

VISUALDSP⁺⁺™ 3.0

Product Bulletin for ADSP-21xx DSPs

Revision 1.0, July 2002

Part Number
82-000349-05

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



Copyright Information

© 2002 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	vii
Intended Audience	vii
Manual Contents	viii
Technical or Customer Support	ix
Supported Processors	ix
Product Information	x
MyAnalog.com	x
DSP Product Information	xi
Related Documents	xi
Online Technical Documentation	xii
From VisualDSP++	xiii
From Windows Explorer	xiii
From the Web	xiii
Printed Manuals	xiv
VisualDSP++ Documentation Set	xiv
Hardware Manuals	xiv
Datasheets	xiv

CONTENTS

Contacting DSP Publications	xv
Notation Conventions	xv

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

Integrated Development and Debugging Environment (IDDE)	2-2
External Makefile Execution	2-2
New Profiler Interface	2-3
Linear Profiling	2-3
Statistical Profiling	2-4
New Streams Interface	2-4
Tcl Commands	2-5
Image Viewer	2-5
Flash Programmer	2-7
Editor Enhancements	2-7
Simulator	2-8
Pipeline Viewer	2-8
Assemblers	2-10
Code Sequence Analysis	2-10

C Headers and Structures Import	2-11
.IMPORT Directive	2-11
Search Option	2-11
Macros	2-12
C Structures Support	2-12
.STRUCT Directive	2-12
.EXTERN STRUCT Directive	2-12
Built-in Functions	2-13
Conditional Assembly	2-13
Compiler and Library	2-14
Pragmas	2-14
Data Alignment Pragma	2-15
Interrupt Handler Pragma	2-15
Altregisters Pragma	2-15
Optimization Level Pragmas	2-16
Linkage Pragmas	2-16
Stack Usage Pragma	2-16
ETSI Support	2-17
Preprocessing of VIDL Files	2-17
Linker	2-18
Expert Linker	2-18
Data Elimination	2-19
New Default LDF Behavior	2-19

CONTENTS

Documentation	2-20
Online Help	2-20
Error Message Numbering and Help	2-20
Unified Index and Search	2-22
Online Manuals	2-22
VisualDSP++ Component Software Engineering (VCSE)	2-23
Integration with VisualDSP++	2-24
VisualDSP++ Kernel	2-25
Enhanced VDK Status Window	2-26
Enhanced State History Window	2-26
Object Protection	2-27

DIFFERENCES BETWEEN RELEASES

Assembler and Preprocessor Differences	3-2
Command Line Switches	3-2
Directives	3-3
Built-in Functions and Operators	3-4
Predefined Macros	3-5
Compiler and Library Differences	3-6
Command Line Switches	3-6
Input and Output Files	3-8
Pragmas	3-9
Library Functions	3-11

Linker Differences	3-12
Command Line Switches	3-12
LDF Operators	3-13
Builtin LDF Macros	3-13
LDF Commands	3-13
ADSP-219x DSP Loader/Splitter	3-14
ADSP-2192-12 DSP Loader/Splitter	3-15
File Formats	3-15
VDK Differences	3-16

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++ Release 3.0 for ADSP-218x and ADSP-219x 16-bit, fixed-point DSPs. It also describes the differences between Release 3.0 and Release 2.0.

For details, refer to the VisualDSP++ 3.0 manuals for ADSP-21xx DSPs and online Help.

Intended Audience

This publication is primarily intended for DSP programmers who are upgrading from the previous release of VisualDSP++ 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), simulator, assembler, compiler, linker, loader, and documentation. Also describes the new VisualDSP++ Component Software Engineering (VCSE) and the changes to the VisualDSP++ Kernel (VDK), which are the key features to Release 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 “*ADSP-21xx*” refers to two families of Analog Devices 16-bit, fixed-point digital signal processors. VisualDSP++ 3.0 for ADSP-21xx DSPs currently supports the following processors.

