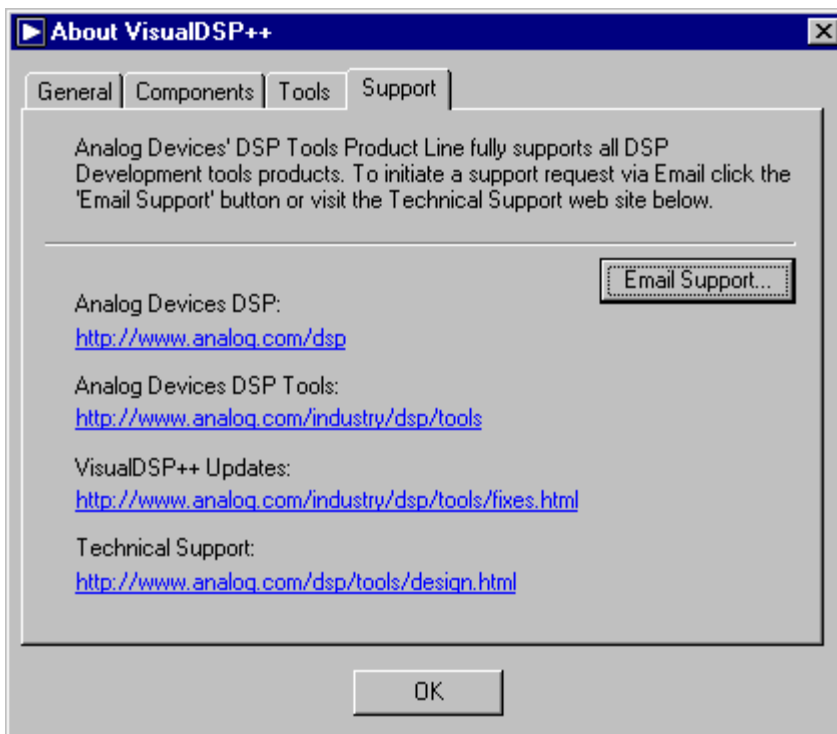


Figure A-12. Support Page

B SIMULATION OF TIGERSHARC PROCESSORS

This appendix provides information to help you simulate TigerSHARC processors.

The information is organized as follows.

- [“ADSP-TS101 Processors” on page B-1](#)
- [“ADSP-TS20x Processors” on page B-13](#)

ADSP-TS101 Processors

This section includes the following topics, which apply to ADSP-TS101 processors.

- Simulator Timing Analysis Overview
- Pipeline Stages
- Stalls
- Aborts
- Pipeline Viewer and Disassembly Window Operations
- Simulator Options

Simulator Timing Analysis Overview

The ADSP-TS101 simulator is a cycle-accurate simulator. It not only models the instruction set functionally, but also correctly models pipeline effects (stalls and aborts).

 Currently, the processor's external port and link ports are not modeled in a cycle-accurate manner. The simulator cycle-counts the code, but the cycle counts used for the external port or link ports are rough estimates of cycle counts that you could obtain by running the code on the chip. You should not rely on these counts for performance evaluation.

The **Pipeline Viewer** window helps you understand how processor timing affects the execution of your program. It shows the flow of instructions through the pipeline and any stalls due to sequencer or memory events. For information about configuring and using the Pipeline Viewer, see [“Pipeline Viewer Window” on page 2-102](#) or the VisualDSP++ on-line Help.

Pipeline Stages

The ADSP-TS101 **Pipeline Viewer** window provides a representation of instruction flow through the processor's pipeline. [Table B-1](#) lists the pipeline stages.

Table B-1. Pipeline Stages – ADSP-TS101

Stage	Abbreviation in Pipeline Viewer
Instruction Fetch 1	F1
Instruction Fetch 2	F2
Instruction Fetch 3	F3
Decode	DECODE
Integer	INT

Table B-1. Pipeline Stages – ADSP-TS101 (Cont'd)

Stage	Abbreviation in Pipeline Viewer
Access	ACCESS
Execute Stage 1	EX1
Execute Stage 2	EX2

Stages F1 through F3 represent the fetch pipe. Stages DECODE through EX2 represent the execution pipe. Fetch pipe stages contain raw quads of 32-bit words fetched from memory, not valid instruction lines.

When you view the **Pipeline Viewer** window in disassembly format (by using the right-click menu), instructions that appear in the fetch pipe are not valid instruction lines, but merely bits and pieces. Valid instruction lines appear in the execution pipe stages.

Stalls

The examples that follow illustrate the way Pipeline Viewer displays different types of stall events for ADSP-TS101 processors. For a complete list of pipeline effects and memory transaction timing, refer to the processor's Hardware Specification.

Stalls Due to IALU Dependency

The following sequence of instructions causes a 4-cycle stall at the Decode stage.

```
J10=0x0;;
```

```

xr5 = [j10 + 1];;      // this instruction is stalled
                        // in Decode until previous
                        // instruction reaches E2:
    
```

Pipeline Viewer									
Cycle	F1	F2	F3	DECODE	INT	ACCESS	EX1	EX2	
81	j	y	n	j2 = 0X1;;	j...	xr...	n...	nop;;	
82	j	j	y	k10 = 0X15;;	j...	j3...	x...	nop; nop; nop;;	
83	j	j	j	j10 = 0;;	k...	j2...	j...	xr5 = r5 + r5;;	
84	j	j	j	xr5 = [j10 + 0X1]...	j...	k1...	j...	j3 = 0X5;;	
85	j	j	j	xr5 = [j10 + 0X1]...	B j...	j1...	k...	j2 = 0X1;;	
86	j	j	j	xr5 = [j10 + 0X1]...	B j...	B j1...	j...	k10 = 0X15;;	
87	j	j	j	xr5 = [j10 + 0X1]...	B j...	B j1...	B j...	j10 = 0;;	
88	j	j	j	xr5 = [j10 + 0X1]...	B j...	B j1...	B j...	B j10 = 0;;	
89	j	j	j	j3 = j3 - j2;;	x...	B j1...	B j...	B j10 = 0;;	
90	j	j	j	IF njeq, JUMP loop...	j...	xr...	B j...	B j10 = 0;;	
91	j	j	j	IF njeq, JUMP loop...	j...	B xr...	x...	B j10 = 0;;	

Figure B-1. Stall Due to IALU Dependency

Stalls Due to Compute Block Dependency

The following sequence of instructions causes a one-cycle stall at the Integer stage.

```
yr3 = r3 - r2;;
```

```
yr4 = r4 - r3;;    // this instruction is stalled
```

```
// in Integer for 1 cycle.
```

```
// The source register (R2)
```

```
// is a destination in the previous
```

```
// instruction.
```

Pipeline Viewer										
Cycle	F1	F2	F3	DECODE	INT	ACCESS	EX1	EX2		
41	x	j	n	yr4 = 0X5;;	nop;;	nop;;	j...	j0 = 0X1; xr0 = ...		
42	n	x	j	yr3 = 0X1;;	yr4 = 0X5;;	nop;;	n...	j0 = 0X1; xr0 = ...		
43	n	n	x	yr2 = 0;;	yr3 = 0X1;;	yr...	n...	nop;;		
44	x	j	n	n	j10 = 0;;	yr2 = 0;;	yr...	y...	nop;;	
45	x	j	x	j	n	yr3 = r3 - ...	j10 = 0;;	yr...	y...	yr4 = 0X5;;
46	x	j	x	j	x	yr4 = r4 - ...	yr3 = r3 - r2;;	j1...	y...	yr3 = 0X1;;
47	j	x	j	j	x	IF nyaeq, ...	yr4 = r4 - r3;;	yr...	j...	yr2 = 0;;
48	y	j	x	j	w	IF nyaeq, ...	yr4 = r4 - r3;;	B yr...	y...	j10 = 0;;
49	n	y	j	j		j0 = 0X1; ...	IF nyaeq, JUM...	yr...	B yr...	yr3 = r3 - r2;;
50	j	n	y	y		j10 = 0;;	j0 = 0X1; xr...	IF...	y...	B yr3 = r3 - r2;;
51	x	j	n	n		yr3 = r3 - ...	j10 = 0;;	j0...	I...	yr4 = r4 - r3;;

Figure B-2. Stall Due to Compute Block Dependency

Aborts

Examples that follow illustrate the way Pipeline Viewer displays different types of abort events for ADSP-TS101 processors. For a complete list of pipeline effects and memory transaction timing, refer to the processor's Hardware Specification.

Aborts Due to an Unpredicted Change of Flow

In the following example, the abort at the Decode stage is due to an unpredicted jump, predicated upon an IALU condition. Aborted stages in the fetch pipe are marked with , and aborted instructions in the execution pipe are marked with .

Pipeline Viewer									
Cycle	F1	F2	F3	DECODE	INT	ACCESS	EX1	EX2	
9	j	n	y	j2 = j2 - 0X1; xr9 = r...	B	k...	k...	j2 = 0X6; r13 = r5 * ...	
10	x	j	n	j2 = j2 - 0X1; xr9 = r...	B	k...	k...	k1 = 0X1; xr3 = r2 + ...	
11		j	x	IF njle, JUMP 0x5(NP); ...	j...	B	k...	k1 = 0X1; xr3 = r2 + ...	
12	n			nop;;	I...	j2...	B	k...	k1 = 0X1; xr3 = r2 + ...
13	y	n		j0 = 0X1; xr0 = r4 + r...	n...	IF...	j...	B	k1 = 0X1; xr3 = r2 + ...
14	y	y	n	j0 = 0X1; xr0 = r4 + r...	j...	nop;;	I...	j2 = j2 - 0X1; xr9 = ...	
15	n	y	y	nop;;	j...	j0...	n...	IF njle, JUMP 0x5(NP);...	
16	j	n	y	yr4 = 0X5;;	n...	j0...	j...	nop;;	
17	x	j	n	yr3 = 0X1;;	y...	nop;;	j...	j0 = 0X1; xr0 = r4 + ...	
18	n	x	j	yr2 = 0;;	y...	yr...	n...	j0 = 0X1; xr0 = r4 + ...	
19	n	n	x	j10 = 0;;	y...	yr...	y...	nop;;	

Figure B-3. Abort Due to an Unpredicted Change of Flow

Abort Due to Mispredicted Change of Flow

In the following example, an abort appears at stage E1 because of a mispredicted change of flow at the end of a loop, predicated on a compute block condition.

