

VISUALDSP++™ 3.0

Product Bulletin

for SHARC® DSPs

Revision 1.0, January 2003

Part Number
82-000321-01

Analog Devices, Inc.
Digital Signal Processor Division
One Technology Way
Norwood, Mass. 02062-9106



Copyright Information

© 2003 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, VisualDSP, the VisualDSP logo, SHARC, the SHARC logo, TigerSHARC, and the TigerSHARC logo are registered trademarks of Analog Devices, Inc.

VisualDSP++, the VisualDSP++ logo, CROSSCORE, the CROSSCORE logo, Blackfin, the Blackfin logo, and EZ-KIT Lite 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 Document	ix
Intended Audience	ix
Manual Contents	x
Technical or Customer Support	xi
Supported Processors	xi
Product Information	xii
MyAnalog.com	xii
DSP Product Information	xiii
Related Documents	xiii
Online Documentation	xiv
From VisualDSP++	xv
From Windows	xv
From the Web	xv
Printed Manuals	xvi
VisualDSP++ Documentation Set	xvi
Hardware Manuals	xvi
Data Sheets	xvi

CONTENTS

Contacting DSP Publications	xvii
Notation Conventions	xvii

INTRODUCTION

Product Description	1-2
VisualDSP++ 3.0 System Requirements	1-2
Processor Support	1-3
Benefits Provided by VisualDSP++ 3.0	1-3

NEW FEATURES AND ENHANCEMENTS

IDDE	2-2
Flash Programmer	2-2
External Makefile Execution	2-2
New Profiler Interface	2-3
Linear Profiling	2-3
Statistical Profiling	2-4
New Streams Interface	2-5
Tcl Commands	2-5
Image Viewer	2-6
Editor	2-7
Assembler	2-8
C Header and Structure Import	2-8
.IMPORT Directive	2-8
Search Option	2-9

Macros	2-9
C Structures Support	2-9
.STRUCT Directive	2-9
.EXTERN STRUCT Directive	2-10
Built-in Functions	2-10
Conditional Assembly	2-10
1.31 Fracts	2-11
Compiler and Library	2-12
Pragma Support	2-12
Data Alignment Pragas	2-13
Interrupt Handler Pragma	2-13
SIMD_for Pragma	2-14
Optimization Level Pragas	2-14
Linkage Pragas	2-15
VIDL File Preprocessing	2-15
Circular Buffer Support	2-16
Access to System Registers	2-16
Linker	2-17
Expert Linker	2-17
Linker Description File	2-18
Internal Memory Segment Width	2-18
Block 1 Alias Address Space (ADSP-216x DSPs)	2-18
Data Elimination	2-19

CONTENTS

ADSP-21161 DSP Loader	2-20
Memory Initialization Utility	2-21
Splitter	2-21
VisualDSP++ Component Software Engineering (VCSE)	2-22
Integration with VisualDSP++	2-23
VisualDSP++ Kernel (VDK)	2-24
Enhanced VDK Status Window	2-25
Enhanced State History Window	2-25
Object Protection	2-26
VisualDSP++ Component Software Engineering (VCSE)	2-27
Integration with VisualDSP++	2-28
Documentation	2-29
Online Help	2-29
Error Message Numbering and Help	2-29
Unified Index and Search	2-31
Online Manuals	2-31

DIFFERENCES BETWEEN RELEASES

Assembler and Preprocessor Differences	3-2
Command Line Switches	3-2
Directives	3-3
Functions and Operators	3-4
Macros	3-4

Compiler and Library Differences	3-6
Command Line Switches	3-6
Input and Output Files	3-9
Pragmas	3-9
Library Functions	3-11
Linker Differences	3-14
Command Line Switches	3-14
LDF and Preprocessor Macros	3-15
LDF Commands	3-15
Loader Differences	3-16
Command-Line Switches	3-16
Splitter Differences	3-17
New File Formats	3-17
VDK Differences	3-18

CONTENTS

PREFACE

Thank you for purchasing VisualDSP++™, Analog Devices development software for digital signal processor (DSP) applications.

Purpose of This Document

This document briefly describes the new features and enhancements provided by VisualDSP++ 3.0 for SHARC® DSPs, 32-bit floating-point digital signal processors. It also describes the differences between VisualDSP++ 3.0 and 2.0.

For details, refer to the VisualDSP++ 3.0 manuals for ADSP-21xxx DSPs, listed in [“Related Documents”](#) , and online Help.

Intended Audience

This publication is primarily intended for DSP programmers who are upgrading from the previous releases of VisualDSP++ development software and who want an overview of the changes to VisualDSP++ 3.0.

Manual Contents

This manual consists of:

- Chapter 1, [“Introduction”](#)

Describes VisualDSP++ 3.0 and its benefits, provides the minimal system requirements for running the product, and lists the supported processors.

- Chapter 2, [“New Features and Enhancements”](#)

Describes what is new in the VisualDSP++ 3.0 Integrated Development and Debugging Environment (IDDE), assembler, compiler, linker, loader, and documentation. Also describes the new Expert Linker (EL), VisualDSP++ Component Software Engineering (VCSE), and the changes to the VisualDSP++ Kernel (VDK), which are the key features to VisualDSP++ 3.0.

- Chapter 3, [“Differences Between Releases”](#)

Describes the differences between Release 3.0 and Release 2.0 software as they pertain to code generation tool chain: commands, switches, operators, directives, pragmas, keywords, macros, and library functions.

Technical or Customer Support

You can reach DSP Tools Support in the following ways.

- Visit the DSP Development Tools website at
www.analog.com/technology/dsp/developmentTools/index.html
- Email questions to
dsptools.support@analog.com
- Phone questions to **1-800-ANALOGD**
- Contact your ADI local sales office or authorized distributor
- Send questions by mail to
 Analog Devices, Inc.
 DSP Division
 One Technology Way
 P.O. Box 9106
 Norwood, MA 02062-9106
 USA

Supported Processors

The name “*SHARC*” refers to a family of 32-bit floating-point data type digital signal processors. VisualDSP++ 3.0 for SHARC DSPs currently supports the following processors.

- ADSP-21060 and ADSP-21060L
- ADSP-21061 and ADSP-21061L
- ADSP-21062 and ADSP-21062L
- ADSP-21065L

PREFACE

- ADSP-21160M, ADSP-21160N
- ADSP-21161N

Product Information

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

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

MyAnalog.com

MyAnalog.com is a free feature of the Analog Devices website that allows customization of a webpage to display only the latest information on products you are interested in. You can also choose to receive weekly email notification containing updates to the webpages 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 for you to select the information you want to receive.

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

DSP Product Information

For information on digital signal processors, visit our website at www.analog.com/dsp, which provides access to technical publications, datasheets, application notes, product overviews, and product announcements.

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

- Email questions or requests for information to dsp.support@analog.com
- Fax questions or requests for information to
1-781-461-3010 (North America)
+49 (0) 89 76903-157 (Europe))
- Access the Digital Signal Processing Division's FTP website at
[ftp ftp.analog.com](ftp://ftp.analog.com) or **ftp 137.71.23.21**
<ftp://ftp.analog.com>

Related Documents

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

VisualDSP++ 3.0 Getting Started Guide for SHARC DSPs

VisualDSP++ 3.0 User's Guide for SHARC DSPs

VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs

VisualDSP++ 3.0 C/C++ Compiler and Library Manual for SHARC DSPs

VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs

VisualDSP++ 3.0 Kernel (VDK) User's Guide

PREFACE

VisualDSP++ 3.0 Component Software Engineering User's Guide

VisualDSP++ 3.0 Quick Installation Reference Card

For hardware information, refer to your DSP user's or technical reference manual and data sheet.

All documentation is available online. Most documentation is available in printed form.