- ADSP-218x family DSPs: ADSP-2181, ADSP-2183, ADSP-2184/84L/84N, ADSP-2185/85L/85M/85N, ADSP-2186/86L/86M/86N, ADSP-2187L/87N, ADSP-2188L/88N, and ADSP-2189M/89N

- ADSP-219x family DSPs: ADSP-2191, ADSP-2192-12, ADSP-2195, ADSP-2196, ADSP-21990, ADSP-21991, and ADSP-21992

Product Information

You can obtain product information from the Analog Devices website, from the product installation 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 76903-557 (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 ADSP-21xx DSPs

VisualDSP++ 3.0 Assembler and Preprocessor Manual for ADSP-218x and ADSP-219x DSPs

VisualDSP++ 3.0 C Compiler and Library Manual for ADSP-218x DSPs

VisualDSP++ 3.0 C/C++ Compiler and Library Manual for ADSP-219x DSPs

VisualDSP++ 3.0 Linker and Utilities Manual for ADSP-218x and ADSP-219x DSPs

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

PREFACE

VisualDSP++ 3.0 Component Software Engineering User's Guide

VisualDSP++ 3.0 User's Guide for ADSP-21xx DSPs

Quick Installation Reference Card

Online Technical Documentation

Online documentation comprises VisualDSP++ Help system and tools manuals, Dinkum Abridged C++ library and 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 are also provided.

A description of each documentation file type is as follows.

File	Description
.CHM	Help system files and VisualDSP++ tools manuals.
.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++ 3.0 CD-ROM at any time by running the installation again.

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

From VisualDSP++

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

From Windows Explorer

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

- Double-click any file that is part of the VisualDSP++ documentation set.
- Double-click the `vdsp-help.chm` file, which is 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.

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

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 <http://www.analog.com/salesdir/continent.asp>.

Hardware Manuals

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.

Datasheets

All datasheets can be downloaded from the Analog Devices website. As a general rule, any datasheet with a letter suffix (L, M, N) can be obtained from the Literature Center at **1-800-ANALOGD (1-800-262-5643)** or downloaded from the website. Datasheets without the suffix can be downloaded from the website only—no hard copies are available. You can ask for the datasheet by a part name or by product number.

If you want to have a datasheet faxed to you, the phone number for that service is **1-800-446-6212**. Follow the prompts and a list of datasheet code numbers will be faxed to you. Call the Literature Center first to find out if requested datasheets are available.

Contacting DSP Publications

Please send your comments and recommendation 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)	Text in bold style indicates 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 required items in syntax descriptions appear within curly brackets and 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, and feature names are in text with <i>letter gothic font</i> .
<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 and the requirements for running its latest revision, VisualDSP++ 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++ 3.0 is Analog Devices project management and development environment for Digital Signal Processor (DSP) application development. The environment successfully integrates the graphical user interface and code generation tool chain, enabling programmers to move easily between editing, building, debugging, and deployment of final products.

The code generation chain is comprised of the processor-specific development tools: simulator, assembler, C/C++ compilers and libraries, linker, loaders, splitter, VisualDSP++ Component Software Engineering (VCSE), VisualDSP++ Kernel (VDK), and utilities necessary for developing DSP applications.

The product installation CD-ROM also includes an evaluation suite of the EZ-KIT Lite™ software, which provides programmers with 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, 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 4.0SP3/2000/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 16-bit, fixed-point processors.

- ADSP-2181, ADSP-2183, ADSP-2184/84L/84N, ADSP-2185/85L/85M/85N, ADSP-2186/86L/86M/86N, ADSP-2187L/84L/87N, ADSP-2188L/88N, and ADSP-2189M/89N
- ADSP-2191, ADSP-2192-12, ADSP-2195 (new), ADSP-2196 (new), ADSP-21990 (new), ADSP-21991 (new), and ADSP-21992 (new)

Benefits Provided by VisualDSP++ 3.0

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

- **Support for new architectures.** VisualDSP++ 3.0 includes support for new 16-bit, fixed-point processors: ADSP-2195 and ADSP-2196 as well as for ADSP-21990, ADSP-21991, and ADSP-21992, mixed-signal DSP controllers.
- **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.

Benefits Provided by VisualDSP++ 3.0

- **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.
- **Multiple language support.** Debug programs written in C, C++ (ADSP-219x DSPs only), 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.
- **Effective debug control.** Set breakpoints on 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, profile, 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** (ADSP-219x DSPs only). 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 cycle-accurate debugging of complex systems. You can arrange processors into logical groups for better control of system's behavior.
- **Code reusability.** Programmers often begin a new project by writing infrastructure portions that transfer data between algorithms. The VisualDSP++ Kernel (VDK) 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 (VCSE) 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 and use components without becoming familiar with the detail of the model and mechanism involved. VisualDSP++ 3.0 also provides wizards for creating interfaces and components.