Pipeline Viewer										
Cycle	F1	F2	F3	DECODE	INT	ACCESS	EX1	EX2		
72		y			j	yr...	j...		IF... y... yr3 = r3 - r2;;	
73	j		y		j	yr...	y...	j1...	I... yr4 = r4 - r3;;	
74	n				j	IF...	y...	yr...	j...	IF nyaeq, JUMP loop2; ...
75	n	n			j	j0...	I...	yr...	y...	j10 = 0;;
76	j	n	n	n	j0...	j...	IF...	y...	yr3 = r3 - r2;;	
77	x	j	n	n	nop;;	j...	j0...	I...	yr4 = r4 - r3;;	
78	n	x	j	n	n...	n...	j0...	j...	IF nyaeq, JUMP loop2; ...	
79	n	n	n	x	xr...	n...	nop;;	j...	j0 = 0X1; xr0 = r4 + ...	
80	y	n	n	n	j3...	x...	n...	n...	j0 = 0X1; xr0 = r4 + ...	
81	j	y	n	n	j2...	j...	xr...	n...	nop;;	
82	j	j	y	n	k1...	j...	j3...	x...	nop; nop; nop;;	

Figure B-4. Abort Due to Mispredicted Change of Flow

Branch Target Buffer Hits

In the following example, the jump instruction was found in the Branch Target Buffer and is marked with **H**. The branch is predicted and taken, and no penalty is exacted for the change of flow.

Cycle	F1	F2	F3	DECODE	INT	ACCESS	EX1	EX2
23	y.	y	k.	IF njle, JUMP loop;...	j.	k1 ...	j..	j0 = 0X1; xr0 = r4 + r3; yr0 = r5...
24	H y.	y	y.	j0 = 0X1; xr0 = r4...	H I.	j2 ...	k..	j0 = 0X1; xr0 = r4 + r3; yr0 = r5...
25	k	H y	y.	k1 = 0X1; xr3 = r2...	j.	H IF ...	j..	k1 = 0X1; xr3 = r2 + r4; yr5 = r6...
26	y.	k	H y.	j2 = j2 - 0X1; xr9...	k.	j0 ...	H I..	j2 = j2 - 0X1; xr9 = r2 + r4; yr1...
27	y.	y	k.	IF njle, JUMP loop;...	j.	k1 ...	j..	IF njle, JUMP loop; k1 = 0X1; yr5...
28	H y.	y	y.	j0 = 0X1; xr0 = r4...	H I.	j2 ...	k..	j0 = 0X1; xr0 = r4 + r3; yr0 = r5...
29	k	H y	y.	k1 = 0X1; xr3 = r2...	j.	H IF ...	j..	k1 = 0X1; xr3 = r2 + r4; yr5 = r6...
30	y.	k	H y.	j2 = j2 - 0X1; xr9...	k.	j0 ...	H I..	j2 = j2 - 0X1; xr9 = r2 + r4; yr1...
31	y.	y	k.	IF njle, JUMP loop;...	j.	k1 ...	j..	IF njle, JUMP loop; k1 = 0X1; yr5...
32	H y.	y	y.	j0 = 0X1; xr0 = r4...	H I.	j2 ...	k..	j0 = 0X1; xr0 = r4 + r3; yr0 = r5...
33	k	H y	y.	k1 = 0X1; xr3 = r2...	j.	H IF ...	j..	k1 = 0X1; xr3 = r2 + r4; yr5 = r6...
34	v.	k	H v.	i2 = i2 - 0X1; xr9...	k.	i0 ...	H I..	i2 = i2 - 0X1; xr9 = r2 + r4; vr1...

Figure B-5. Branch Target Buffer Hits

Pipeline Viewer and Disassembly Window Operations

This section includes the following topics.

- Current Program Counter Value
- Stepping

Current Program Counter Value

The program counter value, marked by ➡ in the **Disassembly** window, is the address of the instruction at stage E1. When a breakpoint is set at a certain address, the simulator halts after the instruction at this address is executed at stage E1.

You can manually change the program counter value in the **PC Register** window to specify which instruction the simulator executes next. Note, however, that the present instruction, indicated by ➡ in the **Disassembly** window, is executed prior to the user-specified instruction. When you change a PC value, the simulator automatically flushes the pipeline and aborts all its instructions. After the pipeline is flushed, normal execution resumes from a memory address indicated by the new PC value. The simulator continues to run until an instruction at the new PC reaches stage E1 of the pipeline.



Manually changing the PC value during simulation can result in unpredictable program behavior. VisualDSP++ does not safeguard against invalid PC values. Make sure that the specified PC indicates a valid instruction within program address space.

An example of an invalid PC is a data memory address whose contents is not recognized as an instruction by the simulator. In this case, the simulator generates an unhandled software exception.

Another example of an erroneous PC is the middle of an instruction line. Only part of the instruction line is executed, while instructions in the beginning of the line are ignored.

ADSP-TS101 Processors

Figure B-6 shows how the program counter value is used for the ADSP-TS101 processor.

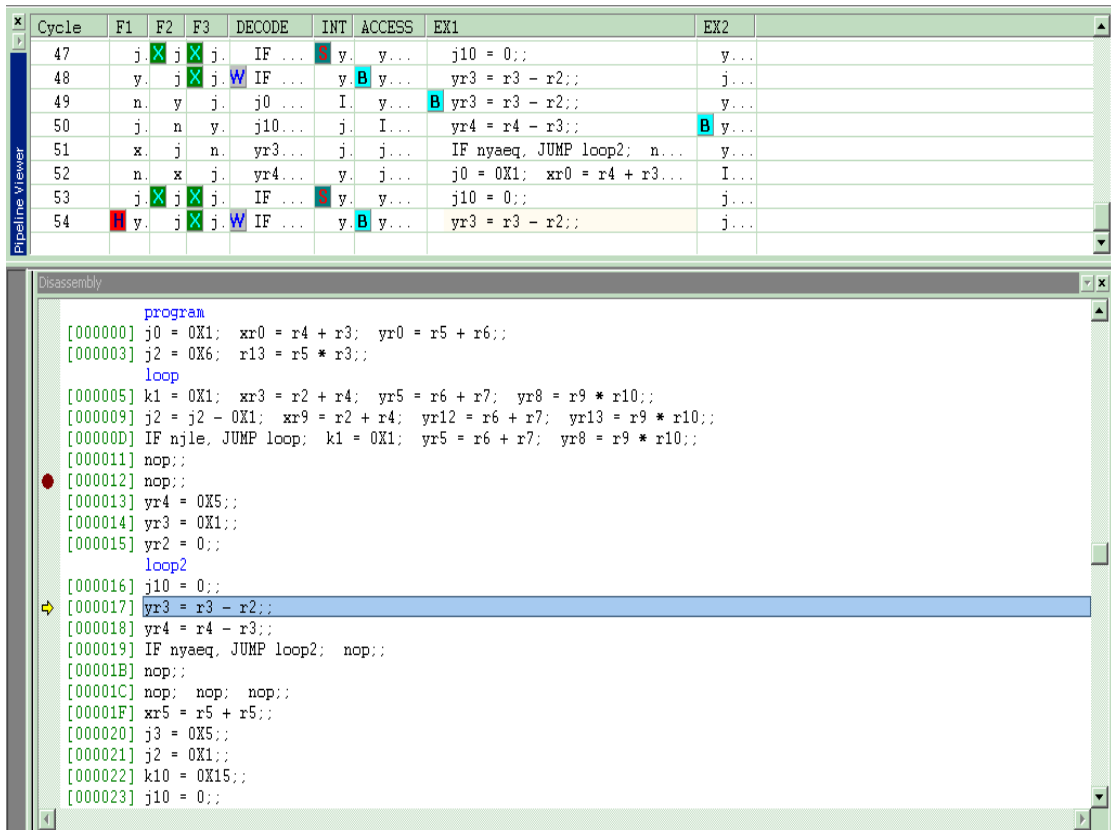


Figure B-6. Using the Program Counter Value (ADSP-TS101 Processor)

Stepping

When you single-step through a program, the simulator performs an instruction line step and skips invalid instructions (aborted, bubbles, or slots occupied by an invalid fetch).

Sometimes one step takes more than one cycle, and the **Pipeline Viewer** window advances by several lines, while the yellow arrow ➡, which marks the current program counter location in the **Disassembly** window, moves to the next instruction line.



While skipping invalid instructions (aborted, bubbles, or an invalid fetch), the simulator still processes memory transactions. When the step is completed and the memory window is updated, a surprisingly large number of new values may appear. When debugging DMA, be aware that a single step may cause several DMA transactions to be completed.

Simulator Options

The **Simulator** submenu under **Settings** provides the **Configure DMA File I/O** command. This command opens the **DMA File I/O Configuration** dialog box (Figure B-7), where you specify files as sources, destinations, or both for DMA transfers.

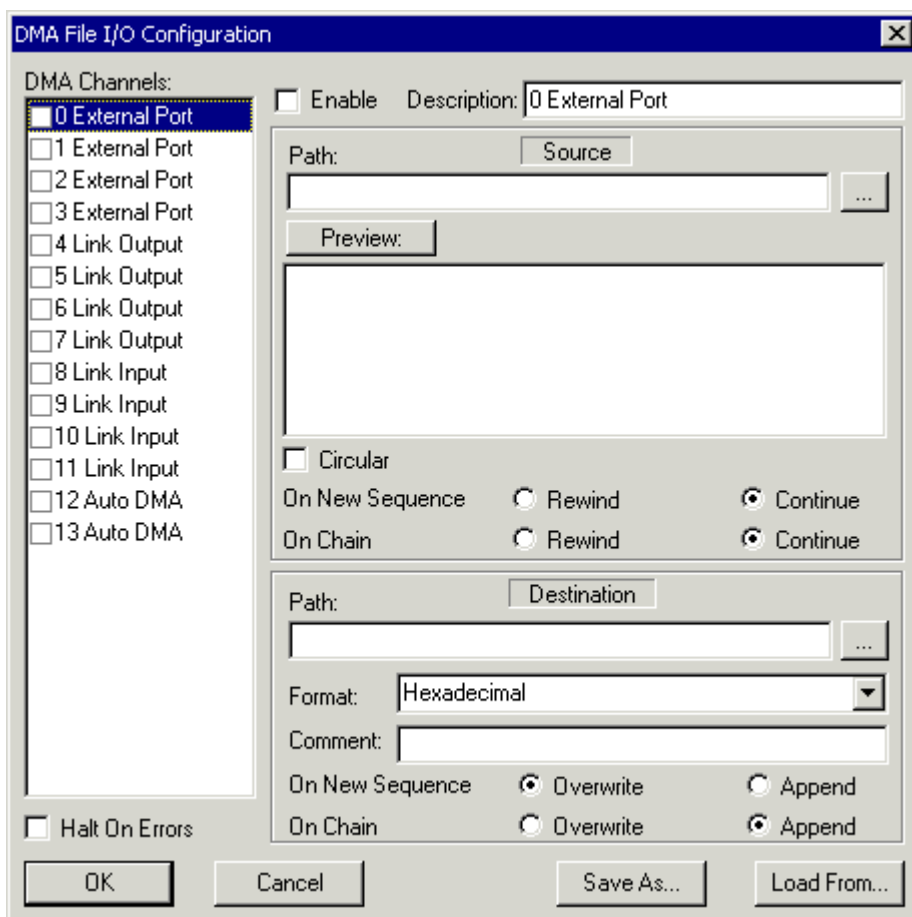


Figure B-7. DMA File I/O Configuration Dialog Box

For information about completing the dialog box and simulating a DMA transfer in the simulator, see the online Help.

