

VISUALDSP++[®] 3.5

Product Release Bulletin

for 32-Bit Processors

Revision 1.0, March 2004

Part Number
82-000035-12

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



Copyright Information

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

Printed in the USA.

Disclaimer

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

Trademark and Service Mark Notice

The Analog Devices logo, the CROSSCORE logo, VisualDSP++, SHARC, TigerSHARC, and EZ-KIT Lite are registered 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	xi
Intended Audience	xi
Manual Contents	xii
Technical or Customer Support	xiii
Supported Processors	xiii
Product Information	xiv
MyAnalog.com	xiv
Processor Product Information	xiv
Related Documents	xv
Online Technical Documentation	xvi
From VisualDSP++	xvii
From Windows	xvii
From the Web	xviii
Printed Manuals	xviii
VisualDSP++ Documentation Set	xviii
Hardware Tools Manuals	xviii
Processor Manuals	xix

CONTENTS

Data Sheets	xix
Notation Conventions	xx

INTRODUCTION

Product Release Description	1-2
VisualDSP++ 3.5 System Requirements	1-2
Platform and Processor Support	1-3
VisualDSP++ 3.5 Product Highlights	1-4

VISUALDSP++ 3.5 MAJOR CHANGES

Changes to Installer	2-2
Discrete Installer	2-2
Emulation Tools Included in Installer	2-4
Changes to SHARC Compiler	2-5
SHARC Command-Line Switches	2-5
Linker Description Files	2-7
Start-Up Code and Libraries	2-8
Using the Start-Up Files	2-8
Interrupt Dispatcher Restrictions	2-10
math.h Header File	2-10
stdio.h Header File	2-11
DSP Header Files	2-12
fft_mag Function	2-13
Miscellaneous Changes	2-13

Changes to TigerSHARC Compiler	2-14
TigerSHARC Command-Line Switches	2-14
Linker Description Files	2-16
Workaround Libraries	2-16
Start-Up Code and Libraries	2-17
Miscellaneous Changes	2-17
Generating Map File in .XML Format	2-18

NEW FEATURES AND ENHANCEMENTS

VisualDSP++ IDDE	3-2
New Processor Support	3-3
Multiple Project Support	3-3
XML Project File Format	3-3
Project Migration	3-4
License Management	3-4
Data Streaming and Logging	3-4
Profile-Guided Optimization	3-4
Integrated Source Code Control	3-4
Automation Aware Scripting Engine	3-5
Address Bar in Memory and Disassembly Windows	3-5
Menus with Icons	3-5
Assembler for TigerSHARC Processors	3-6
Assembler Macros	3-6
VCSE Optimization Directives	3-7

CONTENTS

PAGEWIDTH Directive	3-7
Assembler Command-Line Switch	3-7
Preprocessor Macros	3-8
Preprocessor Command-Line Switches	3-8
Include Path Search Algorithm Matches Compiler	3-9
Assembler for SHARC Processors	3-10
New Assembler Directives	3-10
PAGEWIDTH Directive	3-11
Assembler Command-Line Switches	3-11
Assembler Macros	3-12
Preprocessor Macros	3-12
Preprocessor Command-Line Switches	3-13
Include Path Search Algorithm Matches Compiler	3-13
Compiler and Library for TigerSHARC Processors	3-15
File Extensions	3-16
Compiler Command-Line Switches	3-16
Optimization Control	3-19
Program Argument Support	3-19
Compiler Built-In Functions	3-19
builtins.h Header File	3-20
Optimization Guidance Built-in Functions	3-20
Data Alignment Buffer (DAB) Built-in Functions	3-20
Circular Buffer Data Alignment Buffer (DAB) Built-in Functions	3-21

C++ Exception Handling	3-21
Pragmas	3-22
Predefined Compiler Macros	3-23
GCC Compatibility Extensions	3-24
File I/O Support	3-25
C/C++ Libraries and Start-Up Files	3-25
C Library Functions	3-26
DSP Run-Time Library	3-27
Compiler and Library for SHARC Processors	3-28
File Extensions	3-29
Compiler Command-Line Switches	3-29
Optimization Control	3-32
Pragmas	3-33
GCC Compatibility Extensions	3-35
Predefined Compiler Macro	3-36
seg_int_code Memory/Section Name	3-37
Support for argv/argc	3-38
File I/O Support	3-38
Run-Time Libraries and Start-Up Files	3-39
C Run-Time Library Functions	3-39
DSP Run-Time Library Functions	3-40
Linker and Utilities	3-44
Modified Link Page in Project Options Dialog Box	3-44