Benefits Provided by VisualDSP++ 3.0

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 environment, tools, and documentation.

The information is presented as follows.

- “Integrated Development and Debugging Environment (IDDE)” on page 2-2
- “Simulator” on page 2-8
- “Assemblers” on page 2-10
- “Compiler and Library” on page 2-14
- “Linker” on page 2-18
- “Documentation” on page 2-20
- “VisualDSP++ Component Software Engineering (VCSE)” on page 2-23
- “VisualDSP++ Kernel” on page 2-25
- “Object Protection” on page 2-27

Integrated Development and Debugging Environment (IDDE)

The IDDE provides these new features and enhancements:

- “External Makefile Execution” on page 2-2
- “New Profiler Interface” on page 2-3
- “New Streams Interface” on page 2-4
- “Image Viewer” on page 2-5
- “Flash Programmer” on page 2-7
- “Editor Enhancements” on page 2-7

For more information, refer to the *VisualDSP++ 3.0 User's Guide for ADSP-21xx DSPs* and online Help.

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 in the editor window. A syntax highlighting file 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 2-20](#). 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 supported.

Linear Profiling

A new linear profiling interface is available for the ADSP-219x DSP simulator only. Emulators continue to use the statistical profiling interface. A new menu option (**New Profile**) under **Tools** and **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.

From the **Display** tab, select the profiling data items to display. The linear profiling interface enables the simulator to pass additional profiling data, such as cache hits and misses, which can be displayed in additional columns.

Integrated Development and Debugging Environment (IDDE)

From the **Filter** tab, choose one of the following items to profile.

- **Entire memory space.** The linear and statistical profilers have the same behavior as in VisualDSP++ 2.0.
- **C/C++ functions.** Choose functions to profile from a list of available functions. The profiler discards any collected PC samples that are not within the selected functions.
- **Memory ranges.** Specify the start and end addresses of the memory ranges to profile. The profiler discards any collected program counter (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 (emulator only) 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 the actual or simulated DSP target. VisualDSP++ 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* and online Help.

Image Viewer

An IDDE plug-in window allows to display image and video data or DSP memory stored in a PC image file. Supported file types include .BMP, .JPEG, .PPM, and .MPEG.

Integrated Development and Debugging Environment (IDDE)

The following dialog boxes control the Image Viewer settings.

- The **Configuration** dialog box enables the image source (memory or file) and image attributes to be defined. 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 address and attributes.
- The **Play Video** dialog box allows to play a video clip of MPEG data stored in DSP memory.

Flash Programmer

The VisualDSP++ Flash Programmer (under **Tools** and **Flash Programmer**) provides a consistent, generic interface to numerous DSPs and flash memory devices, allowing you to modify a flash device's memory.

The new utility provides a convenient user interface that simplifies the process of changing data values on a flash device. You no longer have to remove the flash memory from the board—use a separate Flash Programmer and then replace the flash.

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

Editor Enhancements

New features 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

Simulator

VisualDSP++ 3.0 simulator for ADSP-219x DSPs is a cycle accurate simulator with a six-stage pipeline. The simulator functionality models the behavior of the following.

- Instruction set
- Processor sequencer
- Memory hierarchy, including the external SRAM
- Registers, including the MMRs
- All the core peripherals
- All memory transactions

The processor state is updated after each instruction execution.

The correct execution count trails the execution of the instruction by at least the length of the sequencer pipeline and maybe more. When you break execution of the simulator, the cycle count may not be accurate. At the completion of the program, the cycle count is correct.

Pipeline Viewer

The **Pipeline Viewer** (ADSP-219x DSPs only), under **View** and **Debug Windows**, enables you to understand how the processor affects the execution timing of your program.