ADSP-TS20x Processors

This section includes the following topics, which apply to the ADSP-TS201, ADSP-TS202, and ADSP-TS203 processors.

- Simulator Timing Analysis Overview
- Pipeline Stages
- Stalls
- Aborts
- Pipeline Viewer and Disassembly Window Operations
- Simulator Options

Simulator Timing Analysis Overview

The ADSP-TS20x simulator is a cycle-accurate simulator. It not only models the instruction set functionally, but also correctly models pipeline effects (stalls and aborts) and internal memory transactions timing.



Currently, the processor's external port and link ports are not modeled in a cycle-accurate manner. The simulator cycle-counts the code, but the cycle counts used for the external port or link ports are rough estimates of cycle counts that you could obtain by running the code on the chip. You should not rely on these counts for performance evaluation.

The **Pipeline Viewer** window helps you to understand how processor timing affects the execution of your program. For information about configuring and using the Pipeline Viewer, see [“Pipeline Viewer Window” on page 2-102](#) or the VisualDSP++ on-line Help.

Pipeline Stages

The ADSP-TS20x **Pipeline Viewer** window provides a representation of instruction flow through the processor’s pipeline. [Table B-2](#) lists the pipeline stages.

Table B-2. Pipeline Stages – ADSP-TS20x

Stage	Abbreviation in Pipeline Viewer
Instruction Fetch 1	F1
Instruction Fetch 2	F2
Instruction Fetch 3	F3
Instruction Fetch 4	F4
Predecode	PD
Decode	D
Integer	I
Access	A
Execute Stage 1	EX1
Execute Stage 2	EX2

Stages F1 through F4 represent the fetch pipe. Stages PD through EX2 represent the execution pipe. Fetch pipe stages contain raw quads of 32-bit words fetched from memory, not valid instruction lines. When you view the **Pipeline Viewer** window in disassembly format (by using the right-click menu), instructions that appear in the fetch pipe are not valid instruction lines, but merely bits and pieces. Valid instruction lines appear in the execution pipe stages.

Stalls

The examples that follow illustrate the way that Pipeline Viewer displays different types of stall events for ADSP-TS20x processors. For a complete list of pipeline effects and memory transaction timing, refer to the processor's Hardware Specification.

Stalls Due to IALU Dependency

The following sequence of instructions causes a 4-cycle stall at the PreDecode stage.

```
J0=0x40000;;
J1 = j31 + j0;;          // this instruction is stalled
                          // in PreDecode until previous
                          // instruction reaches E2:
```

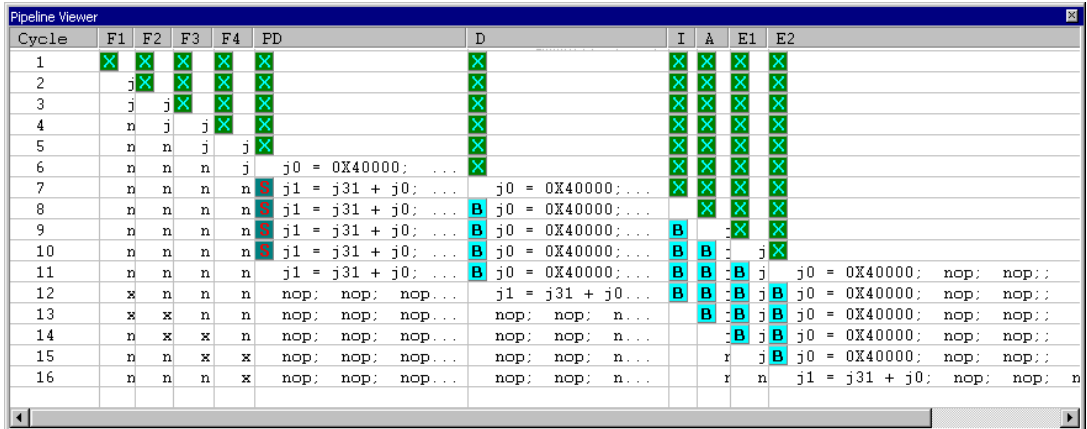


Figure B-8. Stall Due to IALU Dependency

Stalls Due to Compute Block Dependency

The following sequence of instructions causes a one-cycle stall at the Decode stage.

```

xr2 = r0 + r1;;
xr3 = r0 + r2;;    // this instruction is stalled
                   // in Decode for 1 cycle.
                   // The source register (R2)
                   // is a destination in the previous
                   // instruction.
    
```

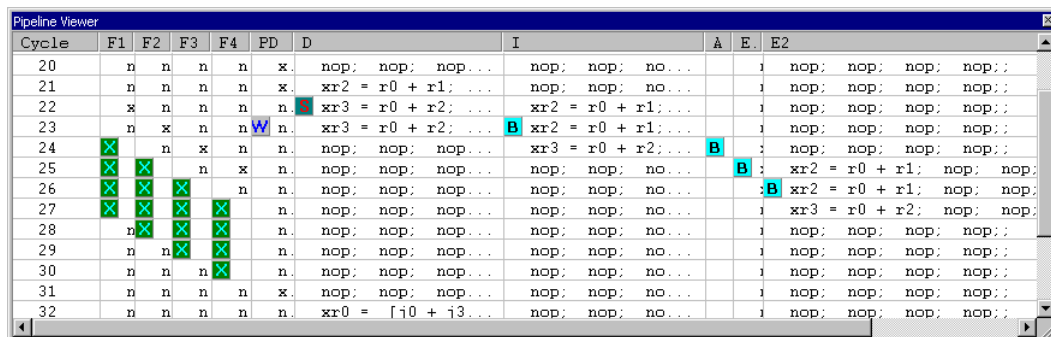


Figure B-9. Stall Due to Compute Block Dependency

Stalls Due to a Cache Miss

The following register load transaction causes a six-cycle stall because of a cache miss.

```
xr0=[j0 + j31];;
```

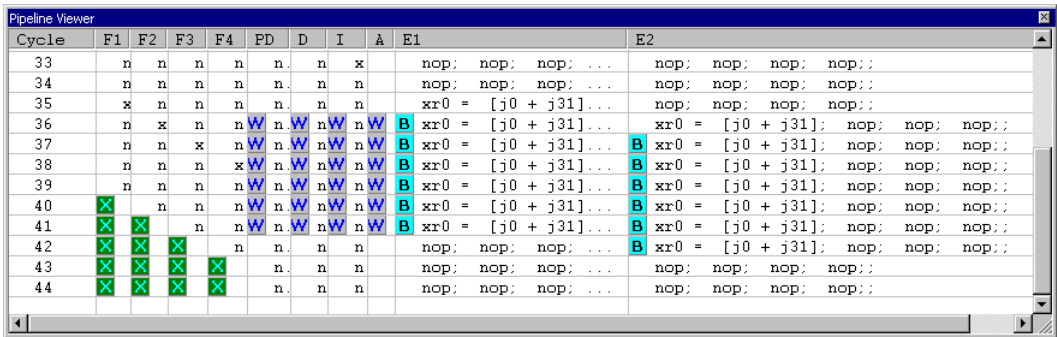


Figure B-10. Stall Due to a Cache Miss

Aborts

Examples that follow illustrate the way that Pipeline Viewer displays different types of abort events for ADSP-TS20x processors. For a complete list of pipeline effects and memory transaction timing, refer to the processor's Hardware Specification.

Aborts Due to an Unpredicted Change of Flow

In the following example, the abort at the Predecode stage is due to an unpredicted jump, predicated upon an IALU condition. The jump is unpredicted because the Branch Target Buffer is disabled in this particular example.

Aborted stages in the fetch pipe are marked with , and aborted instructions in the execution pipe are marked with .

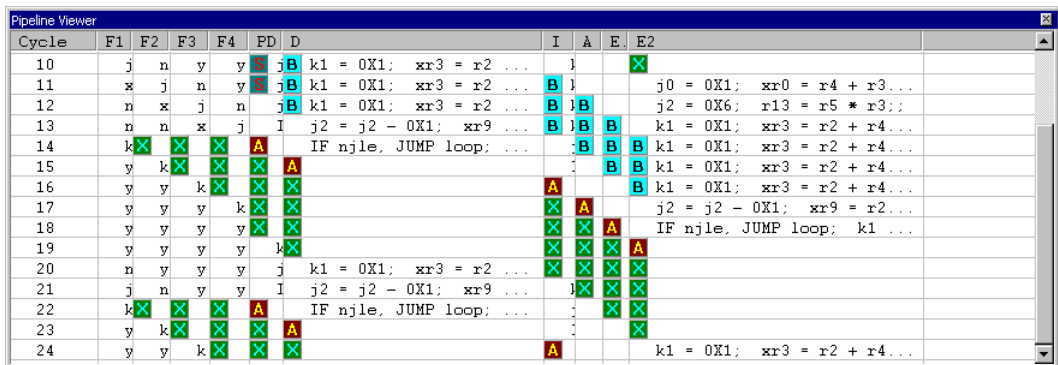


Figure B-11. Abort Due to an Unpredicted Change of Flow

Abort Due to Mispredicted Change of Flow

In the following example, an abort appears at stage E2 because of a mispredicted change of flow at the end of a loop, predicated on a compute block condition.

Pipeline Viewer											
Cycle	F1	F2	F3	F4	PD	D	I	A	E	E2	
100	X	X	X	X	H	I	y	y	H	yr4 = r4 - r3;;	
101	X	X	X	X	W	I	y	B	H	IF nyaeq. JUMP loop2; nop;;	
102	X	X	X	X		j	H	y	B	j10 = 0;;	
103	X	X	X	X		y	y	H	B	yr3 = r3 - r2;;	
104	H	y	X	X		y	y	H	B	yr3 = r3 - r2;;	
105		j	H	y	X	H	I	y	H	yr4 = r4 - r3;;	
106	H	y	j	H	y	W	I	y	B	IF nyaeq. JUMP loop2; nop;;	
107	n	X	X	X	A	A	A	A	A	j10 = 0;;	
108	n	n	X	X	X	A	A	A	A	yr3 = r3 - r2;;	
109	j	n	n	X	X	X	A	A	A	yr3 = r3 - r2;;	
110	x	j	n	n	X	X	X	A	A	yr4 = r4 - r3;;	
111	n	x	j	n	r	X	X	X	A	IF nyaeq. JUMP loop2; nop;;	
112	n	n	x	j	r	n	X	X	X		
113	y	n	n	x	x	r	n	X	X		
114	j	y	n	n	j	x	r	X	X		

Figure B-12. Abort Due to Mispredicted Change of Flow

Branch Target Buffer Hits

In the following example, the first iteration of the loop causes a four-cycle penalty because of the unpredicted change of flow. On the second iteration, the change of flow instruction was found in the Branch Target Buffer and is marked with **H**. The branch is predicted and taken, and no penalty is exacted for the change of flow.