Online Documentation

Online documentation comprises Microsoft HTML Help (.CHM), Adobe Portable Documentation Format (.PDF), and HTML (.HTM and .HTML) files. A description of each file type is as follows.

File	Description
.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.
.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.

Access the online documentation from the VisualDSP++ environment, Windows Explorer, or Analog Devices website.

From VisualDSP++

VisualDSP++ provides access to online Help. It does not provide access to .PDF files or the supplemental reference documentation (Dinkum Abridged C++ library and FlexLM network licence). Access Help by:

- Choosing **Contents**, **Search**, or **Index** from the VisualDSP++ **Help** menu
- Invoking context-sensitive Help on a user interface item (toolbar button, menu command, or window)

From Windows

In addition to shortcuts you may construct, Windows provides many ways to open VisualDSP++ online Help or the supplementary documentation.

Help system files (.CHM) are located in the `VisualDSP\Help` folder. Manuals and data sheets in PDF format are located in the `Docs` folder of the installation CD. The installation CD also contains the Dinkum Abridged C++ library and FlexLM network license manager software documentation in the `\Reference` folder.

Using Windows Explorer

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

From the Web

To download the tools manuals, point your browser at www.analog.com/technology/dsp/developmentTools/gen_purpose.html.

PREFACE

Select a DSP 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

Printed copies of VisualDSP++ manuals may be purchased through Analog Devices Customer Service at **1-781-329-4700**; ask for a Customer Service representative. The manuals can be purchased only as a kit. For additional information, call **1-603-883-2430**.

If you do not have an account with Analog Devices, you will be referred to Analog Devices distributors. To get information on our distributors, log onto www.analog.com/salesdir/continent.asp.

Hardware Manuals

Printed copies of hardware reference and instruction set reference manuals can be ordered through the Literature Center or downloaded from the Analog Devices website. The phone number is **1-800-ANALOGD (1-800-262-5643)**. The manuals can be ordered by a title or by product number located on the back cover of each manual.

Data Sheets

All data sheets can be downloaded from the Analog Devices website. As a general rule, printed copies of data sheets with a letter suffix (L, M, N, S) can be obtained from the Literature Center at **1-800-ANALOGD (1-800-262-5643)** or downloaded from the website. Data sheets without

the suffix can be downloaded from the website only—no hard copies are available. You can ask for the data sheet by part name or by product number.

If you want to have a data sheet faxed to you, the phone number for that service is **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.

Contacting DSP Publications

Please send your comments and recommendations on how to improve our manuals and online Help. You can contact us by:

- Emailing `dsp.techpubs@analog.com`
- Filling in and returning the attached Reader's Comments Card found in our manuals

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.

Example	Description
Close command (File menu) or OK	Text in bold style indicates the location of an item within the VisualDSP++ environment's menu system and user interface items.
{this that}	Alternative required items in syntax descriptions appear within curly brackets separated by vertical bars; read the example as <i>this</i> or <i>that</i> .
[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> .

PREFACE

Example	Description
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipsis; read the example as an optional comma-separated list of <i>this</i> .
.SECTION	Commands, directives, keywords, code examples, and feature names are in text with <i>letter gothic</i> font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
	A note providing information of special interest or identifying a related topic. In the online version of this book, the word Note appears instead of this symbol.
	A caution providing information about critical design or programming issues that influence operation of a product. In the online version of this book, the word Caution appears instead of this symbol.

1 INTRODUCTION

This chapter describes the product, VisualDSP++, and the requirements for running its latest revision, 3.0. It also lists the supported processors and some of the benefits provided by this release.

The information is organized as follows.

- “Product Description” on page 1-2
- “VisualDSP++ 3.0 System Requirements” on page 1-2
- “Processor Support” on page 1-3
- “Benefits Provided by VisualDSP++ 3.0” on page 1-3

Product Description

VisualDSP++ is Analog Devices project management and development environment for Digital Signal Processor (DSP) applications.

VisualDSP++ 3.0 successfully integrates a graphical user interface and code generation and debugging tools, enabling programmers to move easily between editing, building, debugging, and deployment of final products.

The code generation tool chain is comprised of the processor-specific software necessary for completing a SHARC DSP based project: simulator, assembler, C/C++ compiler and libraries, linker, loader, splitter, VisualDSP++ Component Software Engineering (VCSE), VisualDSP++ Kernel (VDK), and utilities.

The product CD-ROM also includes an evaluation suite of the EZ-KIT Lite™ software, which provides an easy method for initial evaluation of a target processor system and allows application prototyping.

The successor to VisualDSP++ 2.0, this software release incorporates a number of new features and enhancements, as described in [“New Features and Enhancements” on page 2-1](#).

VisualDSP++ 3.0 System Requirements

To install and run VisualDSP++ 3.0, your PC must provide the following software, configuration, and system resources.

- Windows 98/NT 4.0/SP3/2000/ME/XP
- At least 100 MB of available hard drive space
- At least 32 MB of RAM
- CD-ROM drive
- Internet Explorer 4.01 or later

Processor Support

VisualDSP++ 3.0 supports the following SHARC processors.

- ADSP-21060/60L
- ADSP-21061/61L
- ADSP-21062/62L
- ADSP-21065L
- ADSP-21160M/60N
- ADSP-21161N

Benefits Provided by VisualDSP++ 3.0

Some of the benefits provided by VisualDSP++ Release 3.0 include:

- **Rapid application development (RAD).** The tight integration between the VisualDSP++ environment and the underlying tools enables you to develop applications rapidly, as you do not have to create all the required control code manually. Using automatic code generation and file templates, you can concentrate on algorithms and control flow rather than on implementation details. VisualDSP++ provides both the means to develop code that is easy to read and maintain and the option to optimize manually.
- **Easy-to-use debugging activities.** Debug with one common, easy-to-use interface for all processor simulators and emulators, or hardware evaluation and development boards. Switch easily between these targets.

Benefits Provided by VisualDSP++ 3.0

- **Multiple language support.** Debug programs written in C, C++, or assembly, and view your program in machine code. For programs written in C/C++, you can view the source in either C/C++ or mixed C/C++ and assembly, and can display the values of local variables or evaluate expressions (global and local) based on the current context.

VisualDSP++ 3.0 offers greater ease in intermixing C/C++ language control code with your assembly language algorithms. The assembler now supports directives that enable easy access to C-language data structures. Should your C structures change, the accompanying assembly language code is updated without the need for any recoding.

- **Effective debug control.** Set breakpoints on source line numbers, symbols, and addresses and then step through the program's execution to find problems in coding logic. Set watchpoints (conditional breakpoints) on registers, stacks, and memory locations to identify when they are accessed.
- **Tools for improving performance.** Use the trace and linear and statistical profiles to identify bottlenecks in your DSP application and to identify program optimization needs. Use plotting to view data arrays graphically. Generate interrupts, outputs, and inputs to simulate real-world application conditions.
- **Multiprocessor debugging.** You can easily manage and debug any number of processors from the same debug session. Fully synchronous multiprocessor operations such as step, run, and halt allow debugging of complex systems. You can arrange processors into logical groups for better control of your system's behavior.

- **Code reusability.** Programmers often begin a new project by writing infrastructure portions that transfer data between algorithms. The VisualDSP++ Kernel provides this control logic as a standard, portable, and reusable library. The kernel's design takes advantage of commonality in user applications and encourages code reuse.

Each thread of execution is created from a user-defined template, either at boot time or dynamically by another thread. Multiple threads can be created from the same template, but the state associated with each created instance of the thread remains unique. Each thread template represents a complete encapsulation of an algorithm that is unaware of other threads in the system unless it has a direct dependency.

The kernel also promotes good coding practice and organization by partitioning large applications into blocks or subsystems of functionality that are easy to maintain.