The **Pipeline Viewer** window displays the pipeline stages and cycle-by-cycle analysis of the instructions passing through the stages. The window also displays informational messages for instructions indicated with an event icon. These types of messages may appear:

- Stalls detected

New Features and Enhancements

- Kills detected
- Multicycle instruction messages

For more information, see the *VisualDSP++ 3.0 Getting Started Guide*, *VisualDSP++ 3.0 User's Guide*, and online Help.

Assemblers

VisualDSP++ Release 3.0 provides new assembler features:

- “Code Sequence Analysis” on page 2-10
- “C Headers and Structures Import” on page 2-11
- “C Structures Support” on page 2-12
- “Conditional Assembly” on page 2-13

For more information, refer to the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for ADSP-218x and ADSP-219x DSPs* and online Help.

Code Sequence Analysis

New command line switches `-stallcheck=cond` and `-stallcheck=all` help optimize assembly code by providing code sequence analysis and feedback on latencies and stalls. Feedback from the assembler consists of informational messages displayed in the **Output** window of VisualDSP++. You can review dynamic information about latencies and stalls as provided by a cycle accurate simulator.

A latency condition (use before availability) exists when a pair of instructions incur extra cycles between them because of their proximity to each other in the code. Such cases are reported when you use the default switch:

```
-stallcheck=cond
```

You can avoid a latency condition’s cycle loss by separating the two instructions with as many instructions as cycles lost.

Multicycle behavior exists when an instruction, sometimes only under certain circumstances, takes more than one cycle to complete. You cannot

avoid this cycle loss without removing the instruction that causes it. Such cases (in addition to conditional stalls) are reported when you use the following switch:

```
-stallcheck=all
```

C Headers and Structures Import

Now the assemblers can import C header files and access structures defined in C header files. VisualDSP++ 3.0 includes the following directive, command line switch, and macros to support this feature.

.IMPORT Directive

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

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 file, 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 the `struct` type—that requires the `.STRUCT` directive.

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.

Assemblers

Macros

For each `.IMPORT` header, the assembler calls the compiler driver with the appropriate processor 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 assemblers can call the compilers to process structure definitions provided by C header files. Release 3.0 includes the following directives, built-in functions, and macros that support this feature.

For more information about C structures support, refer to the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for ADSP-218x and ADSP-219x DSPs* and online Help.

.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 assemblers support 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-21xx 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 the units appropriate for its processor. For ADSP-21xx DSPs, the `sizeof()` units are in words.

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 ADSP-218x and ADSP-219x DSPs* and online Help for more information.

Compiler and Library

VisualDSP++ 3.0 ADSP-218x (C) and ADSP-219x (C/C++) DSP compilers provide advantages in performance over VisualDSP++ 2.0. The compilers generate efficient code that is optimized for both size and execution.

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

- [“Pragmas” on page 2-14](#)
- [“ETSI Support” on page 2-17](#)
- [“Preprocessing of VIDL Files” on page 2-17](#)

The compilers also include a number of processor-specific enhancements. For information on these enhancements, refer to the *VisualDSP++ 3.0 C Compiler and Library Manual* for your target DSP.

Pragmas

ADSP-21xx DSP compilers support a number of pragmas. Pragmas are implementation-specific directives that modify the compiler's behavior for:

- Data alignment
- Interrupt or exception handling
- General optimization
- External module linkage
- Stack usage (ADSP-218x DSPs only)

Data Alignment Pragma

The `align num` pragma is useful for declaring arrays that need to be on a circular boundary. Such arrays might be required to make use of a bit-reversal sorting algorithm implemented using the ADSP-219x DSP DAG1 bit-reversal mode. For ADSP-218x DSP architectures, the `align num` pragma is also useful for declaring arrays at correct base addresses. The latter can be passed to assembly functions and used as circular buffers or storage for autobuffering.

Interrupt Handler Pragma

The `interrupt` pragma applies to the function declaration or definition that immediately follows the pragma. The pragma causes the compiler to generate the function code that may be used as a self-dispatching interrupt handler.

An interrupt pragma function saves its full context, including all modes and registers required for the function's operation, and restore the context upon exit. The function returns using a return from interrupt (RTI) instruction.

Altregisters Pragma

The `altregisters` pragma may be used in conjunction with the `interrupt` pragma to indicate that the compiler can optimize the saving and restoring of registers through use of the secondary register sets. Note the use of the `altregisters` pragma is not safe when nested interrupts are enabled.