CONTENTS

Migrating LDFs from Previous Installations	3-46
C++ Global Constructors and Destructors	3-46
Run-Time Argument Processing for PGO	3-47
C++ Exceptions	3-47
Zero-Initialized Data	3-47
Run-Time Initialized Data (TigerSHARC Processors Only)	3-48
Modifying Interrupt Registers (SHARC Processors Only) ...	3-48
Others	3-48
Linker Command-Line Switches	3-48
Updated List of LDF Keywords	3-50
Modifications to LDF Commands	3-51
KEEP() LDF Command	3-51
KEEP_SECTIONS()	3-51
MAP()	3-51
MEMORY{} LDF Command	3-51
SECTIONS{} LDF Command	3-52
OVERLAY_GROUP{} Command	3-53
Expert Linker	3-53
Expert Linker Menu Updates	3-54
Profiling Object Sections	3-54
Memory Map File (.XML)	3-55
Archiver	3-55
Archiver Switches	3-55

Warnings for Duplicate Library Entries	3-57
Improved Support for File Specifications	3-57
Tagging an Archive with Version Information	3-57
Basic Version Information	3-57
User-Defined Version Information	3-57
Printing Version Information	3-58
Removing Version Information from an Archive	3-58
Checking Version Number	3-58
Adding Text to Version Information	3-58
Loaders and Splitter	3-59
TigerSHARC Loader Features	3-59
SHARC Loader Features	3-59
Splitter for TigerSHARC Processors	3-60
HEXUTIL Command Line Tool	3-60
VCSE	3-61
VDK	3-63
Simulators for TS101 and TS20x Processors	3-64
Object Protection	3-64
Documentation Changes	3-64
Compiler Manuals	3-64
VisualDSP++ 3.5 User's Guide	3-65
Online Help	3-65
Error Messages	3-65

CONTENTS

Merged Index	3-66
Online Manuals	3-66

OBSOLETE OR REMOVED FEATURES

Assembler and Preprocessor	4-2
Preprocessor Command-Line Switch	4-2
Compiler and Library for SHARC Processors	4-3
C/C++ Compiler Command-Line Switches	4-3
Compiler and Library for TigerSHARC Processors	4-5
Linker	4-7
Tcl Scripting Engine	4-7

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++ 3.5 for 32-bit SHARC® (ADSP-21xxx) and TigerSHARC® (ADSP-TSxxx) digital signal processors. It also describes the differences (obsolete features and functions) between VisualDSP++ 3.5 and previous VisualDSP++ releases.

For details, refer to the VisualDSP++ 3.5 manuals listed in [“Related Documents”](#) and online Help.

Intended Audience

This publication is primarily intended for programmers who are upgrading from the previous releases of VisualDSP++ development software and who want an overview of the changes to VisualDSP++ 3.5.

Manual Contents

This manual consists of:

- Chapter 1, “[Introduction](#)”
Describes VisualDSP++ 3.5 and its benefits, provides the minimal system requirements for running the product, and lists the supported processors.
- Chapter 2, “[VisualDSP++ 3.5 Major Changes](#)”
Describes major changes in VisualDSP++ 3.5 compared to VisualDSP++ 3.0 releases.
- Chapter 3, “[New Features and Enhancements](#)”
Describes what is new in the VisualDSP++ 3.5 IDDE, assembler, compiler, linker, loader, and documentation. Also describes the new features in the Expert Linker (EL), VisualDSP++ Component Software Engineering (VCSE), and the VisualDSP++ Kernel (VDK).
- Chapter 4, “[Obsolete or Removed Features](#)”
Describes the removed/obsolete features in VisualDSP++ 3.5, compared to the previous VisualDSP++ software release as they pertain to the code generation tool chain: commands, switches, operators, directives, pragmas, keywords, macros, and library functions.

Technical or Customer Support

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

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

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

Supported Processors

VisualDSP++ 3.5 release supports 32-bit SHARC (ADSP-21xxx) and TigerSHARC (ADSP-TSxxx) processors. For more information, refer to [“Platform and Processor Support” on page 1-3](#).

Product Information

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

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

MyAnalog.com

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

Registration

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

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

Processor Product Information

For information on embedded processors and DSPs, visit our Web site at www.analog.com/processors, which provides access to technical publications, data sheets, 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.

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

Related Documents

For information on product related development software, see these publications:

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

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

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

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

VisualDSP++ 3.5 Assembler and Preprocessor Manual for SHARC Processors

VisualDSP++ 3.5 Assembler and Preprocessor Manual for TigerSHARC Processors

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

VisualDSP++ 3.5 Loader Manual for 32-Bit Processors

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

VisualDSP++ Kernel (VDK) User's Guide

VisualDSP++ Component Software Engineering User's Guide

Quick Installation Reference Card

PREFACE

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

Visit the Technical Library Web site to access all processor and tools manuals and data sheets:

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

Online Technical Documentation

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

Each documentation file type is described in the following table.

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

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

From VisualDSP++

From VisualDSP++ environment:

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

From Windows

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

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

Using Windows Explorer

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

Using the Windows Start Button

- Access VisualDSP++ online Help by clicking the **Start** button and choosing **Programs**, **Analog Devices**, **VisualDSP++**, and **VisualDSP++ Documentation**.