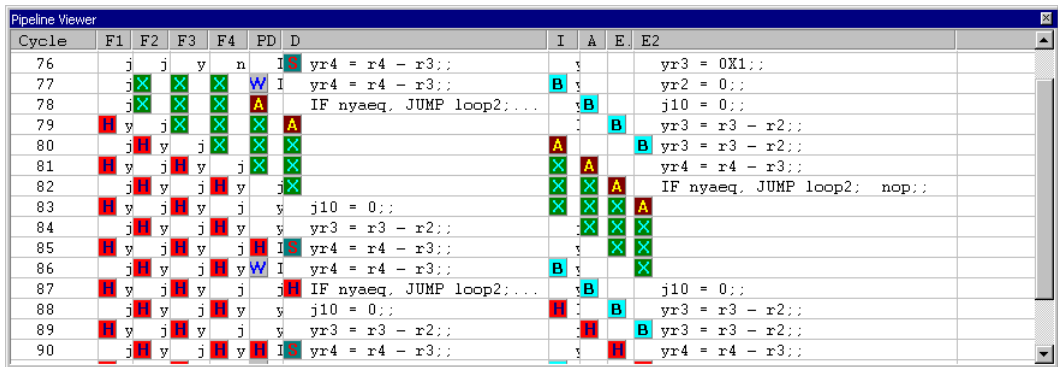


Figure B-13. Branch Target Buffer Hits

Pipeline Viewer and Disassembly Window Operations

This section includes the following topics.

- Current Program Counter Value
- Stepping

Current Program Counter Value

The program counter value, marked by ➡ in the **Disassembly** window, is the address of the instruction at stage E2. When a breakpoint is set at a certain address, the simulator halts after the instruction at this address is executed at stage E2.

You can manually change the program counter value in the **PC Register** window to specify which instruction the simulator executes next. Note, however, that the present instruction, indicated by ➡ in the **Disassembly** window, is executed prior to the user-specified instruction. When you change a PC value, the simulator automatically flushes the pipeline and aborts all its instructions. After the pipeline is flushed, normal execution resumes from a memory address indicated by the new PC value. The simulator continues to run until an instruction at the new PC reaches stage E2 of the pipeline.



Manually changing the PC value during simulation can result in unpredictable program behavior. VisualDSP++ does not safeguard against invalid PC values. Make sure that the specified PC indicates a valid instruction within program address space.

An example of an invalid PC is a data memory address whose contents is not recognized as an instruction by the simulator. In this case, the simulator generates an unhandled software exception.

Another example of an erroneous PC is the middle of an instruction line. Only part of the instruction line is executed, while instructions in the beginning of the line are ignored.

Figure B-14 shows how the program counter value is used for ADSP-TS20x processors.

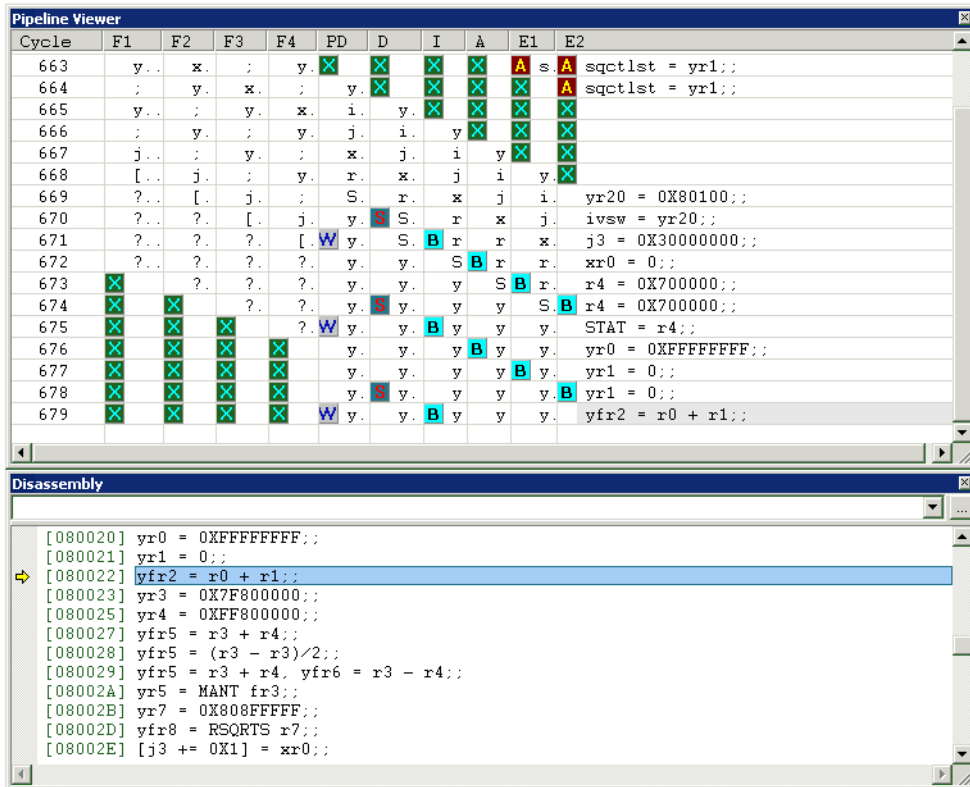


Figure B-14. Using the Program Counter Value (ADSP-TS20x Processors)

Stepping

When you single-step through a program, the simulator performs an instruction line step and skips invalid instructions (aborted, bubbles, or slots occupied by an invalid fetch).

Sometimes one step takes more than one cycle, and the **Pipeline Viewer** window advances by several lines, while the yellow arrow ➡, which marks the current program counter location in the **Disassembly** window, moves to the next instruction line.



While skipping invalid instructions (aborted, bubbles, or an invalid fetch), the simulator still processes memory transactions. When the step is completed and the memory window is updated, a surprisingly large number of new values may appear. When debugging DMA, be aware that a single step may cause several DMA transactions to be completed.

Simulator Options

The **Simulator** submenu under **Settings** provides the **Select Loader Program** command. This command opens the **Open File** dialog box and lets you specify a custom loader program. Once selected, the loader program automatically runs before a user program is loaded. The simulator defaults to a standard loader program (TS20x_prom.dxe, where x is 1, 2 or 3), but you can define your own loader by compiling a program into a .DXE file. If you create your own loader, your code must contain the label `_init_debug_end` to ensure that the loader is executed.

C SIMULATION OF SHARC PROCESSORS

Depending on your selected target processor, several simulator options are available on submenus under the **Settings** menu.

This appendix describes the options that help you simulate SHARC processors.

The information is organized as follows.

- [“Anomaly Options” on page C-2](#)
- [“Event Options” on page C-4](#)
- [“How to Record a Simulator Anomaly or Event” on page C-8](#)
- [“Select Processor ID Options” on page C-10](#)
- [“Simulator Options \(ADSP-21161Only\)” on page C-10](#)
- [“Load Sim Loader Options” on page C-11](#)
- [“SPI Simulation in Slave Mode” on page C-12](#)

Anomaly Options

The **Anomalies** submenu provides the following options to help you determine where an anomaly might be affecting your code. The default for these anomalies is OFF (disabled).

- **Shadow Write** – This command opens the **Configure Simulator Event** dialog box, from which you can configure reporting for Shadow Write anomalies.
- **SIMD FIFO** – This command opens the **Configure Simulator Event** dialog box, from which you can configure reporting for SIMD FIFO anomalies.

ADSP-21x6x Anomalies

If you know that your silicon has anomalies, the simulator lets you configure reporting for the following anomaly events.

- Shadow Write FIFO anomaly
- SIMD read from internal memory with Shadow Write FIFO hit anomaly

Shadow Write FIFO Anomaly (ADSP-2116x Only)

For examples and workarounds, refer to anomaly 39 at the Analog Devices DSP Web site.

This anomaly has been identified in the shadow write FIFOs that exist between the internal memory array of the ADSP-21160M and core I/O processor (IOP) buses that access the memory. (Refer to the Hardware Reference for more details on shadow register operation.) A particular sequence of a core write followed by a read of the same internal memory address, in conjunction with a certain type of IOP activity can cause the core read to return incorrect data.

Under the circumstances described below, the Read from Addr 1 incorrectly returns the data for Addr 2.

This problem is caused by the shadow write FIFO erroneously returning data for a core read when data should have been returned from internal memory. During write operations, data is placed in the 1st stage of a two-stage shadow write FIFO. Data is moved from first to second stage when a second write is performed (by either processor core or IOP). Similarly, data is moved from the second stage of the FIFO to internal memory when neither the core nor the IOP accesses memory in a core cycle.

On read operations, address compare logic allows data to be fetched either from internal memory or from the FIFOs. Note that each memory block has one Shadow Register FIFO, and all core and IOP accesses to internal memory use this FIFO. The internal memory clock (not visible to the user) runs at twice the core clock frequency. So, each core cycle consists of two memory cycles, with one memory cycles dedicated to the core and the other dedicated to the IOP.

SIMD Read from Internal Memory With Shadow Write FIFO Hit Anomaly (ADSP-2116x Only)

For examples and workarounds, refer to anomaly 40 at the Analog Devices DSP Web site.

This anomaly has been identified in the Shadow Write FIFOs that exist between the internal memory array of the ADSP-21160M and core /IOP busses that access the memory. (Refer to the Hardware Reference for more details on shadow register operation.)

When SIMD reads cross Long Word Address boundaries (that is, odd Normal Word addresses or non-Long Word boundary aligned Short Word addresses) and the data for the read is in the Shadow Write FIFO, the result is Revision 0.0 behavior for the read.

Event Options

The **Events** submenu provides the following options.

- **FP Denorm** – This command opens the **Configure Simulator Event** dialog box, where you can configure the reporting of the generation of a floating-point denormal result. The default is OFF (disabled).
- **Short Word Anomaly** (all ADSP-2106x processors except the ADSP-21065L) – This command opens the **Configure Simulator Event** dialog box, where you can configure reporting for short-word accesses that fail. The default is OFF (disabled).
- **Access to 21065L 9th column Even Address** (ADSP-21065L processors only) – This command opens the **Configure Simulator Event** dialog box, where you can configure reporting for invalid memory access. The default is ON (enabled).

FP Denorm

You can configure what happens when an **FP Denorm** event occurs. Denormal operands flush to zero when input to a computation unit and do not generate an underflow exception. Refer to your processor's Hardware Reference for more information about DSP floating-point operations.

Short Word Anomaly

This option applies to all ADSP-2106x processors except the ADSP-21065L processor.

Short-word accesses (read or write) fail following a stalled instruction. Any access (read or write) to short-word memory space can fail if it follows a stalled instruction or causes an instruction stall (see below for examples).