Optimization Level Pragmas

The `optimize_off`, `optimize_for_space`, and `optimize_for_speed` pragmas change the optimization level while a given module is being compiled. 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` and `weak` change how a given global function or variable is viewed during the linking stage.

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

The `weak_entry` pragma causes the compiler to generate the function definition with weak linkage. It applies to the function declaration or definition that immediately follows the pragma.

Stack Usage Pragma

This pragma applies to the ADSP-218x DSP architectures only. The `make_auto_static` pragma is associated with the function definition that immediately follows the pragma. It directs the compiler to place all automatic variables used in the function in static store. This may be beneficial because an access to a static store for the ADSP-218x DSPs is

done in one instruction, while an access to a local automatic stack may require three instructions.

ETSI Support

The ETSI (European Telecommunications Standards Institute) support for ADSP-218x and ADSP-219x processors is a collection of functions that provides high-performance implementations for operations commonly required by DSP applications. These operations provided by the ETSI library (`libetsi.dlb`) and compiler built-in functions (defined in `ETSI_fract_arith.h`) include support for fractional, or fixed-point, arithmetic. The results obtained from these operations have well defined overflow and saturation conditions. The ETSI support operations are Analog Devices extensions to ANSI C standard.

Refer to the *VisualDSP++ 3.0 C/C++ Compiler and Library Manual* for your target DSP for more information.

Preprocessing of VIDL Files

VisualDSP++ 3.0 compiler preprocessor supports the preprocessing of Interface Definition Language (VIDL) specifications for VCSE components and interfaces. Every specification (`.IDL`) 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.

Linker

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

- [“Expert Linker” on page 2-18](#)
- [“Data Elimination” on page 2-19](#)
- [“New Default LDF Behavior” on page 2-19](#)

For more information, refer to the *VisualDSP++ 3.0 Linker and Utilities Manual for ADSP-218x and ADSP-219x DSPs*.

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.

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

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.

New Default LDF Behavior

The default `.LDF` file no longer enables you to enter user mode by default. The `SMALL` configuration, remaining in supervisor mode and not registering event handlers, is now the default.

Note: To be in user mode, you must pass the `USERMODE` option to the linker.

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 errors and warnings when returning a result. Error reporting has been enhanced as follows.

- Every error 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 message.
- 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

New Features and Enhancements

- Every 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.
- You can promote or demote error messages individually. You can demote an error to a warning or remark, and you can promote a remark or warning to an error. You can also restrict the errors that can be promoted or demoted. For example, a condition in the input that might crash the tool can be set to ensure it is always an error.
- You can suppress or restrict the reporting of an individual error message at the command line through `#pragma` in the input source. The form of the pragma is consistent across all tools to enable header files to be shared.

Documentation

To view an explanation of the error message, select the six-character error identifier (for example, `cc0251`) in the **Output** window's **Build** page and 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. To access VisualDSP++ online documentation from online Help, click the **Start** button and choose **Programs**, **VisualDSP**, and **VisualDSP++ Help**.

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

- Datasheets directory contains one .PDF file for each data sheet.
- Tools Manuals directory contains one .PDF file for each tool manual.
- Hardware Manuals directory contains one .PDF file for each chapter in each manual.

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

VisualDSP++ Kernel

The following VisualDSP++ Kernel (VDK) features have been added in Release 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* 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

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.

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.

VisualDSP++ 3.0 produces code compatible with VisualDSP++ 2.0, so the 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 the VisualDSP++ 2.0. Similar, 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 3-12](#)
- [“ADSP-219x DSP Loader/Splitter” on page 3-14](#)
- [“ADSP-2192-12 DSP Loader/Splitter” on page 3-15](#)
- [“File Formats” on page 3-15](#)
- [“VDK Differences” on page 3-16](#)

Assembler and Preprocessor Differences

VisualDSP++ 3.0 ADSP-218x and ADSP-219x DSP assemblers have some differences you may encounter when upgrading from VisualDSP++ 2.0. There are some differences to the preprocessor as well.

The following changes have the most impact on your existing assembly projects.