- **Flexibility and extensibility.** VisualDSP++ Component Software Engineering tools simplify the process of developing, documenting, and validating reusable software components. VisualDSP++ support includes the means to display information about available components and to incorporate them into a project. You use the VCSE Interface Definition Language (VIDL) to define interfaces and the components that implement them.

VisualDSP++ 3.0 provides a VIDL compiler that enables you to create, use, and reuse components without becoming familiar with the detail of the model and mechanism involved. VisualDSP++ 3.0 also provides wizards for creating interfaces and components.

- **Graphical presentation of memory allocation.** Expert Linker parses a project's .LDF file and graphically displays the mapping of a target DSP's memory and the project's object sections. You can use drag-and-drop actions to arrange the object sections within a

Benefits Provided by VisualDSP++ 3.0

graphical representation of memory. When you are satisfied with the memory layout, and the project is ready to be built, Expert Linker saves the changes to the `.LDF` file.

EL is able to show graphically how much space is allocated for your program's heap and stack. After loading and running the program, it can show how much of the heap and stack has been used. You can interactively reduce the amount of space allocated to heap or stack if more memory than required is used. This allows you to free up that memory to store other memory objects, such as DSP code and data.

2 NEW FEATURES AND ENHANCEMENTS

VisualDSP++ 3.0 has a number of new features and enhancements designed to increase productivity and shorten application development cycles. This chapter describes the new features and enhancements introduced in Integrated Development and Development Environment (IDDE), tools, and documentation.

The information is presented as follows.

- [“IDDE” on page 2-2](#)
- [“Assembler” on page 2-8](#)
- [“Compiler and Library” on page 2-12](#)
- [“Linker” on page 2-17](#)
- [“ADSP-21161 DSP Loader” on page 2-20](#)
- [“Splitter” on page 2-21](#)
- [“VisualDSP++ Component Software Engineering \(VCSE\)” on page 2-22](#)
- [“VisualDSP++ Kernel \(VDK\)” on page 2-24](#)
- [“Object Protection” on page 2-26](#)
- [“Documentation” on page 2-29](#)

IDDE

The IDDE provides these new features and enhancements:

- “Flash Programmer” on page 2-2
- “External Makefile Execution” on page 2-2
- “New Profiler Interface” on page 2-3
- “New Streams Interface” on page 2-5
- “Image Viewer” on page 2-6
- “Editor” on page 2-7

For more information about IDDE, refer to the *VisualDSP++ 3.0 User's Guide for SHARC DSPs* and online Help.

Flash Programmer

The VisualDSP++ Flash Programmer (under **Tools** and **Flash Programmer**) provides a convenient graphical interface to on-board flash memory devices. Use the Flash Programmer to view various flash memory information, fill or erase portions of flash memory, or load an Intel hex formatted file into flash memory.

Analog Devices supplies flash drivers (algorithms) for EZ-KIT Lite evaluation systems, including the ADSP-21xxx DSP EZ-KIT Lites. Additionally, the Flash Programmer allows you to create custom flash drivers or modify existing drivers for use on custom boards with a variety of processor and flash device combinations.

External Makefile Execution

The IDDE has been enhanced to support the loading and building of external `gmake`-compatible makefiles. This feature enables you to develop,

maintain, and use your own external makefile without having to leave the IDDE. When you open a makefile, it is displayed in the **Project** window as a single file node. Double-clicking on the makefile icon in the **Project** window opens the makefile on the editor window. A syntax highlighting is provided for easier makefile editing.

The IDDE does not consume, parse, or perform any type of syntax checking on the makefile and does not modify the loaded makefile in any way. While a makefile is opened, all project related commands, with the exception of **Build** and **Build All**, are disabled. When a **Build** or **Build All** command is issued, the `gmake` engine is executed and all output is displayed on the **Build** page of the **Output** window.

If an error occurs, the error text, which includes the line number and name of the file, appears on the **Build** page of the **Output** window. The error text conforms to the syntax described in the section [“Error Message Numbering and Help” on page 2-29](#). Double-clicking the text opens the specified file and places the cursor on the line that has the error.

New Profiler Interface

The legacy profiling tool supported by simulators has been merged with the new statistical/linear profiler.

The **Profile** option has been removed from the **Tools** menu and from the **View/Debug Windows** menu since the legacy profiler is no longer required.

Linear Profiling

A new linear profiling interface is available for the ADSP-21xxx DSP simulators. Emulators continue to use the statistical profiling interface. A new menu command (**New Profile**) under **Tools --> Linear Profiling** opens the **Linear Profiling Results** window.

Settings in the **Profile Window Properties** dialog box control which and how data items are utilized in the profiling interface. To open the properties dialog box, right-click in the **Linear Profiling Results** window.

Settings on the **Display** page control which profiling data items are displayed. In addition to ‘traditional’ profiling information, the linear profiler yields cache hits and misses.

Settings on the **Filter** page control which data items are profiled:

- **Entire memory space.** The linear and statistical profilers have the same behavior as in VisualDSP++ 2.0.
- **C/C++ functions.** Functions are selected from a list of available functions. The profiler discards any collected program counter (PC) samples that are not within the selected functions.
- **Memory ranges.** The start and end addresses of the memory ranges are specified. The profiler discards any collected PC samples that are not within the selected memory ranges.

Most of the time, when building debug versions of the code, you profile without concern for non-debug code. A check box provides the option of filtering out any PC samples that have no corresponding debug information. If you put a check mark in the box, the profiler discards any PC samples that have no debug information.

Statistical Profiling

The **Statistical Profiling Results** window has been modified to include a horizontal scroll bar at the bottom of the window. The scroll bar is useful when the window is not big enough to display both the histogram and source panes.

New Streams Interface

The **Streams** dialog box (under the **Settings** menu) has been overhauled to make the process of creating or modifying a stream easier. The dialog box contains a list of current streams. Add new streams by pressing the **Add** button to invoke the **Add New Stream** dialog box to guide you through the setup procedure. Modify current streams by selecting a stream from a list of current streams and pressing the **Edit** button. The **Edit Stream** dialog box appears to guide you through the edit procedure.

The settings for the streams are automatically saved and restored to ensure you do not have to set up the streams every time you start the IDDE. The settings are saved on a per-session basis.

Tcl Commands

VisualDSP++ 3.0 has extended Tcl command set (version 8.3) with several procedures for steaming data into and from a simulated DSP target. The current release introduces the following Tcl commands for stream management.

Table 2-1. New Tcl Commands

Command	Description
dspaddstream	Adds a stream to the debug session.
dspdeletestream	Deletes the stream identified by ID.
dspdeleteallstream	Deletes all the streams in the debug session.
dspliststream	Returns the list of streams. The list consists of the ID of the stream, source device or a file, destination device or a file.

For more information, see the *VisualDSP++ 3.0 User's Guide for SHARC DSPs* and online Help.

Image Viewer

VisualDSP++ allows you to display image and video data stored in a PC image file or DSP memory. Supported file types include .BMP, .JPEG, .PPM, and .MPEG.

The following dialog boxes control the Image Viewer settings.

- The **Configuration** dialog box defines the image source (memory or file) and image attributes. If the image is located in DSP memory, the image address, size, and format must be specified. The image is read from DSP memory, based on the parameters entered. If the image is located in a file, the image size is determined automatically. The image data is read from the file and is transferred to DSP memory, starting at the specified start address.
- The **Image Viewer** window renders the image. Scroll bars are enabled if the image is larger than the window. Buttons for zooming the image in and out are available. Each press of the zoom-in or zoom-out button enlarges or decreases the size of the image by a factor of 2. As the mouse is moved over the image, the status bar displays:
 - DSP address where the selected pixel is located
 - Red, green, and blue (RGB) pixel values
 - Column and row in pixel coordinates
- The **Gamma Correction** dialog box provides slider controls for adjusting the red, green, and blue (RGB) pixel values of the image. Linking the sliders adjusts the brightness of the image. Clicking **OK** writes the adjusted image data to DSP memory.
- The **Export** dialog box enables the image to be sent to a printer or copied to the clipboard or a file.