A DMA process cannot cause this anomaly. It is restricted to conditions set up in the core processor. A DMA process is not impacted by this anomaly (that is, the DMA will function correctly even if the anomaly occurs). Refer to your processor's Hardware Reference.



The occurrence of the failure varies with temperature, voltage, and frequency.

An instruction can stall because of the following circumstances.

1. DAG Stalls

An instruction that loads a DAG register followed by an instruction that uses the same DAG for a memory access causes the second instruction to stall.

```
L2= 8;  
DM(I0, M0) = R1;    // Both L2 and I0 reside in DAG1,  
                    // causing this instruction  
                    // to stall.
```

Failure can occur if I0 points to a short-word space or if the above instruction sequence is followed by a short word access.

Event Options

2. PM Memory Data Access (Cache Misses)

Any instruction that uses the PM bus to perform a data access causes an instruction stall the first time it is executed.

```
PM(I8,M8) = R1;
```

3. Memory Block Conflicts

If an instruction requires two accesses to the same memory block, the instruction stalls.

```
DM(I0, M0)=R0, PM (I8,M8) = R1;  
    // A stall will occur when both address  
    // pointers point to the same memory block.
```

4. Wait States for External Memory Accesses

An instruction that contains an external memory access where the wait state setting for that memory bank is greater than 0 or the access is held off because of the ACK signal or through bus arbitration causes that instruction to stall.

5. Multiple Bus Accesses to IOP Registers in the Same IOP Register Group

An instruction may be stalled because of multiple buses trying to access IOP registers in the same group. For this anomaly, the only applicable case is if an external host or processor accesses the same group of registers at the same time as the core.

```
Ex. DM(MSGR0) = R1;    // Instruction executed while  
                       // the host or another processor  
                       // writes to MSGR2.
```

6. Executing Instructions from External Memory with Wait States

Any instruction being executed from external memory where the wait state setting for that memory bank is greater than 0 or the access is held off because of the ACK signal causes that instruction to stall.

A workaround for cases 1–5 above is to insert a NOP between the stallable instruction and the short-word access (or remove the stall condition).

Case 6 has no workaround.



Ensure that a failing sequence does not occur when you use the delayed branch (DB) option with jumps, calls, and returns. For example, ensure that the two instructions in the RTI (DB) do not cause an instruction stall in a return to code that includes short-word accesses.

Access to ADSP-21065L Short Word Internal Memory 9th Column at Even Addresses

An Access to ADSP-21065L 9th column Even Address event is a simulator anomaly. The event occurs during access to ADSP-21065L short word internal memory 9th column even addresses. The simulator allows the event, but the processor does not.

The simulator issues a warning when the event occurs. VisualDSP++ lets you suppress the warning (in various ways). Selecting this option lets you control the simulator's behavior when the event occurs. When this option is not selected, a message box pops up and a warning appears in the **Output** window's **Console** view.

How to Record a Simulator Anomaly or Event

You can record various simulator anomalies and events for ADSP-21x6x processors.

To record a simulator anomaly or event:

1. From the **Settings** menu, choose **Anomalies** or **Events**.
2. Choose the anomaly or event to record.

The anomalies are: **Shadow Write** or **SIMD FIFO**

The events are: **FP Denorm**, **Short Word Anomaly**, or **Access to 21065L 9th column Even Address**

The **Configure Simulator Event** dialog box ([Figure C-1](#)) appears with the selected anomaly or event (for example, **FP Denorm**) displayed in the title bar of the dialog box.

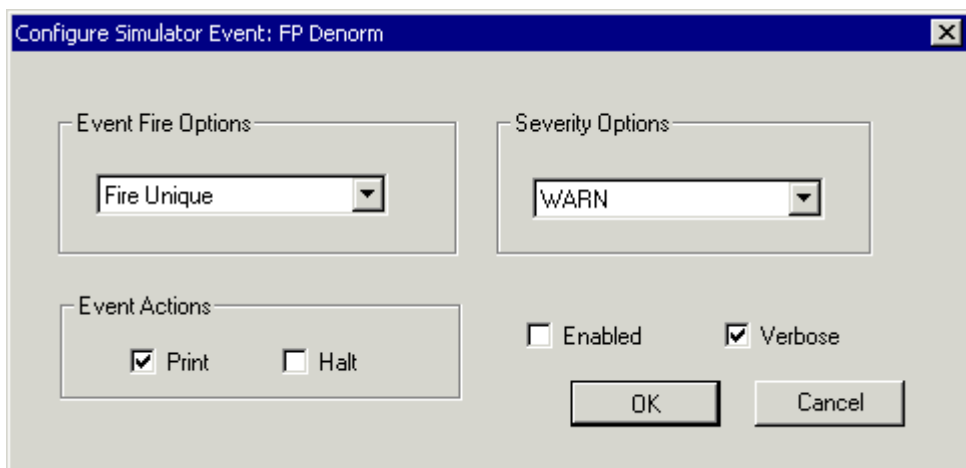


Figure C-1. Configure Simulator Event Dialog Box

Use this dialog box to configure the simulator to handle the anomaly or event.

3. Specify options, described in [Table C-1](#).

Table C-1. Options in the Configure Simulator Event Dialog Box

Item	Purpose
Event Fire Options	These options specify the frequency of reporting an event. Fire Once logs the event only the first time it occurs. Fire Unique logs the event once for each unique event (a unique set of occurrences which is specific to the event type). Select this option to prevent the reporting of multiple messages for the same event. Events (for example, FP Denorm) – If a PC has had this event before, it is not unique and is not reported. Anomalies (for example, Shadow Write, SIMD FIFO, or Short Word) – If the PC at the time of the memory write and the PC at the time of the memory read taken as a pair have had this event before, it is not unique and is not reported. Fire All logs every occurrence of the event.
Severity	This option specifies the degree of the event. INFO writes a message in black typeface to the Output window. WARN writes a message in black typeface to the Output window. ERROR writes a message appears in the Output window in red typeface and rings a bell. FATAL writes a message appears to the Output window in red typeface and rings a bell.
Event Actions	Print writes messages to the Output window. Halt halts processing after the event has occurred. Using this option is similar to using a watchpoint.

Select Processor ID Options

Table C-1. Options in the Configure Simulator Event Dialog Box

Item	Purpose
Enabled	Enables this event check
Verbose	Specifies that multi-line messages are written to the Output window When this option is not selected, messages are one line long.

4. Click OK.

Select Processor ID Options

For ADSP-2106x or ADSP-2116x processors only, you can load a processor into the simulator. You can select **Single Processor** or the processor in a multiprocessor group (for example: **Processor 1**, **Processor 2**, and so on).

Simulator Options (ADSP-21161 Only)

The **Simulator** submenu under the **Settings** menu provides the **CLKDBL** option for ADSP-21161 processors only. Use this command to double the clock speed circuitry.



Clock doubling lets you set control bits, but it does not affect how the simulator runs.

You can configure the processor's CLKOUT pin to be either 1x or 2x the rate of CLKIN. A check mark (✓) beside the **CLKDBL** command on the **Simulator** submenu indicates that this option is selected.

Load Sim Loader Options

Depending on the target processor, the **Load Sim Loader** submenu provides these boot options:

- Boot from Host (32-bit Host, 16-bit Host, or 8-bit Host)
- Boot from PROM
- Boot from Link (ADSP-21060, ADSP-21062, and ADSP-2116x processors only)
- Boot from SPI (32-bit Host, 16-bit Host, or 8-bit Host for ADSP-2116x processors only)
- None of Above (disables boot mode)

You create a boot loader file (.LDR) based on the peripheral from which you are loading. In a simulation target session, choose a peripheral (boot option) and boot file as follows.

1. From the **Settings** menu, choose **Load Sim Loader** and a boot option (such as **Boot from Host**) to open the **Open a Boot File** dialog box.
2. Navigate to the .LDR file to boot load and select it. Then click **Open** and **OK**. A message instructs you to issue a reset instruction to execute the loader.
3. From the **Debug** menu, choose **Reset**. The simulator runs and boots in the boot-kernel (also called loader-kernel). A message instructs you to issue a run instruction, which executes the loader.
4. From the **Debug** menu, choose **Run (F5)** to execute the loader. The loader runs to completion and then displays a message indicating that the loader is finished and that it is OK to run the application.

SPI Simulation in Slave Mode

 Before clicking **Run**, load the symbols for the program as follows.

- a. From the **File** menu, choose **Load Symbols** to open the **Load a Processor Program's Symbols** dialog box.
 - b. Select the **.DXE** that was used to create the **.LDR** file. A message indicates that the selected symbols are loading.
5. From the **Debug** menu, choose **Run (F5)** to run the application.

For booting information, see the Hardware Reference for your SHARC processor or the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

SPI Simulation in Slave Mode

For ADSP-21161 processors, the external SPI is not modeled in simulation. Since the master controls the timing, there is no timing information for the transfers in slave mode. Except during booting, the **BAUDR** field of the slave **SPICTL** is used for the timing of the external master. Since this field has no effect in slave mode in the hardware, you can use it for simulation.

When the SPI is enabled in slave mode and the **SPITX** buffer is not empty, an SPI transmit/receive operation occurs. If the **SPITX** and the **SPIRx** buffers are empty, a word is read speculatively into **SPIRX** buffer. The SPI reads a maximum of one word ahead. Once the **SPIRX** buffer is emptied by a DMA operation or a read, the SPI reads another word in the time specified by the **BAUDR** field of **SPICTL**.

During booting, the SPI operates at its fastest supported baud rate. When you rely on the **BAUDR** field in slave mode, a maximum value of 0x5 is used for this field. The “End of file” and “File not connected” errors are suppressed until a read of **SPIRX** or a valid DMA is set up.

Refer to the *ADSP-21161 SHARC DSP Hardware Reference* for details.