- [“Command Line Switches” on page 3-2](#)
- [“Directives” on page 3-3](#)
- [“Built-in Functions and Operators” on page 3-4](#)
- [“Table 3-4 briefly describes the new assembly built-in functions and preprocessor operator.” on page 3-4](#)

Command Line Switches

ADSP-21xx DSP assemblers introduce the following command line switches ([Table 3-2](#)).

Table 3-2. New Assembler Switches

Switch	Description
<code>-flags-pp -opt1 [, -opt2...]</code>	Passes the specified option or a list of options to the compiler. Used when compiling <code>.IMPORT C</code> header files.
<code>-proc ADSP-ID</code>	Specifies an ADSP-218x or ADSP-219x processor for which the assembler should produce suitable code.
<code>-Wnumber1 [, number2 ...]</code>	Suppresses any report of the specified warning or a list of warning.

Directives

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

Table 3-3. New Assembly Directives

Directive	Description
.ENDIF	Ends a conditional assembly block.
.ELIF	Adds an alternative condition to the .IF/.ENDIF conditional block.
.ELSE	Adds an alternative condition to the .IF/.ENDIF conditional block.
.EXTERN STRUCT	References a global symbol that was defined in another file.
.IF	Begins a conditional assembly block.
.IMPORT	Provides the assembler with the structure layout (C struct) information.
.LIST	Starts listing of source lines.
.LIST_DATA	Starts listing of data opcodes.
.LIST_DATFILE	Starts listing of data initialization files.
.LIST_DEFTAB	Sets the default tab width for listings.
.LIST_LOCTAB	Sets the local tab width for listings.
.LIST_WRAPDATA	Starts wrapping opcodes that do not fit listing columns.
.NOLIST	Stops listing of source lines.
.NOLIST_DATA	Stops listing of data opcodes.
.NOLIST_DATFILE	Stops listing of data initialization files.

Assembler and Preprocessor Differences

Table 3-3. New Assembly Directives (Cont'd)

Directive	Description
<code>.NOLIST_WRAPDATA</code>	Stops wrapping opcodes that do not fit listing columns.
<code>.STRUCT</code>	Defines and initializes data objects based on C typedefs from <code>.IMPORT C</code> header files.

Built-in Functions and Operators

[Table 3-4](#) briefly describes the new assembly built-in functions and preprocessor operator.

Table 3-4. New Assembly Functions and Preprocessor Operator

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

Predefined Macros

Predefined macros are defined by the assembler to specify the architecture and language being processed. The following table lists two new macros defined by the assembler when it invokes the preprocessor.

Table 3-5. Assembler Feature Macros

Macro Definition	Notes
<code>-D_LANGUAGE_ASM =1</code>	Always present.
<code>-D_LANGUAGE_C=1</code>	Used for C compiler calls to specify <code>.IMPORT</code> headers. Replaces <code>_LANGUAGE_ASM</code> .

Compiler and Library Differences

The compiler differences between Releases 3.0 and 2.0 are as follows.

- [“Command Line Switches” on page 3-6](#)
- [“Input and Output Files” on page 3-8](#)
- [“Pragmas” on page 3-9](#)
- [“Library Functions” on page 3-11](#)

Command Line Switches

VisualDSP++ 3.0 compilers introduce some new command line switches. These C mode switches are listed in [Table 3-6](#).

Table 3-6. New Compiler Switches

Switch	Description
-alttok	Allows alternative keywords and sequences in sources.
-debug-types	Supports building a *.h file directly and writing a complete set of debugging information for the header file.
-dollar	New for ADSP-218x DSP compiler. Accepts dollar signs in symbols.
-J	Performs "old-style" preprocessing.
-Mt <i>filename</i>	Makes dependencies for the specified source file.
-MQ	Generates make rules only; does not compile. No notification when input files are missing.
-no-dir-warnings	Disables warnings about include and library switch directories that do not exist.