- The **Property** dialog box displays image an image's address and other attributes.
- The **Play Video** dialog box allows to play a video clip of MPEG data stored in DSP memory.

Editor

Enhancements to VisualDSP++ 3.0 editor windows include:

- Right-click menu command that toggles the display of line numbers
- Brace matching based on the position of the text cursor
- Auto-positioning of closing curly brackets for proper alignment with preceding opening brace
- Right-click command to open header files from the file name in an `#include` statement
- Support for dragging and dropping text into an open **Expressions** window

For more information on the graphical user interface, refer to the *VisualDSP++ 3.0 User's Guide for SHARC DSPs* and online Help.

Assembler

VisualDSP++ 3.0 assembler for SHARC DSPs provides the following new features:

- “C Header and Structure Import” on page 2-8
- “C Structures Support” on page 2-9
- “Conditional Assembly” on page 2-10

For more information, refer to the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs* and online Help.

C Header and Structure Import

The assembler can import C header files and access structures defined in C header files. The following directives, command-line options, and macro definitions support this feature.

.IMPORT Directive

The `.IMPORT` directive makes C `struct` layouts (declared in the file being imported) visible inside an assembly program. The `.IMPORT` file is a C header file.

The `.IMPORT` directive provides the assembler with the `typedef` and `struct` names as well as the name, sequence, size, and offset of each data member. This allows referencing the `struct` data members by name in assembly files, just as in C. The assembler calculates the offsets within the structure based on the size and sequence of the data members. Note that the `.IMPORT` directive does not allocate space for a variable of a `struct` type—that requires the `.STRUCT` directive.

If the imported file is modified, the assembly file will be re-assembled during the next build, updating any offsets that may have changed.

Search Option

The assembler calls the compiler to process each header file in an `.IMPORT` directive. The `-flags-compiler -opt1 [-opt2...]` switch takes a list of one or more comma-separated options, which are passed onto the compiler command line for `.IMPORT` header compilations. Use this switch to override the order in which the `include` directories are searched. The `-flags-compiler` options always take precedence over the assembler's `-I` options.

Macros

For each `.IMPORT` header, the assembler calls the compiler driver with the appropriate preprocessor option. The compiler sets the machine constants accordingly to the preprocessor and defines `_LANGUAGE_C` as 1. The macro replaces `_LANGUAGE_ASM`, which is present by default.

C Structures Support

Now the assembler can call the compiler to process structure definitions provided by imported C header files. The following directives and builtins support this feature.

.STRUCT Directive

The `.STRUCT` directive allows you to define and initialize high-level data objects within the assembly file in which the directive appears. The `.STRUCT` directive creates a `struct` variable using a C style typedef, as it follows from the `.IMPORT` C header file(s). Structure initialization can be of the long or short form. The difference is that the long form includes data member names and is assembler-specific, while the short form does not include data member names and corresponds to the syntax in C `struct` initialization.

.EXTERN STRUCT Directive

The `.EXTERN STRUCT` directive allows an assembly code module to reference a `struct` that is defined and initialized in another assembly file or in C compiled code. Code in the assembly file then can reference the data members by name, just as if they are declared locally.

Built-in Functions

The assembler supports the `offsetof()` and `sizeof()` built-in functions for passing information obtained from the imported C `struct` layouts.

The `offsetof()` builtin is used to calculate the offset of a specified member from the beginning of its parent data structure. For ADSP-21xxx DSPs, the `offsetof()` units are in words.

The `sizeof()` builtin returns the amount of storage associated with an imported C `struct` or data member. It provides functionality similar to its C counterpart. This builtin returns the size in units appropriate for the target processor. For SHARC DSPs, the `sizeof()` units are words.

For more information about C structures support, refer to the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs* and online Help.

Conditional Assembly

VisualDSP++ 3.0 introduces conditional assembly directives for evaluation of assembly time constants using relational expressions. The conditional assembly directives are: `.IF`, `.ELIF`, `.ELSE`, and `.ENDIF`. These directives are in addition to the C style preprocessing directives `#if`, `#elif`, `#else`, and `#endif`. The expressions may include relational and logical operators.

The assembler supports nested conditional directives. The outer conditional result propagates to the inner condition, just as it does in C preprocessing. These directives are reserved keywords.

Refer to the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs* and online Help for more information.

1.31 Fracts

The assembler supported fracts use 1.31 format, represented by a sign bit and “31 bits of fraction”. This is in the -1 to $+1-2^{**31}$ range. For example, the 1.31 fract format maps the constant $0.5r$ to 2^{**31} .

For more information, see the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs* and online Help.

Compiler and Library

VisualDSP++ 3.0 C/C++ compiler for SHARC DSPs generates efficient code, optimized for both size and execution. Of the new compiler's features and enhancements, the most notable are:

- [“Pragma Support” on page 2-12](#)
- [“VIDL File Preprocessing” on page 2-15](#)
- [“Circular Buffer Support” on page 2-16](#)
- [“Access to System Registers” on page 2-16](#)

The compiler also includes a number of processor-specific enhancements. For information on these enhancements, refer to the *VisualDSP++ 3.0 C/C++ Compiler and Library Manual for SHARC DSPs* and online Help.

Pragma Support

The compiler supports a number of `#pragmas`. These pragmas are implementation-specific directives that modify the compiler's behavior for:

- Data alignment
- Interrupt handling
- SIMD processing (ADSP-2116x DSPs)
- General optimization
- External module linkage

Data Alignment Pragas

The data alignment pragmas modify how the compiler arranges data within DSP memory. The alignment pragmas include `align num`, `pack(alignopt)`, and `pad(alignopt)`.

The `#pragma align num` is used to alter the alignment of variables and elements within a structure. The pragma applies to a declaration that immediately follows the pragma. The pragma's effect is that the next data element is forced to be aligned on a boundary specified by `num`. Alignments specified using this pragma must be a power of 2.

The structure may need padding via `#pragma pad(alignopt)` to ensure all fields are properly aligned and that the structure's overall size is a multiple of its alignment. Changing the structure's data alignment via `pack(alignopt)` reduces the structure's size. If `pack` or `pad` are currently enabled, then using `align` will override the alignment of the immediately followed declaration.

For detailed descriptions of the data alignment pragmas, refer to the corresponding section in the *VisualDSP++ 3.0 C/C++ Compiler and Library Manual for SHARC DSPs* and online Help.

Interrupt Handler Pragma

The `#pragma interrupt` is applied to the function declaration or definition that immediately follows the pragma and causes the compiler to generate the function code that can be used as a self-dispatching interrupt handler. The compiler ensures that all used registers are restored at the end of the function.

For a detailed description of the interrupt handler pragma, see the corresponding section in the *VisualDSP++ 3.0 C/C++ Compiler and Library Manual for SHARC DSPs* and online Help.

SIMD_for Pragma

The `#pragma SIMD_for` pragma is used with ADSP-2116x DSPs and enables the compiler to take advantage of Single-Instruction, Multiple-Data (SIMD) operations. The `SIMD_for` pragma is applied to the loop instructions that immediately follow the pragma line and causes the compiler to process the instructions in SIMD mode if the loop is suitable for SIMD execution.

Certain conditions can limit the number of cases in which SIMD operations may occur. The compiler attempts to check these conditions and issues warnings or errors when a potential problem is detected.

For a detailed description of the SIMD pragma, see the corresponding section in the *VisualDSP++ 3.0 C/C++ Compiler and Library Manual for SHARC DSPs* and online Help.

Optimization Level Pragmas

The `optimize_off`, `optimize_for_space`, and `optimize_for_speed` pragmas change the optimization level while a function, whose definition immediately follows the pragma line, is being compiled.