I INDEX

Symbols

.ACH files, [A-24](#)
.ASM files, [2-25](#), [A-24](#)
.BNL files, [A-24](#)
.BNM files, [A-24](#)
.BNU files, [A-24](#)
.C files, [2-25](#), [A-24](#)
.CPP files, [2-25](#), [A-24](#)
.CXX files, [2-25](#), [A-24](#)
.DAT files, [A-24](#)
.DLB files, [2-25](#), [A-24](#)
.DLO files, [A-24](#)
.DOJ files, [1-31](#), [1-33](#), [1-34](#), [2-25](#),
[A-24](#)
.DPJ files, [1-57](#), [A-24](#)
.DSP files, [2-25](#)
.DXE files, [1-34](#), [A-24](#)
.EXE files, [A-24](#)
.H files, [A-24](#)
.H# files, [A-24](#)
.HPP files, [A-24](#)
.HXX files, [A-24](#)
.IDL files, [A-24](#)
.IDM files, [A-24](#)
.IS files, [A-24](#)
.JS files, [A-24](#)

.LDF files, [1-33](#), [1-34](#), [1-35](#), [2-25](#),
[A-24](#)
.LDR files, [A-24](#), [C-11](#)
.LST files, [A-24](#)
.MAK files, [1-62](#), [A-24](#)
.MAP files, [A-24](#)
.MK, [1-62](#)
.MK files, [1-62](#), [A-24](#)
.OBJ files, [A-24](#)
.OVL files, [1-34](#), [A-24](#)
.PP files, [A-24](#)
.S files, [2-25](#), [A-24](#)
.S# files, [A-24](#)
.SM files, [1-34](#), [A-24](#)
.STK files, [A-24](#)
.TC8 files, [A-24](#)
.TCL files, [A-24](#)
.TXT files, [A-24](#)
.VBS files, [A-24](#)
.VDK files, [A-24](#)
.XML files, [A-24](#)

Numerics

3-D waterfall plots, *See* waterfall
plots

INDEX

A

- aborts, Pipeline Viewer
 - ADSP-TS101 DSPs, [B-6](#)
 - ADSP-TS20x DSPs, [B-17](#)
- About VisualDSP++ dialog box, using, [A-64](#)
- access to ADSP-21065L 9th column
 - Even Address event, [C-7](#)
- adding files to your project, [1-26](#)
- anomalies
 - ADSP-2116x DSPs, [C-10](#)
 - ADSP-21x6x processors, [C-2](#)
 - shadow write FIFO, [C-2](#)
 - short word, [C-4](#)
 - SIMD FIFO, [C-3](#)
- Anomalies submenu (SHARC), [C-2](#)
 - Shadow Write, [C-2](#)
 - SIMD FIFO, [C-3](#)
- anomaly events, recording, [C-8](#)
- archiver, [1-44](#)
- assembler, [1-33](#)
 - about, [1-33](#)
 - input files, [2-25](#)
 - terms, [1-33](#)
- assembling language files into object files, [1-33](#)

B

- background telemetry channel (BTC)
 - BTC Memory window, [2-90](#)
 - changing BTC priority, [2-89](#)
 - defining channels in your program, [2-86](#)

- bookmarking Help topics, [A-54](#)
- bookmarks, [2-56](#)
 - in editor windows, [2-56](#)
 - in online Help, [A-54](#)
- boolean operators, defining Help searches, [A-61](#)
- boot
 - kernel, [1-46](#)
 - loading or booting, [1-46](#), [C-11](#)
 - options, [C-11](#)
- boot-loadable files, [C-11](#)
- breakpoints, [3-14](#)
 - conditional, [3-15](#)
 - editor window, [2-56](#)
 - symbols, [3-14](#)
 - unconditional, [3-15](#)
- BTC Memory window, [2-90](#)
- build options
 - files, [1-28](#)
 - projects, [1-28](#)
- build project, [1-68](#)
- build settings, [1-69](#)
 - custom, [1-69](#)
 - individual file, [1-69](#)
 - project wide, [1-69](#)
- build type, *See* configuration
- buttons
 - appearance on toolbars, [2-10](#)
 - for Windows functions, [2-51](#)
 - on built-in toolbars, [2-8](#)

C

- C programs, compiling, [1-31](#)
- C++ programs, compiling, [1-31](#)

- C++ run-time libraries, [1-32](#)
- Cache Viewer, [2-107](#)
 - Address View page, [2-116](#)
 - Configuration page, [2-110](#)
 - Detailed View page, [2-111](#)
 - event log file, [2-109](#)
 - Histogram page, [2-115](#)
 - History page, [2-112](#)
 - Performance page, [2-114](#)
- Call Stack window, [2-79](#)
- channel definitions (BTC example), [2-87](#)
- CLKOUT pin, configuring, [C-10](#)
- Clock doubling, ADSP-21161
 - DSPs, [C-10](#)
- code
 - development tools, [1-2](#), [2-25](#)
 - file association with tools, [2-25](#)
- command-line parameters, [A-33](#)
- commands
 - on a control menu, [2-5](#)
 - program execution operation, [3-12](#)
 - single stepping, [3-12](#)
 - stepping, [3-12](#)
 - toolbar buttons, [A-38](#)
 - user tools, [2-13](#)
- comments
 - rules for, [A-47](#)
 - start and stop strings, [A-47](#)
- compiler, [1-31](#)
 - input files, [2-25](#)
 - options, [1-31](#), [1-33](#), [1-34](#), [1-44](#), [1-45](#)
- compiling, [1-31](#)
 - C programs, [1-31](#)
 - C++ programs, [1-31](#)
- conditional breakpoints, [3-15](#)
- configuration, [1-67](#)
 - Debug, [1-67](#)
 - plot, [2-130](#)
 - project, [1-67](#)
 - Release, [1-67](#)
- Configurator, VisualDSP++, [3-4](#)
- Configure Simulator Event dialog
 - box, [C-8](#), [C-9](#)
- configuring
 - Plot window, [2-130](#)
 - plots, [2-130](#)
- constellation plots, [3-23](#)
- control menu, [2-4](#), [2-5](#)
- conventions used in this manual, [xxx](#)
- creating
 - files to add to your project, [1-26](#)
 - new plot window, [2-130](#)
- current program counter value
 - ADSP-TS101 DSPs, [B-9](#)
 - ADSP-TS20x DSPs, [B-21](#)
- custom build
 - options, [1-28](#)
 - settings, [1-69](#)
- Custom Registers window, [2-97](#)
- customer support, [xxiii](#)
- customizing
 - plot window, [2-131](#)
 - toolbar, [2-9](#)

INDEX

D

data

- files, [1-33](#)
- input and output simulation, [3-17](#)
- sets, defined, [2-130](#)
- transfers, simulating, [1-23](#)

Debug configuration, [1-67](#)

debug sessions

- managing, [3-3](#)
- multiple, [3-3](#)
- multiprocessor, [3-4](#)
- running multiple, [3-3](#)
- selecting at startup, [3-11](#)
- setting up, [3-2](#)
- switching, [3-3](#)
- viewing list of, [3-3](#)

debugging

- features of VisualDSP++, [1-5](#)
- IDDE features, [1-5](#)
- multiple processors, [3-4](#)
- overview of, [1-23](#)
- windows used while debugging, [2-52](#)

declarations, [1-35](#)

dependencies, project, [1-68](#)

development tools, [1-2](#)

Disassembly windows, [2-59](#), [2-60](#), [2-61](#)

- examples, [2-59](#)
- features, [2-61](#)
- right-click menu, [2-62](#)
- symbols, [2-63](#)

docking, [2-46](#)

- toolbars, [2-9](#)

windows, [2-46](#)

dotprodc.dxe, automatically loading, [1-29](#)

DSP

- development tools, [1-2](#)
- plotting memory, [3-10](#)

E

editing

- features, [1-3](#)
- files, [1-27](#)

editor files, comments, [A-47](#)

Editor Tab mode, [2-31](#)

editor windows

- bookmarks, [2-56](#)
- Editor Tab mode, [2-31](#)
- expression evaluation, [2-56](#)
- features, [2-54](#)
- source mode vs. mixed mode, [2-57](#)
- symbols, [2-56](#)

elfloader.exe, [1-45](#), [1-46](#)

emulation, [1-23](#), [3-8](#)

- debug session management, [3-3](#)
- restarting the program, [3-13](#)
- statistical profiling, [3-7](#)

environment, simulating hardware, [1-23](#)

error messages, [2-32](#), [2-41](#), [2-52](#)

- in the Output window, [2-32](#)
- log file, [2-41](#), [2-42](#), [2-52](#)

evaluating expressions, [2-56](#)

event log file, [2-109](#)

- events
 - in Pipeline Viewer, [2-105](#)
 - thread, [2-121](#)
 - using the data cursor, [2-121](#)
- Events submenu (SHARC), [C-4](#)
 - Access to ADSP-21065L 9th
 - column Even Address, [C-7](#)
 - FP Denorm, [C-4](#)
 - Short Word Anomaly, [C-4](#)
- executable, loading, [3-12](#)
- Expert Linker, [1-37](#)
 - overview, [1-37](#)
 - stack and heap usage, [1-42](#)
 - window, [1-39](#)
- expressions
 - about, [2-66](#)
 - C expressions, [2-67](#)
 - context-sensitive evaluation, [2-56](#)
 - evaluating, [2-56](#)
 - in an Expressions window, [2-66](#)
 - nested, using to search Help, [A-62](#)
 - register, [2-67](#)
 - regular, [A-43](#), [A-44](#), [A-45](#), [A-46](#)
 - tagged, [A-46](#), [A-47](#)
 - types of, [2-66](#)
 - use of, [2-66](#)
 - viewing value of, [2-56](#)
 - window, [2-64](#)
- Expressions window, [2-64](#)
- extensions, DSP project file, [A-24](#)
- external interrupts, generating, [1-23](#)
- eye diagrams, [3-24](#)
 - example of, [3-24](#)
 - FIFO, [3-24](#)
- EZ-ICE target, [3-2](#)
- F
 - features
 - new in VisualDSP++, [1-7](#)
 - online Help, [A-48](#)
 - project build, [1-4](#)
 - project management, [1-4](#)
 - VDK, [1-6](#)
 - VisualDSP++, [1-2](#)
 - file and tool options, [1-28](#)
 - file building options, [1-28](#)
 - file tree, [2-15](#)
 - icons, [2-15](#)
 - Project window, [2-15](#)
 - files
 - .ASM, [2-25](#)
 - .C, [2-25](#)
 - .CPP, [2-25](#)
 - .CXX, [2-25](#)
 - .DLB, [2-25](#)
 - .DOJ, [1-31](#), [1-33](#), [1-34](#), [2-25](#)
 - .DPJ, [1-57](#)
 - .DSP, [2-25](#)
 - .LDF, [1-33](#), [1-35](#), [1-46](#), [2-25](#)
 - .MAK, [1-62](#)
 - .S, [2-25](#)
 - .VPS, [2-124](#)
 - assembler, [1-33](#)
 - associations with tools, [2-25](#)
 - automatic placement, [2-26](#)
 - building, [1-70](#)
 - compiler, [1-31](#)
 - data, [1-33](#)