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

Switch	Description
-no-dollar	New for the ADSP-218x DSP compiler. Stops accepting dollar signs in symbols.
-no-widen-muls	Disable widening multiplications optimization.
-opt-forspace	Produces code optimized for minimal size.
-path tool <i>directory</i>	Updated for the ADSP-219x DSP compiler. Now uses the specified directory as the location of the driver definition file, in addition to the assembler, compiler, archiver, or linker.
-pch	Generates and uses precompiled header files (*.pch).
-pchdir <i>directory</i>	Specifies the location of PCH repository.
-proc ADSP- <i>ID</i>	Specifies a processor for which the compiler should produce suitable code.
-val-global < <i>name-list</i> >	Specifies that the names given by < <i>name-list</i> > are present in all globally defined variables.
-W{error remark suppress warn} <i>num- ber</i>	Functionally modified. Overrides the default severity of the specified diagnostic messages (errors, remarks, or warnings).
-workaround	Enables code generator workaround for specific hardware defects.

Compiler and Library Differences

The compilers of the current release do not support all the command line switches presented in VisualDSP++ 2.0. The removed switches are listed in [Table 3-7](#).

Table 3-7. Obsolete Compiler Switches

Switch	Description
-ansi	Supports ANSI/ISO standard C language.
-deps	Instructs driver to rebuild only components of a target that have changed.
-Fno	Disables the specified level of optimization.
-MD	Generates make rules and outputs to a file.
-no_std_assert	Ignores any predefined assertions and predefined system-specific preprocessor macros.

Input and Output Files

VisualDSP++ 3.0 compilers can process Interface Definition Language, template instantiation, memory map, and symbol map files, as described in [Table 3-8](#).

Table 3-8. New Input/Output Files

File Extension	Description
.IDL	Interface Definition Language specifications for VCSE components and interfaces.
.II	ADSP-219x DSP compiler only. Template instantiation files; used internally by the compiler when instantiating C++ templates.

Table 3-8. New Input/Output Files (Cont'd)

File Extension	Description
.MAP	DSP system memory map file output.
.SYS	DSP system symbol map file output.

Pragmas

Table 3-9 through Table 3-14 list VisualDSP++ 3.0 pragmas for data alignment, interrupt handling, alternative registers usage, optimization, linkage, and stack usage.

Table 3-9. Data Alignment Pragma

Pragma	Description
<code>align <i>number</i></code>	Applies to the symbol that immediately follows the <i>number</i> argument. Useful for declaring circular buffer arrays.

Table 3-10. Interrupt Handler Pragma

Pragma	Description
<code>interrupt</code>	Applies to the function definition that immediately follows the pragma. Causes generation of the function code that can be used as interrupt handler.

Table 3-11. Alternative Registers Pragma

Pragma	Description
<code>altregisters</code>	Used with the <code>interrupt</code> pragma. Specifies that the secondary registers can be used for saving and restoring of registers.

Compiler and Library Differences

Table 3-12. Optimization Pragas

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 3-13. Linking Pragas

Pragma	Description
<code>linkage_name</code>	Ensures that the named variable or function declaration is used as the external reference.
<code>weak_entry</code>	Applies to the symbol or functional definition that immediately follows the pragma. Causes generation of the symbol or function with weak linkage.

Table 3-14. ADSP-218x DSPs Stack Usage Pragma

Pragma	Description
<code>make_auto_static</code>	Applies to the ADSP-218x DSP architectures only. The pragma is associated with the function definition that immediately follows the pragma. It directs the compiler to place all automatic variables used in the function in <code>static</code> store.

Library Functions

The libraries provided with the current release of software have a number of new and enhanced functions, as described in [Table 3-15](#).

Table 3-15. Library Functions

Function	Description
<code>external_memory_read()</code> <code>external_memory_write()</code>	<code>sysreg.h</code> allows the use of functions that generate efficient inline instructions to implement read and write of values from and to external memory.
<code>interruptf</code>	The <code>interruptf</code> interrupt dispatcher does the same as <code>interrupt</code> , except it switches between primary and secondary register sets to save and restore registers instead of using the data stack.
<code>signalf</code>	The <code>signalf</code> interrupt dispatcher does the same as <code>interrupt</code> , except it switches between primary and secondary register sets to save and restore registers instead of using the data stack.
<code>cotf</code>	Added as part of the <code>cot</code> function.
CFIR	For ADSP-219x DSPs only. Improved function description and syntax synopsis.
FIR	For ADSP-219x DSPs only. Improved function description and syntax synopsis.
FIR_DECIMA	For ADSP-219x DSPs only. Improved function description and syntax synopsis.
FIR_INTERP	For ADSP-219x DSPs only. Improved function description and syntax synopsis.
IIR	For ADSP-219x DSPs only. Improved function description and syntax synopsis.
<code>acos</code> , <code>asin</code> , <code>atan</code> , <code>atan2</code> , <code>cos</code>	Trigonometric functions. Improved function description and syntax synopsis.