The `#pragma optimize_for_space` and `#pragma optimize_for_speed` turn the optimizer on or, if optimization is already enabled, switch focus between reducing code size and increasing performance, respectively. The `optimize_for_space` pragma has the same effect as setting the `-O` switch. The `optimize_for_speed` has the same effect as setting the `-Os` switch.

The `#pragma optimize_off` turns off the optimizer and has the same effect as compiling with optimization disabled (`-O0`).

These pragmas are subject to the following restrictions.

- They apply only to back-end optimizations. Front-end optimizations always work with the optimizations specified on the command line or by the IDDE.
- They are valid only before a function definition.
- They are ignored if optimization is not requested in the IDDE or on the command line.
- They are ignored if Interprocedural Analysis (IPA) is enabled.

Linkage Pragmas

The linkage pragmas, `linkage_name` and `retain_name`, change how a given global function or variable is viewed during the linking stage.

The `#pragma linkage_name` associates the named identifier with the next external function declaration. It ensures the identifier is used as the external reference, overriding the default compiler behavior.

The `#pragma retain_name` indicates that the external function or variable declaration that follows the pragma is not removed even though IPA detects that the `name` is not utilized.

VIDL File Preprocessing

VisualDSP++ 3.0 C/C++ compiler supports the preprocessing of Interface Definition Language (VIDL) specifications (`.IDL`) for VCSE components and interfaces. Every specification is analyzed by the preprocessor prior to syntax analysis. Refer to the *VisualDSP++ 3.0 Component Software Engineering User's Guide* and online Help for details.

Circular Buffer Support

Two new built-ins are offered to support the SHARC DSP's circular buffer mechanisms:

<code>__builtin_circindex()</code>	Circular buffer increment of an index.
<code>__builtin_circptr()</code>	Circular buffer increment of a pointer.

Access to System Registers

Four new built-ins are offered to provide access to system registers. The functions are defined in the `sysreg.h` header file, as follows.

<code>__builtin_sysreg_write_nop()</code>	Toggles all the bits of the nominated system register that set in the supplied bit mask and places a NOP ; statement after the instruction.
<code>__builtin_sysreg_bit_clr_nop()</code>	
<code>__builtin_sysreg_bit_set_nop()</code>	Toggles all the bits of the nominated system register that are set in the supplied bit mask and places a NOP ; statement after the instruction.
<code>__builtin_sysreg_bit_tgl_nop()</code>	Toggles all the bits of the nominated system register that are set in the supplied bit mask and places a NOP ; statement after the instruction.

Linker

Of the new linker features and enhancements, the most notable are:

- [“Expert Linker” on page 2-17](#)
- [“Linker Description File” on page 2-18](#)
- [“Data Elimination” on page 2-19](#)

For more information, refer to the *VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs* and online Help.

Expert Linker

The Expert Linker provides a GUI-based means of supplying the link commands currently supplied to the linker in a Linker Description File (.LDF). The tool also provides graphical and hypertext representations of text output for cross-reference, the linker map, and ELFDUMP.

Expert Linker enables you to visualize the commands in an .LDF file, make changes by manipulating graphical representations of the .LDF file, and generate an .LDF file. Any .LDF file that the Expert Linker supports can be read, modified, and written out to ensure support from both the linker and the Expert Linker. Key features include:

- Ability to display the memory layout (memory maps of supported parts) in the **Expert Linker** window and change the memory description. Use the **Memory Map** pane’s right-click menu to access various commands.
- Ability to drag and drop (add, move) sections and object files.
- Representation of object files after “pre-linking.” A determination of the sections appears in the link after you eliminate unused functions and objects.

Linker

Zoom functions that provide greater detail of executables and objects. You can zoom down to symbols and lengths.

For more information on the Expert Linker, refer to the corresponding chapter of the *VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs* and online Help.

Linker Description File

Internal Memory Segment Width

VisualDSP++ 3.0 linker rejects memory segments of `WIDTH(40)` in internal SHARC DSPs memory.

The linker forces the `WIDTH()` specifier in a memory segment definition to match the physical width of the memory segment. In previous releases, users were allowed to define a `DM` or `PM` section with `WIDTH(40)`, intended to hold extended-precision floating-point data. The DSP always had 48 bits available for each of these sections. In VisualDSP++ 3.0, the `WIDTH()` of such sections should be changed to 48 (the physical width of the memory segment) in order for the segment definitions to work as they have previously.

The linker does accept the 40-bit width in external memory space.

Block 1 Alias Address Space (ADSP-216x DSPs)

The linker rejects memory segments in the `Block 1` alias address space in the Linker Description Files for ADSP-21061 and ADSP-21062 DSPs. LDFs using the `Block 1` alias space must be changed to define only the physically available memory.

Data Elimination

The style of assembly code generated by the compiler was modified to provide the linker with greater scope for eliminating unneeded data and code. This modification does not affect the efficiency or size of the generated code.

ADSP-21161 DSP Loader

The loader program for ADSP-21161 DSPs has been modified and may not be compatible with the 2.0 loader.

The **Loader** page of the **Project Options** dialog box has been altered as follows.

- The **Mem21k** option button has been added to the page. By default, the loader (the **Loader** option button) is used to process an input file (.DXE). If the **Mem21k** is selected, the loader is not being invoked, and the mem21k program is launched to exclusively process the input file.
- For ADSP-21020 DSPs, the only permitted boot mode is JTAG; -bJTAG is automatically entered in the **Additional Options** box.
- The page uses consolidated kernels for VisualDSP++ 3.0 projects. Any VisualDSP++ 2.0 projects that utilized the default kernels and have been brought over to 3.0, automatically switch any 2.0 kernels to the 3.0 default kernels. Users do not have to switch the kernels manually.
- The loader of the current release obtains external memory width information from the input .DXE files. Consequently, the option buttons for the external memory width selections have been removed from the property page.
- The entries in the **Multiprocessor** list box are now editable.

Additional changes to the loader are as follows.

- The loader now processes true 64-bit sections from an input file.
- The loader processes external 40-bit sections from an input file, in addition to 48-and 32-bit sections of the previous release.

- The loader issues a warning message when it encounters a packed section from an input file.

Memory Initialization Utility

The `mem21k` utility has been updated to support `DATA64` sections.

Splitter

The splitter of the current release has been updated to support `DATA64` sections. The splitter treats `DATA64` as an independent data type, in addition to `PM` and `DM`.

The **Splitter** page of the **Project Options** dialog box has been updated to reflect the splitter changes. Selecting the new **64-bit memory** option button extends output data width and outputs file numbers to support `DATA64` sections. Equally, this option can be set on the splitter's command line with `-64`.

Refer to the *VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs* and online Help for more information about the splitter.

VisualDSP++ Component Software Engineering (VCSE)

VCSE is a combination of tools and guidelines that simplify the process of developing reusable 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 assembly, 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
- Encourage third party developers to provide the implementation of algorithms as easily used components

VCSE supports an Interface Definition Language (IDL) and a VIDL 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 that a component supports and generates the framework code needed to implement the component. The developer of the component can then concentrate on providing the implementation of the methods that the component is to provide. In addition, the VIDL compiler can generate a simple test harness to help in testing the component.

Integration with VisualDSP++

Integration of VCSE with VisualDSP++ simplifies the process of creating components and of incorporating and utilizing components from a variety of developers.

VCSE menu options are accessed from the **Tools** menu and enable you to:

- Use a wizard to create a new interface specification
- Use a wizard to create a VCSE component and a project to build the component
- Create a project to support the creation of a VCSE component
- Install and view components on the local system
- Download and install new or updated components from the ADI web site
- Add an installed component to a project

VisualDSP++ Kernel (VDK)

The following VisualDSP++ Kernel features have been added to 3.0.