INDEX

- DSP project, [A-24](#)
- event log, [2-109](#)
- executable, [1-35](#)
- extensions, [A-24](#)
- header, [1-33](#)
- in a project, [2-23](#)
- language, [1-33](#)
- linker, [1-34](#), [1-35](#), [1-46](#)
- log, [2-41](#), [2-42](#)
- nested folders in Project window, [2-22](#)
- object, [1-33](#), [1-34](#)
- overlay, [1-34](#)
- placement rules, [2-26](#)
- placing into folders automatically, [2-22](#)
- PROM, [1-44](#), [1-45](#)
- specifying build settings, [1-67](#)
- used by DSP projects, [A-24](#)
- vdk_config.cpp, [2-27](#)
- vdk_config.h, [2-27](#)
- VisualDSP_Log.txt, [2-42](#)
- finding
 - and replacing tagged expressions, [A-46](#)
 - regular expressions in find/replace operations, [A-43](#)
- Flash Programmer
 - flash devices, [3-28](#)
 - flash driver, [3-29](#)
 - functions, [3-28](#)
 - interface window, [3-30](#)
 - window controls, [3-31](#)
- Flash Programmer window, [3-30](#)
- floating, [2-49](#)
 - toolbars, [2-9](#)
 - window commands, [2-46](#)
 - windows, [2-47](#), [2-49](#), [2-50](#)
- focus
 - multiprocessor debug session, [2-98](#)
 - pinning, [3-6](#)
 - window, [3-6](#)
- folders
 - automatic file placement, [2-26](#)
 - in the Project window, [2-15](#), [2-22](#)
 - project, [2-22](#)
- FP Denorm, [C-4](#)
- functions, displaying local variables, [2-69](#)
- G**
 - General page, [1-29](#)
 - generating external interrupts, [1-23](#)
 - global build options, [1-28](#)
 - glossary, [A-2](#)
 - groups, multiprocessor, [2-100](#)
- H**
 - hardware breakpoints, [3-16](#)
 - latency, [3-16](#)
 - restrictions, [3-17](#)
 - hardware simulation, [1-23](#)
 - header files, [1-33](#)
 - heaps, usage in Expert Linker, [1-42](#)
 - Help
 - About VisualDSP++ dialog box, using, [A-64](#)

- bookmarking frequently used topics, [A-54](#)
- copying example code, [A-53](#)
- features and operations, [A-48](#)
- Help window, using, [A-48](#)
- invoking, [A-49](#)
- navigating in, [A-55](#)
- overview of Help system, [1-73](#)
- printing online documents, [A-63](#)
- printing online Help, [A-54](#)
- using navigation buttons, [A-52](#)
- using search features, [A-57](#)
- viewing context-sensitive Help, [A-50](#)

I

- I/O, hardware simulation data transfer, [1-23](#)
- icons
 - editor windows, [2-56](#)
 - Pipeline Viewer events, [2-105](#)
 - Project window, [2-18](#)
- idde.exe, command-line parameters, [A-33](#)
- IDL, *See* Interface Definition Language
- Image Viewer, [2-134](#), [3-19](#)
 - Export Image dialog box, [2-138](#)
 - Gamma Correction dialog box, [2-137](#)
 - Image Configuration dialog box, [2-137](#)
 - right-click menu, [2-136](#)

- instruction timing analysis
 - ADSP-TS101 DSPs, [B-2](#)
 - ADSP-TS20x DSPs, [B-13](#)
- Interface Definition Language (IDL), [1-54](#)
- interrupts, [3-17](#)
 - generating, [1-23](#)
 - hardware simulation, [1-23](#)

J

- JTAG emulator, [3-2](#)
 - breakpoints, [3-14](#)
 - debug session management, [3-3](#)
 - debug sessions, [3-3](#)
 - platforms, [3-2](#)
 - sampling, [3-8](#)
 - statistical profiling, [2-70](#), [3-7](#)

K

- Kernel page, [2-27](#)
- kernel, *See* VDK
- keyboard shortcuts, [A-27](#)

L

- libraries, C++ run-time, [1-32](#)
- license installation, [1-13](#)
 - client license, [1-15](#)
 - server license, [1-15](#)
 - single-user license, [1-14](#)
- license management, [1-11](#)
 - license installation, [1-13](#)
 - license status, [1-12](#)
 - licensing options, [1-11](#)
 - product serial numbers, [1-17](#)

INDEX

- product upgrades, [1-16](#)
- software registration, [1-16](#)
- temporary licenses, [1-12](#)
- valid vs. expired licenses, [1-12](#)
- validation codes, [1-16](#)
- line plots, [3-21](#)
- linear profiling, [3-7](#)
- Linear Profiling Results window, [2-70](#)
- linker
 - input files, [2-25](#)
 - overview, [1-34](#)
- Linker Description File, [1-35](#)
- linking, object files, [1-34](#), [1-35](#)
- Load Sim Loader submenu options, [C-11](#)
- loader, [1-45](#)
- loading
 - programs, [3-12](#)
 - scripts from a shortcut, [A-37](#)
- local build options, [1-28](#)
- Locals window, [2-69](#)
- locating, text using regular expressions, [A-43](#)
- log file, [2-41](#)
- logging error messages, [2-52](#)
- M
- makefiles, [1-62](#)
 - example makefile, [1-64](#)
 - Output window, [1-63](#)
 - rules, [1-63](#)
- managing
 - debug sessions, [3-3](#)
 - licenses, [1-11](#)
 - projects, [1-4](#)
 - source files, [1-4](#)
- MDI child windows, [2-46](#)
- memory
 - plots from, [3-10](#)
 - windows, [2-79](#)
- Memory Map window, [2-93](#)
- memory windows, [2-80](#), [2-82](#)
 - customization, [2-86](#)
 - examples, [2-80](#)
- menu bar, [2-6](#)
- menus
 - application menu bar, [2-6](#)
 - control menu, [2-4](#), [2-5](#)
 - title bar right-click menu, [2-4](#)
- messages
 - written to VisualDSP_Log.txt file, [2-41](#)
- mixed mode, [2-58](#)
 - editor window, [2-57](#)
 - examples, [2-58](#)
 - vs. source mode, [2-57](#)
- modes, [2-57](#)
 - mixed, [2-57](#), [2-58](#)
 - source, [2-57](#)
- multiprocessing
 - focus and pinning, [3-5](#)
 - pinning a window to a processor, [2-101](#)
 - setting the focus, [2-101](#)
 - setting up an MP debug session, [3-4](#)
- multiprocessor, [3-4](#)

- debugging, [3-4](#), [3-5](#)
- groups, [2-100](#)
- See* multiprocessor debug sessions
- window tabs, [2-99](#)
- multiprocessor debug sessions, [3-4](#)
 - debugging, [3-5](#)
 - focus, [3-6](#), [3-7](#)
 - Multiprocessor window, [2-98](#)
- Multiprocessor window, [2-98](#)
 - Groups page, [2-100](#)
 - Status page, [2-99](#)
- MyAnalog.com, [xxv](#)

N

- nested expressions, defining Help
 - searches, [A-62](#)
- nested folders, [2-22](#)
- nodes, in Project window, [2-15](#)

O

- object files, [1-33](#)
- online Help system, [A-48](#)
- operations
 - program execution, [3-12](#)
 - program execution commands, [3-12](#)
- options
 - compiler, [1-31](#)
 - file and tool, [1-28](#)
 - file building, [1-28](#)
 - project building, [1-28](#)
- Output window, [2-32](#)
 - Build page, [2-33](#)
 - capture of all messages, [2-41](#)

- Console page, [2-33](#)
- customization, [2-42](#)
- right-click menu, [2-43](#)
- overlays, files, [1-35](#)
- overriding, project-wide options, [1-69](#)

P

- PGO, *See* profile-guided optimization
- pipeline
 - stage event icons in Pipeline Viewer, [2-105](#)
- pipeline stages
 - ADSP-TS101 DSPs, [B-2](#)
 - ADSP-TS20x DSPs, [B-14](#)
- Pipeline Viewer
 - aborts (ADSP-TS101 DSPs), [B-6](#)
 - aborts (ADSP-TS20x DSPs), [B-17](#)
 - current program counter value (ADSP-TS101 DSPs), [B-9](#)
 - current program counter value (ADSP-TS20x DSPs), [B-21](#)
 - event details, [2-106](#)
 - event icons, [2-105](#)
 - pipeline stages (ADSP-TS101 DSPs), [B-2](#)
 - pipeline stages (ADSP-TS20x DSPs), [B-14](#)
 - properties, [2-104](#)
 - right-click menu, [2-103](#)
 - stalls (ADSP-TS101 DSPs), [B-3](#)
 - stalls (ADSP-TS20x DSPs), [B-15](#)

INDEX

- stepping (ADSP-TS101 DSPs),
 [B-11](#)
- stepping (ADSP-TS20x DSPs),
 [B-23](#)
- window, [2-102](#), [B-4](#), [B-5](#), [B-6](#),
 [B-7](#), [B-8](#), [B-9](#), [B-15](#), [B-16](#),
 [B-17](#), [B-18](#), [B-19](#), [B-20](#), [B-21](#)
- Pipeline Viewer window, [2-102](#)
- platforms, DSP configuration, [1-22](#)
- plot windows, [2-123](#), [2-124](#), [2-130](#)
 - capabilities, [2-124](#)
 - creating a new window, [2-130](#)
 - features, [2-127](#)
 - presentation of, [2-131](#)
 - right-click menu, [2-124](#), [2-127](#)
 - See also* plots
 - status bar, [2-124](#)
- plots
 - 3-D waterfall, [2-130](#), [3-25](#)
 - configuration of, [2-130](#)
 - constellation, [3-23](#)
 - data sets, [2-130](#)
 - DSP memory, [3-10](#)
 - eye diagram, [3-24](#)
 - line, [3-21](#)
 - presentation options, [2-133](#)
 - See also* plot windows
 - spectrogram, [3-27](#)
 - types of, [3-19](#)
 - waterfall, [3-25](#)
- plotting, DSP memory, [3-10](#)
- positioning, windows, [2-50](#)
- post-build options, [1-70](#)
- preferences
 - IDL font and color for editing,
 [1-50](#)
 - load file and advance to main,
 [1-29](#)
 - VisualDSP++ and tool output
 color, [2-33](#)
- Preferences dialog box, [1-29](#), [2-33](#)
- presentation, of plot windows,
 [2-131](#)
- printing
 - online documents, [A-63](#)
 - online Help, [A-54](#)
- processor, loading into simulator,
 [C-10](#)
- product
 - serial numbers, [1-17](#)
 - upgrades, [1-16](#)
- profile-guided optimization (PGO),
 [1-8](#)
- profiles, code analysis, [3-7](#)
- profiling, [3-7](#), [3-8](#)
 - linear, [3-8](#)
 - statistical, [3-7](#), [3-8](#)
- Program Counter (PC) register, [3-7](#),
 [3-8](#), [3-13](#), [3-15](#)
- program execution commands, [3-12](#)
- program operations
 - loading the executable program,
 [3-12](#)
 - restarting the program, [3-13](#)
 - selecting a debug session at
 startup, [3-11](#)
 - using breakpoints, [3-14](#)