Linker Differences

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

- [“Command Line Switches” on page 3-12](#)
- [“LDF Operators” on page 3-13](#)
- [“Builtin LDF Macros” on page 3-13](#)
- [“LDF Commands” on page 3-13](#)

Command Line Switches

The switches listed in [Table 3-16](#) have been introduced in VisualDSP++ 3.0.

Table 3-16. 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 as 1.
<code>-proc <i>ADSP-ID</i></code>	Specifies a target processor for which the linker should produce suitable code.

LDF Operators

[Table 3-17](#) lists the new LDF operator for use in memory address expressions.

Table 3-17. New LDF Operator

LDF Operator	Description
<code>ABSOLUTE(<i>expression</i>)</code>	The linker returns the value <i>expression</i> . Use this operator to assign an absolute address to a symbol.

Builtin LDF Macros

[Table 3-18](#) describes the new LDF built-in macro.

Table 3-18. New Builtin LDF Macro

Switch	Description
<code>\$COMMAND_LINE_OUTPUT_DIRECTORY</code>	The macro expands into the path of the output directory, which is set with the linker's <code>-od</code> switch (or <code>-o</code> switch when <code>-od</code> is not specified).

LDF Commands

The following LDF command ([Table 3-19](#)) is no longer supported.

Table 3-19. Obsolete LDF Command

Switch	Description
<code>DYNAMIC()</code>	The <code>DYNAMIC()</code> command allows a Dynamic Linking Object (DLO) to be located at runtime.

ADSP-219x DSP Loader/Splitter

Table 3-20 lists the command line options that are new to VisualDSP++ 3.0 loader, which also includes the splitting functionality, for ADSP-2191, ADSP-2195, ADSP-2196, ADSP-21990, ADSP-21991, and ADSP-21992 DSPs.

Table 3-20. New ADSP-219x Loader Switches

Switch	Description
-dADSP-2195 -dADSP-2196 -dADSP-21990 -dADSP-21991 -dADSP-21992	Specifies the processor. The new DSP IDs are: ADSP-2195, ADSP-2196, ADSP-21990, ADSP-21991, and ADSP-21992.
-proc <i>ADSP-ID</i>	Preferable way to specify the processor. The valid IDs are: ADSP-2191, ADSP-2195, ADSP-2196, ADSP-21990, ADSP-21991, and ADSP-21992.
-p <i>address</i>	Specifies a hexadecimal integer as the PROM starting address.
-readall	Creates a non-bootable image and non-boot stream image in the same output file, together with the boot-loadable image.
-romsplitter	Creates a non-bootable image only. This switch overwrites -b <i>boot_type</i> and any other switch bounded by boot types.
-maskaddr #	Masks all EPROM address bits above or equal to #.
-split #[#]	Valid only when width is 16. Generates two output files for the 16-bit wide EMI data bus.
-forcefirstbyte	Forces the writing of the first byte of the UART boot stream. Use this option when the bit rate is high.
-host3bytes	HOST boot only, when width is 8. Uses three bytes to represent 24-bit data. The default is four bytes with one byte of zero padding.

Refer to the *VisualDSP++ 3.0 Linker and Utilities Manual for ADSP-218x and ADSP-219x DSPs* and online Help for more information about the ADSP-219x loader.

ADSP-2192-12 DSP Loader/Splitter

[Table 3-21](#) lists the command line option that is new to VisualDSP++ 3.0.

Table 3-21. New ADSP-2192-12 Loader Switch

Switch	Description
-proc ADSP-2192	The preferable way to specify the ADSP-2192-12 processor.

Refer to the *VisualDSP++ 3.0 Linker and Utilities Manual for ADSP-218x and ADSP-219x DSPs* for further information about the ADSP-2192-12 loader.

File Formats

[Table 3-8](#) lists the newly added file formats.

Table 3-22. 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 ADSP-218x and ADSP-219x DSPs*.

VDK Differences

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

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