- Inter-thread messaging
- A block-based memory manager
- Dynamic creation of semaphores and device flags
- Counting (taking any value within a specified range) semaphores
- Support for measuring thread stack usage
- Local storage for threads
- Sizeable reductions in the costs of many common API functions and a significant reduction in interrupt latency
- Kernel page support for the new functionality

You can use VDK messages to:

- Communicate information between threads
- Control access to a shared resource
- Signal the occurrence of an event and communicate information about the event
- Allow two threads to synchronize

The maximum number of supported messages can be configured separately for each project, and the messages owned by each thread can be viewed in the **VDK Status** window.

The memory pool manager provides support for deterministic and highly efficient memory allocation for one or more memory pools. Each memory pool contains memory blocks of a single size, but multiple pools can be defined, each with a different block size.

Device driver support has been redesigned and device drivers are now constructed as input/output objects. See the *VisualDSP++ 3.0 Kernel (VDK) User's Guide* and online Help for instructions on how to convert device drivers of the previous release for use in Release 3.0 projects.

Enhanced VDK Status Window

The **VDK Status** window's tree view has been extended to display values for:

- Additional thread items (last error, stack, and message queue)
- Semaphores
- Events
- Event words
- Device flags
- Memory pools

Enhanced State History Window

The **VDK State History** window displays:

- Log events, including user defined events
- User-configurable colors, patterns, and fonts

Each thread state can have a color and pattern. Each event can have a color but not a pattern. Font selection applies to all text in the display. Selecting

Object Protection

Use **as default** loads the custom settings each time the **VDK State History** window is created.

Object Protection

DSP tools now support the encryption and decryption of object files. This feature enables algorithm providers to distribute objects and libraries under license protection.

VisualDSP++ Component Software Engineering (VCSE)

VCSE consists of a combination of tools and guidelines that simplify the process of developing reusable 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 assembly, 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
- Encourage third party developers to provide the implementation of algorithms as easily used components

VCSE supports an Interface Definition Language (IDL) and a VIDL 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 that a component supports and generates the framework code needed to implement the component. The developer of the component can then concentrate on providing the implementation of the methods that the component is to provide. In addition, the VIDL compiler can generate a simple test harness to help in testing the component.

Integration with VisualDSP++

Integration of VCSE with VisualDSP++ simplifies the process of creating components and of incorporating and utilizing components from a variety of developers.

VCSE menu options are accessed from the **Tools** menu and enable you to:

- Use a wizard to create an new interface specification
- Use a wizard to create a VCSE component and a project to build the component
- Create a project to support the creation of a VCSE component
- Install and view components on the local system
- Download and install new or updated components from the ADI web site
- Add an installed component to a project

Documentation

This section describes the changes to online Help and online manuals.

Online Help

New VisualDSP++ online Help features include error message numbering, a unified index for all manuals in the documentation set, and the ability to search for keywords across the documentation set.

Error Message Numbering and Help

Tools that do batch processing can produce a list of diagnostic error messages when returning a result. Error reporting has been enhanced as follows.

- Every error message is uniquely identified with a number that is consistent from release to release. Consistent numbering enables you to become familiar with the meaning of an error.
- Tool identifiers prepended to error numbers identify the tool involved when an error is reported to customer support. The tool prefixes are defined as follows.

ar0001	archiver
cc0001	compiler
ea0001	assembler
el0001	Expert Linker
ld0001	loader
li0001	linker
pp0001	assembler/linker preprocessor
vc0001	VIDL compiler

Documentation

- Every diagnostic error is documented. Detailed information about an error includes the severity level, condition that causes the error and remedy, examples, and related information.
- Error messages are output in a format that can be parsed by the IDDE. Where applicable, they include file name and line number information. Error messages that do not refer to a particular source location do not include the file name and line number information.
- Every tool uses the same hierarchy of error message severity, described as follows from the highest to the lowest level of severity.

fatal	Identifies an error so severe that further processing of the input is suspended. The tool reports a failure.
error	Identifies a problem that causes the tool to report a failure. Further processing of the input might be allowed to continue to enable additional problems to be reported.
warning	Identifies a situation that does not prevent the tool from processing the input, but may indicate potential problems.
remark	Provides information of possible interest.

- You can suppress the reporting of warnings and remarks, either entirely or individually.
- You can override the severity of compiler discretionary diagnostic messages. A discretionary message is marked by -D and is an error, remark, or warning whose severity can be promoted, demoted, or suppressed. For example, you can promote a compiler remark or warning to an error. You can demote an compiler error to a warning or remark.

To view an explanation of the error message, select with the mouse or keyboard the six-character error identifier (for example, cc0251) in the **Output** window's **Build** page. Press the F1 key. Error message details appear in the Help window.

Unified Index and Search

Index entries from all the VisualDSP++ 3.0 manuals have been merged into one unified index, which you can access by clicking the **Index** tab in the online Help. You can use the **Search** function in the online Help to search for keywords and instructions across all VisualDSP++ 3.0 publications for the target processor family. The Help window lists each occurrence and its location (manual).

Online Manuals

Some manuals in the VisualDSP++ 3.0 documentation set have been converted to HTML Help and merged to facilitate searches in the online Help. Each book is still available in PDF format.

The **Docs** directory of your VisualDSP++3.0 for SHARC DSPs installation CD contains a complete set of documentation for Digital Signal Processors that are supported by your development tools suite. The set is comprised of VisualDSP++ tools and EZ-KIT Lite manuals, hardware manuals, and data sheets placed in the appropriate folders:

- **Datasheets** directory contains one .PDF file for each data sheet.
- **Tools Manuals** and **EZ-KIT Lite Manuals** directories contains one .PDF file for each manual.
- **Hardware Manuals** directory contains one .PDF file for each chapter in each manual.

3 DIFFERENCES BETWEEN RELEASES

This chapter describes the differences between VisualDSP++ 3.0 and VisualDSP++ 2.0. Read this chapter if upgrading from the previous software release.

Existing project files (.DPJ) can be imported into the new release. However, once the project file is imported, you are not able to bring the project back in 2.0. Similarly, new projects created using VisualDSP++ 3.0 cannot be used by earlier versions of the tools.

Of the new and changed features, the following have the most impact on your existing project files:

- [“Assembler and Preprocessor Differences” on page 3-2](#)
- [“Compiler and Library Differences” on page 3-6](#)
- [“Linker Differences” on page 3-14](#)
- [“Loader Differences” on page 3-16](#)
- [“Splitter Differences” on page 3-17](#)
- [“New File Formats” on page 3-17](#)
- [“VDK Differences” on page 3-18](#)

You may want to consult the cover letter that accompanies the product installation CD for the last-minute information concerning this release.

Assembler and Preprocessor Differences

VisualDSP++ 3.0 assembler for SHARC DSPs has some differences you may encounter when upgrading from 2.0. There are some differences to the preprocessor as well.

The changes that may impact your existing assembly projects are described as follows.

- [“Command Line Switches” on page 3-2](#)
- [“Directives” on page 3-3](#)
- [“Functions and Operators” on page 3-4](#)
- [“Macros” on page 3-4](#)

Command Line Switches

The assembler introduces the following command-line switches ([Table 3-1](#)).

Table 3-1. New Assembler Switches

Switch	Description
<code>-flags-compiler</code>	Passes the specified option or a list of options to the compiler. Used when compiling <code>.IMPORT C</code> header files.
<code>-flags-pp</code>	Passes the specified option or a list of options to the preprocessor.
<code>-proc ADSP-ID</code>	Specifies the ADSP-21xxx DSP for which the assembler should produce suitable code. Assembling with this switch defines macros listed in “Macros” on page 3-4 .
<code>-Wnumber</code>	Selectively disables warnings specified by one or more message numbers.