- using hardware breakpoints, [3-16](#)
 - using program execution
 - commands, [3-12](#)
 - using unconditional and conditional breakpoints, [3-15](#)
 - using watchpoints, [3-15](#)
- programming tips, [1-17](#)
- project
 - build, [1-4](#), [1-58](#)
 - build settings, [1-69](#)
 - building options, [1-28](#)
 - configurations, [1-67](#)
 - debugging, [1-5](#), [1-23](#)
 - defined, [1-57](#)
 - dependencies, [1-27](#)
 - files, [2-23](#)
 - folders, [2-15](#)
 - management, [1-4](#)
 - nodes, [2-15](#)
 - options, [1-58](#)
 - subfolders, [2-15](#)
 - VisualDSP++, [1-57](#)
 - window, [2-15](#)
- Project box (showing active project), [1-60](#)
- project groups, [1-59](#)
- Project window, [1-25](#), [2-15](#), [2-27](#)
 - about, [2-15](#)
 - files, [2-15](#)
 - Kernel page, [1-25](#), [2-27](#)
 - nodes, [2-18](#)
 - Project view nodes, [2-16](#)
 - rules, [1-72](#)
 - use of folders, [2-22](#)
- projects
 - development overview, [1-17](#)
 - development stages, [1-20](#)
 - programming overview, [1-17](#)
 - project groups, [1-59](#)
- project-wide file and tool options, [1-28](#)
- PROM files, [1-45](#)
- pull-tabs, [2-47](#)
- R**
- register windows
 - custom, [2-97](#)
- regular expressions, [A-43](#), [A-44](#), [A-45](#), [A-46](#)
- Release configuration, [1-67](#)
- restarting
 - program during emulation, [3-13](#)
 - program during simulation, [3-13](#)
- right-click menus, [2-46](#)
 - commands, [2-46](#)
 - in plot windows, [2-124](#)
- S**
- SCC, *See* source code control
- scripts
 - issuing from a command line, [A-34](#)
 - issuing from a menu, [A-35](#)
 - issuing from a user tool, [A-36](#)
 - issuing from an editor window, [A-36](#)
 - issuing from the Output window, [A-34](#)

INDEX

- loading from a shortcut, [A-37](#)
- viewing script command status, [A-35](#)
- scroll bars, descriptions of, [2-47](#)
- searches
 - in online Help, [A-57](#)
 - normal, [A-43](#)
 - regular expressions vs. normal, [A-43](#)
 - special character rules, [A-45](#)
- Select Processor ID submenu
 - options, [C-10](#)
- sessions
 - debug, [3-3](#)
 - selecting at startup, [3-11](#)
- setting
 - build options, [1-69](#)
 - custom build options, [1-28](#)
- shadow write FIFO anomaly, [C-2](#)
- short word anomaly, [C-4](#)
- shortcut keys, *See* keyboard shortcuts
- SIMD FIFO, [C-3](#)
- simulating
 - data I/O streams, [3-17](#)
 - data transfers, [1-23](#)
 - hardware, [1-23](#)
 - input/output data, [3-17](#)
 - interrupts, [1-23](#)
 - SHARC processors, [C-1](#)
 - TigerSHARC processors, [B-1](#)
- simulation, [3-8](#)
 - debug session management, [3-3](#)
 - linear profiling, [3-8](#)
 - loading a processor, [C-10](#)
 - options, [C-1](#)
 - platforms, [1-22](#)
 - restarting the program, [3-13](#)
 - SPI in slave mode, [C-12](#)
- simulator
 - ADSP-21065L processors, [C-4](#), [C-7](#)
 - ADSP-2106x processors, [C-4](#), [C-10](#)
 - ADSP-21161 processors, [C-10](#), [C-12](#)
 - ADSP-2116x processors, [C-2](#), [C-3](#), [C-10](#)
 - ADSP-21x6x processors, [C-2](#), [C-8](#), [C-11](#)
 - ADSP-TS101 processors, [B-1](#)
 - ADSP-TS20x processors, [B-13](#)
 - instruction timing analysis (ADSP-TS101 processors), [B-2](#)
 - instruction timing analysis (ADSP-TS20x processors), [B-13](#)
- Simulator submenu options (SHARC), [C-10](#)
- single stepping, available commands, [3-12](#)
- software
 - registration, [1-16](#)
 - serial numbers, [1-17](#)
 - upgrades, [1-16](#)
- source code control (SCC), [1-61](#)
- source files, [1-3](#)
 - comments, [A-47](#)

- editing features, [1-3](#)
- in a project, [2-23](#)
- management, [1-4](#)
- source mode, editor windows, [2-57](#)
- source windows, *See* editor windows
- spectrogram plots, [3-27](#)
 - example of, [3-27](#)
 - FFT output, [3-27](#)
- SPI simulation in slave mode, [C-12](#)
- splitter, [1-44](#)
- stack windows, [2-97](#)
- stacks, usage in Expert Linker, [1-42](#)
- stalls, Pipeline Viewer
 - ADSP-TS101 DSPs, [B-3](#)
 - ADSP-TS20x DSPs, [B-15](#)
- Statistical Profiling Results window,
 - [2-70](#), [2-71](#), [2-72](#)
- statistical profiling, vs. linear
 - profiling, [3-7](#)
- status bar, [2-13](#), [2-14](#), [2-124](#)
 - examples, [2-13](#)
 - in plot windows, [2-124](#)
- status icons
 - editor window, [2-56](#)
 - Pipeline Viewer, [2-105](#)
- status messages, log file, [2-41](#)
- stepping, available commands,
 - [3-12](#), [3-13](#)
- stepping, Pipeline Viewer
 - ADSP-TS101 DSPs, [B-11](#)
 - ADSP-TS20x DSPs, [B-23](#)
- steps, development
 - add and edit project source files, [1-26](#)

- build a debug version of the
 - project, [1-29](#)
- build a release version of the
 - project, [1-29](#)
- create a project, [1-26](#)
- set project options, [1-26](#)
- streams, [3-17](#)
- subfolders, in the project tree, [2-15](#)
- symbols
 - editor window, [2-56](#)

T

- Target Load window, [2-122](#)
- Tcl, [A-34](#), [A-35](#)
 - menu issuance, [A-35](#)
- technical support, [xxiii](#)
- terms, VisualDSP++, [A-2](#)
- threads, [2-117](#)
 - idle, [2-122](#)
 - status, [2-117](#), [2-121](#)
- title bar, [2-4](#)
 - components, [2-3](#)
 - right-click menu commands, [2-46](#)
- toolbars, [2-7](#), [A-38](#)
 - built-in, [2-8](#)
 - button appearance, [2-10](#)
 - customization, [2-9](#)
 - docked versus floating, [2-9](#)
 - shape, [2-12](#)
- tools
 - access to, [1-3](#)
 - code development, [1-2](#), [2-25](#)
 - command line access, [1-58](#)
 - input files, [2-25](#)

INDEX

- project options, [1-58](#)
- third-party, [1-3](#)
- user configured, [2-13](#)
- Tools menu, user tools, [2-13](#)
- traces, [2-68](#), [2-120](#), [3-24](#), [3-25](#)
- U
- unconditional breakpoints, [3-15](#)
- user interface, parts of, [2-1](#)
- V
- validation codes (for creating licenses), [1-16](#)
- variables, global vs. local, [2-66](#)
- VCSE, [1-47](#)
 - component manager, [1-51](#)
 - component model, [1-48](#)
 - components, [1-47](#), [1-49](#)
 - overview, [1-47](#)
 - structure of, [1-52](#)
 - tool chain integration, [1-50](#)
 - tools, [1-49](#)
 - user interface, [1-50](#)
 - wizards, [1-51](#)
- VDK, [1-25](#), [2-27](#), [2-117](#), [2-122](#)
 - about, [1-25](#)
 - features, [1-6](#)
 - Kernel page in Project window, [2-27](#)
 - overview of, [1-6](#)
 - VDK State History window, [2-119](#)
 - VDK Status window, [2-117](#)
- VDK State History window, [2-119](#)
- VDK Status window, [2-117](#)
- vd_k_config.cpp, [2-27](#)
- vd_k_config.h, [2-27](#)
- VisualDSP++
 - control menu, [2-5](#), [2-6](#)
 - debugging, [1-5](#), [1-24](#)
 - editing features, [1-3](#)
 - editor, [2-29](#)
 - editor windows, [2-29](#)
 - environment, [1-2](#)
 - features, [1-2](#)
 - file association for tools, [2-25](#)
 - files, [A-24](#), [A-25](#)
 - glossary, [A-2](#)
 - Help system, [1-73](#), [A-48](#)
 - kernel, [1-25](#)
 - keyboard shortcuts, [A-27](#)
 - log file, [2-41](#)
 - menu bar, [2-6](#), [2-7](#)
 - overview, [1-1](#)
 - parts of, [2-2](#)
 - parts of the user interface, [2-1](#)
 - product serial numbers, [1-17](#)
 - product upgrade, [1-16](#)
 - programming overview, [1-17](#)
 - project, [1-57](#)
 - project build features, [1-3](#)
 - project development, [1-20](#)
 - Project window, [2-15](#), [2-18](#)
 - purpose, [1-3](#)
 - software registration, [1-16](#)
 - source file editing features, [1-3](#)
 - toolbar, [A-38](#)
 - tools - file association, [2-25](#)

validation codes, [1-16](#)
 VisualDSP++ Configurator, [3-4](#)

W

watchpoints, [3-15](#)
 waterfall plots, [3-25](#)
 grid of sampled data, [3-26](#)
 rotating, [3-25](#)
 windows
 BTC Memory, [2-90](#)
 Cache Viewer, [2-110](#)
 Call Stack, [2-79](#)
 Custom Registers, [2-97](#)
 debugging, [2-52](#)
 Disassembly, [2-59](#), [2-60](#), [2-61](#),
 [2-62](#)
 docked, [2-47](#), [2-48](#)
 editor, [2-54](#)
 Expert Linker, [1-39](#)
 Expressions, [2-64](#)
 Flash Programmer, [3-30](#)
 Help, [A-48](#)
 Image Viewer, [2-135](#)
 Linear Profiling Results, [2-70](#)
 Locals, [2-69](#)
 manipulation of, [2-46](#)
 MDI, [2-46](#)
 memory, [2-80](#)

Memory Map, [2-93](#)
 Multiprocessor, [2-98](#)
 Output, [2-32](#), [2-33](#), [2-43](#)
 parts of the user interface, [2-1](#)
 pipeline symbols in Disassembly
 windows, [2-58](#), [2-60](#)
 Pipeline Viewer, [2-102](#), [B-4](#), [B-5](#),
 [B-6](#), [B-7](#), [B-8](#), [B-9](#), [B-15](#), [B-16](#),
 [B-17](#), [B-18](#), [B-19](#), [B-20](#), [B-21](#)
 plot, [2-123](#)
 Project, [2-15](#), [2-19](#)
 Project view, [2-16](#)
 pull-tabs, [2-47](#)
 register, [2-94](#)
 right-click menu commands, [2-46](#)
 rules for positions, [2-50](#)
 scroll bars, [2-47](#)
See also VisualDSP++
 stack, [2-97](#)
 Statistical Profiling Results, [2-70](#)
 Target Load, [2-122](#)
 Trace, [2-67](#)
 VDK State History, [2-119](#), [2-121](#)
 VDK Status, [2-117](#)
 Windows buttons, [2-51](#)

X

X-Y plots, [3-22](#)

INDEX