The command-line switches listed in [Table 3-2](#) no longer appear on the assembler's `-help` screen.

Table 3-2. Removed Assembler Switches

Switch	Change for VisualDSP++ 3.0
<code>-21[020 060 061 062 065 160 161]</code>	In VisualDSP++ 2.0, specifies the ADSP-21xxx DSP. Now the preferred way to specify a target DSP architecture is via <code>-proc</code> .

Directives

The following assembly directives are new to VisualDSP++ 3.0 ([Table 3-3](#)). Note that these directives are reserved keywords.

Table 3-3. New Assembly Directives

Directive	Description
<code>.ENDIF</code>	Ends a conditional assembly block.
<code>.ELIF</code>	Adds an alternative condition to the <code>.IF/.ENDIF</code> conditional block.
<code>.ELSE</code>	Adds an alternative condition to the <code>.IF/.ENDIF</code> conditional block.
<code>.EXTERN STRUCT</code>	References a global symbol defined in another file.
<code>.IF</code>	Begins a conditional assembly block.
<code>.IMPORT</code>	Provides the assembler with the structure layout (C struct) information.
<code>.STRUCT</code>	Defines and initializes data objects based on C typedefs from <code>.IMPORT C</code> header files.

Functions and Operators

Table 3-4 briefly describes the new assembly built-in functions and operators. For more information about the builtins, see the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs* and online Help.

Table 3-4. New Functions and Operators

Operator	Description
<code>offsetof()</code>	Calculates the offset of a specified member from the beginning of its parent data structure in units that are appropriate to the target architecture. For ADSP-21xxx DSPs, units are in words.
<code>sizeof()</code>	Returns the amount of storage associated with an imported C <code>struct</code> or data member.
<code>...</code>	Specifies a variable-length argument list in macro definition statements.
<code>-></code>	References a struct's data members, as in C.

Macros

The following macros (Table 3-5) are defined by the assembler for the architecture and language being processed.

Table 3-5. Assembler Feature Macros

Macro Definition	Description
<code>-D__ADSP21000__ =1</code>	Always present.
<code>-D__ADSP21020__ =1</code> <code>-D__2102X__ =1</code>	Present when processing with the <code>-proc ADSP-21020</code> switch.
<code>-D__ADSP21060__ =1</code> <code>-D__2106X__ =1</code>	Present when processing with the <code>-proc ADSP-21060</code> switch.

Table 3-5. Assembler Feature Macros (Cont'd)

Macro Definition	Description
-D__ADSP21061__ =1 -D__2106X__ =1	Present when processing with the -proc ADSP-21061 switch.
-D__ADSP21062__ =1 -D__2106X__ =1	Present when processing with the -proc ADSP-21062 switch.
-D__ADSP21065__ =1 -D__2106X__ =1	Present when processing with the -proc ADSP-21065 switch.
-D__ADSP21160__ =1 -D__21160__ =1	Present when processing with the -proc ADSP-21160 switch.
-D__ADSP21161__ =1 -D__21161__ =1	Present when processing with the -proc ADSP-21161 switch.
-D_LANGUAGE_ASM =1	Always present.
-D_LANGUAGE_C =1	Present when compiling .IMPORT C headers. Replaces _LANGUAGE_ASM.

Compiler and Library Differences

The compiler differences between VisualDSP++ 3.0 and 2.0 are described as follows.

- [“Command Line Switches” on page 3-6](#)
- [“Input and Output Files” on page 3-9](#)
- [“Pragmas” on page 3-9](#)
- [“Library Functions” on page 3-11](#)

Command Line Switches

VisualDSP++ 3.0 C/C++ compiler for SHARC DSPs includes some command line switches that are new or whose functionality has been changed since 2.0. These C mode switches are summarized in [Table 3-6](#).

Table 3-6. New C Compiler Switches

Switch	Description
-debug-types	Supports building a *.h file directly and writing a complete set of debugging information for the header file.
-dollar	Accepts dollar signs in identifiers.
-expert-linker	Provides the initial Expert Linker link line.
-float_to_int	Use a support library function to convert a float to an integer.
-fp-associative	Treats floating-point multiply and addition as an associative.
-I-	Establishes the point in the include directory list at which the search for the header files enclosed in angle brackets should begin.
-line-debug	Instructs the compiler to generate limited debugging information.

Table 3-6. New C Compiler Switches (Cont'd)

Switch	Description
<code>-Mt filename</code>	Specifies an alternative output file name for the make dependency target.
<code>-MQ</code>	Generates make rules only; does not compile. No notification is given when input files are missing.
<code>-no-dir-warnings</code>	Disables warnings about non-existent <code>include</code> and library directories.
<code>-no-dollar</code>	Does not accept dollar signs in symbols.
<code>-no-fp-associative</code>	Does not treat floating-point multiply and addition as an associative.
<code>-no-simd</code>	Disables automatic SIMD mode when compiling for ADSP-2116x DSPs.
<code>-nothreads</code>	Specifies that all compiled code need not be thread-safe.
<code>-proc ADSP-ID</code>	Specifies a processor for which the compiler should produce suitable code. The preferred way to specify the processor.
<code>-R</code>	Removes all directories from the standard search path for source files.
<code>-switch-pm</code>	Specifies that the compiler should place switch tables in program memory.
<code>-sysdefs</code>	Defines the system definition macros listed on page 3-4 .
<code>-val-global name-list</code>	Specifies that the names given by <code>name-list</code> are present in all globally-defined variables.
<code>-writes-opts</code>	Passes the user options (but not input file names) via a temporary file.

Compiler and Library Differences

In addition, the compiler provides methods to avoid hardware anomalies that affect ADSP-21160 DSPs ([Table 3-7](#)).

Table 3-7. ADSP-21160 DSP Compiler Switches

Switch	Description
-21160rev0	Ensures that the code compiled with this option will avoid the RFRAME anomaly.
-swf_screening	Ensures that the code compiled with this option will avoid the FIFO anomaly.

The C/C++ compiler of the current release does not support all the command-line switches presented in VisualDSP++ 2.0. The removed switches are listed in [Table 3-8](#).

Table 3-8. Obsolete C/C++ Compiler Switches

Switch	Description
-constant-read-write	Specifies that const data may be modified elsewhere.
-signed-char	Makes the char data type signed.
-unsigned-char	Makes the char data type unsigned.
-instant[all used]	Instantiates all or used members of a class.

Input and Output Files

The compiler now can process Interface Definition Language, template instantiation, memory map, and symbol map files, as described in [Table 3-9](#).

Table 3-9. New Input/Output Files

File Extension	Description
.IDL	Interface Definition Language specifications for VCSE components and interfaces.
.II	Template instantiation files; used internally by the compiler when instantiating C++ templates.
.MAP	DSP system memory map file output.
.SYM	DSP system symbol map file output.

Pragmas

[Table 3-10](#) through [Table 3-14](#) list VisualDSP++ 3.0 pragmas for data alignment, interrupt handling, SIMD loop processing, optimization, and linkage.

Table 3-10. Data Alignment Pragmas

Pragma	Description
<code>align number</code>	Applies to the declaration that immediately follows the directive. Aligns the symbol on the specified boundary.

Compiler and Library Differences

Table 3-10. Data Alignment Pragma (Cont'd)

Pragma	Description
<code>pack(alignopt)</code>	Applies to the <code>struct</code> that immediately follows the pragma. Reduces the structure's size by changing the data alignment.
<code>pad(alignopt)</code>	Applies to the <code>struct</code> that immediately follows the pragma. Increases the structure's alignment to improve the compiler's opportunities in vectorizing access to the data.

Table 3-11. Interrupt Handler Pragma

Pragma	Description
<code>interrupt</code>	Applies to the function declaration or definition that immediately follows the pragma. The generated function code can be used as a self-dispatching interrupt handler.

Table 3-12. ADSP-2116x DSPs SIMD Pragma

Pragma	Description
<code>SIMD_for</code>	Applies to the loop instructions that immediately follow the pragma. The instructions are processed in SIMD mode if the loop adheres to SIMD standards.

Table 3-13. General Optimization Pragma

Pragma	Description
<code>optimize_off</code>	Turns off the optimizer if it is enabled.
<code>optimize_for_space</code> <code>optimize_for_speed</code>	Turn the optimizer back on if it is disabled, or switch focus between reducing code size and increasing performance if the optimizer has been already enabled.

Table 3-14. Linking Pragmas

Pragma	Description
<code>linkage_name</code>	Ensures that the named variable or function declaration is used as the external reference.
<code>retain_name</code>	Indicates that the named external function is not to be removed even if the function is not used.

Library Functions

The libraries provided with the current release of the ADSP-21xxx DSPs compiler have a number of new and enhanced functions, as described in [Table 3-15](#) through [Table 3-18](#). The library documentation has been enhanced as well.

Table 3-15. New C/C++ Run-Time Library Functions

Function	Description
<code>circindex</code>	Performs a circular buffer operation on loop index.
<code>circptr</code>	Performs a circular buffer operation on a pointer.
<code>heap_lookup_name</code>	Obtains a primary heap identifier. Replaces the <code>heap_lookup</code> function of the previous release.
<code>isinf</code>	Tests for infinity.
<code>isnan</code>	Tests for a not-a-number (NaN).
<code>set_alloc_type</code>	Sets a heap for dynamic memory allocation.

Compiler and Library Differences

Table 3-16. New ADSP-2106x DSP Library Functions

Function	Description
cvecdot	Complex vector dot product.
cvecsadd	Complex vector scalar addition.
cvecsm1t	Complex vector scalar multiply.
cvecssub	Complex vector scalar subtraction.
cvecvadd	Complex vector addition.
cvecvm1t	Complex vector multiply.
cvecvsub	Complex vector subtraction.
matinv	Real matrix inversion.
transpm	Real matrix transpose.
vecdot	Vector dot product.
vecsadd	Vector scalar addition.
vecsm1t	Vector scalar multiply.
vecssub	Vector scalar subtraction.
vecvadd	Vector addition.
vecvm1t	Vector multiply.
vecvsub	Vector subtraction.

Table 3-17. New ADSP-2116x DSP Library Functions

Function	Description
biquad	Biquad filter section.
cfft	Fast N-point complex input FFT.
matinv	Real matrix inversion.
transpm	Real matrix transpose.
twidfft	Generates twiddle factors for the fast FFT function <code>cfft</code> .

Table 3-18. Obsolete C/C++ Run-Time Library Functions

Function	Description
heap_init	Initializes a heap.
heap_switch	Sets a heap for dynamic memory allocation.
heap_lookup	Obtains a primary heap identifier. In 3.0, is replaced with the <code>heap_lookup_name</code> function.

Linker Differences

VisualDSP++ 3.0 linker, Linker Description File, and utilities include the following changes:

- [“Command Line Switches” on page 3-14](#)
- [“LDF and Preprocessor Macros” on page 3-15](#)
- [“LDF Commands” on page 3-15](#)

Command Line Switches

The command-line switches listed in [Table 3-19](#) and [Table 3-20](#) have been introduced or functionally changed since the previous release.

Table 3-19. New Linker Switches

Switch	Description
<code>-od <i>fileName</i></code>	Specifies the output directory. Sets the value of the <code>\$COMMAND_LINE_OUTPUT_DIRECTORY</code> macro to 1.
<code>-verbose</code>	Same as <code>-v</code> (outputs status information while linking).

Table 3-20. Legacy Linker Switch

Switch	Description
<code>-DADSP-<i>ID</i></code>	Used to specify the target SHARC DSP for which the linker should produce suitable code. The valid IDs are: ADSP-21020, ADSP-21060, ADSP-21061, ADSP-21062, ADSP-21065, ADSP-21160, and ADSP-21161. In 3.0, the preferred way to specify a target architecture is via the <code>-proc</code> switch.

LDF and Preprocessor Macros

The following macros have introduced in 3.0:

- The `$COMMAND_LINE_OUTPUT_DIRECTORY` built-in LDF macro expands into the path of the output directory, which is set with the linker's `-od` switch (or `-o` switch when `-od` is not present).
- The `__EZKIT_LICENSE_RESTRICTION_SHARC__` preprocessor macro has been introduced for use in LDF files. Running the linker with an EZ-KIT Lite license defines the macro as 1.

LDF Commands

We recommend you to revise any LDF files which include the `OVERLAY_GROUP{ }` legacy LDF command.

In VisualDSP++ 2.0, the `OVERLAY_GROUP{ }` is used to group overlays, allowing each overlay to run from a different start address in run-time memory. The linker of the current release still processes all overlay groups and explicit `OVERLAY_GROUP{ }` commands, producing a warning. However, the preferable way to manage overlays is to create a separate output section for each overlay group.

Loader Differences

Command-Line Switches

VisualDSP++ 3.0 loader for ADSP-2106x/ADSP-2116x DSPs offers the following new command-line switches ([Table 3-21](#)).

Table 3-21. New Loader Switches

Switch	Description
-bJTAG	Specifies the JTAG boot mode. For ADSP-21020 DSPs; this is the only permitted boot mode.
-NoKernel	For ADSP-2106x/ADSP-21160 DSPs, builds an .LDR file that does not include a boot-kernel.

The ADSP-21161 DSP loader of the current release does not support all the command-line switches presented in VisualDSP++ 2.0. The removed switch is listed in [Table 3-22](#).

Table 3-22. Obsolete ADSP-21161 DSP Loader Switch

Switch	Description
-ExternalMemoryWidth #	In 2.0, specifies an external memory width. In 3.0, the loader obtains the width information from the input .DXE files. The External memory instruction width option buttons have been removed from the loader's property page as well.

Refer to the *VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs* and online Help for more information about the loaders.

Splitter Differences

VisualDSP++ 3.0 splitter for ADSP-2116x DSPs offers the following new command-line switch ([Table 3-23](#)).

Table 3-23. New ADSP-216x/ADSP-21160 DSP Splitter Switch

Switch	Description
-64	Directs the splitter to extract all segments declared as 64-bit memory segments from a .DXX file. This switch influences the operation of the -ram and -norom switches, adding DATA64 memory as their target.

See the *VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs* and online Help for more information about the splitter.

New File Formats

[Table 3-24](#) lists the newly added file formats.

Table 3-24. New Input/Output Files

File Extension	Description
.BNA	IDMA host boot file format.
.IDM	ASCII-byte stream format.

For more information, see “File Formats” in the *VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs* and online Help.

VDK Differences

The differences between VDK of VisualDSP++ 3.0 and 2.0 are as follows.

- Device driver architecture has fundamentally changed. Device drivers have become part of the I/O interface, and are, consequently, class based.

While the underlying changes to device drivers have a minimal affect on the use of a device driver, the implementation of the device driver has changed significantly. Therefore, device drivers created under VisualDSP++ 2.0 are incompatible with the device driver model in VisualDSP++ 3.0. Additionally, device flags are no longer associated with an individual device driver but are instead created as global objects.

The *VisualDSP++ 3.0 Kernel (VDK) User's Guide* contains an appendix describing how to convert device drivers created under VisualDSP++ 2.0 for use in projects built with 3.0.

- In VisualDSP++ 2.0, all semaphores are binary semaphores created at boot time. In VisualDSP++ 3.0, all semaphores are counting semaphores that can be dynamically created.
- VDK state history is displayed within the VisualDSP++ environment if the `vdk.cpp` file is compiled with debugging information, even in `Release` configuration. There is no negative impact on runtime performance since this source file contains only data declarations.