PREFACE

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

From the Web

To download manuals, at the following Web site:

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

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

Printed Manuals

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

VisualDSP++ Documentation Set

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

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

Hardware Tools Manuals

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

Processor Manuals

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

Data Sheets

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

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

Notation Conventions

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

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

Example	Description
Close command (File menu)	Titles in reference sections indicate the location of an item within the VisualDSP++ environment's menu system (for example, the Close command appears on the File menu).
{this that}	Alternative items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as <i>this</i> or <i>that</i> . One or the other is required.
[this that]	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional <i>this</i> or <i>that</i> .
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of <i>this</i> .
.SECTION	Commands, directives, keywords, and feature names are in text with letter gothic font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
	This symbol indicates a note that provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.
	This symbol indicates a warning that advises on an inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word Warning appears instead of this symbol.

1 INTRODUCTION

This chapter describes the product, VisualDSP++, and the requirements for running its latest revision, VisualDSP++ 3.5. It also lists the supported processors and some of the benefits provided by this release.

The information is organized as follows.

- [“Product Release Description” on page 1-2](#)
- [“VisualDSP++ 3.5 System Requirements” on page 1-2](#)
- [“Platform and Processor Support” on page 1-3](#)
- [“VisualDSP++ 3.5 Product Highlights” on page 1-4](#)

Product Release Description

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

VisualDSP++ 3.5 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 comprises of the processor-specific software necessary for completing a DSP-based project: simulator, assembler, C/C++ compiler and libraries, linker, loader, splitter, and utilities. Analog Devices also provides VisualDSP++ Component Software Engineering (VCSE) with the VIDL compiler and VisualDSP++ Kernel (VDK),

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.

As the successor to VisualDSP++ 3.0, this software release incorporates a number of new features and enhancements, as described in [“New Features and Enhancements”](#).

VisualDSP++ 3.5 System Requirements

To install and run VisualDSP++ 3.5, your PC must provide the following software, configuration, and system resources.

- Windows 98 SR2 (or greater)/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

Platform and Processor Support

The name “SHARC” refers to a family of Analog Devices, Inc. high-performance 32-bit floating-point digital signal processors that can be used in speech, sound, graphics, and imaging applications.

VisualDSP++ currently supports the following SHARC processors:

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

The name “TigerSHARC” refers to a family of Analog Devices, Inc. floating-point and [8-bit, 16-bit and 32-bit] fixed-point processors.

VisualDSP++ currently supports the following TigerSHARC processors:

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

VisualDSP++ 3.5 Product Highlights

Major product highlights and benefits include:

- **Platform and Processor Support.**

VisualDSP++ 3.5 for 32-bit processors supports all Analog Devices 32-bit SHARC (ADSP-21xxx) and TigerSHARC (ADSP-TSxxx) processors on Windows® 98, Windows ME, Windows NT 4.0, Windows 2000, and Windows XP. See [“Platform and Processor Support”](#) for the list of all supported processors.

- **Robust and Flexible Project Management**

The VisualDSP++ 3.5 Integrated Development and Debugging Environment (IDDE) provides robust and flexible project management for the development of applications and includes access to all the activities necessary to create and debug projects. It enables users to open and switch between multiple projects in the same session.

- **Multiple Project Support**

VisualDSP++ provides the ability to switch among multiple open projects in the same IDDE session. The **Project** window displays active projects.

- **License Management in the IDDE**

License management (installation and validation) has been integrated into the VisualDSP++ IDDE. Installing a Flexible License Manager (FlexLM) license server is still handled by the separate installation application.

- **Data Streaming and Logging**

VisualDSP++ now offers the ability to stream and log data from a target processor without halting the processor. The IDDE takes advantage of this capability in plot windows. If the target supports background

telemetry channel (BTC), the plot window is updated while the target is running. See the *VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors* for information about using BTC.

- **Time-Saving Debugger**

The VisualDSP++ debugger has a user-friendly, common interface to simulators and emulators available from Analog Devices and participating third-parties. In addition, the debugger has many features that greatly reduce debugging time. Users can view C/C++ source code interspersed with the resulting assembly code, profile execution of a range of instructions in a program, set watchpoints on hardware, view program and data memory, and trace instruction execution and memory accesses. These time-saving features enable users to quickly correct coding errors, identify bottlenecks, and examine signal processor performance all within the debugger. Also, when used with the simulator, the debugger can generate inputs, outputs, and interrupts to simulate real-world application conditions and give users better insight for tuning the performance of their code.

- **VisualDSP++ Kernel**

The VisualDSP++ Kernel (VDK) is a real-time operating system that provides scheduling and resource allocation services applicable to embedded programming. VDK applications are configured within the IDDE and initial source files are generated from templates, where appropriate, to provide a working project framework for inserting your code.

- **Automation API and Aware Scripting Engine**

The Automation Aware Scripting Engine using the ActiveX script host framework allows the use of multiple popular scripting languages such as VBScript and JavaScript to access the Automation API. Now you are able to interface with the IDDE using either a single command or a script file.

VisualDSP++ 3.5 Product Highlights

- **Multiple Processor (MP) Support**

The VisualDSP++ multiple processor (MP) support provides a single seamless interface for debugging multiple processors on the same physical hardware. You can easily issue parallel step, run, and halt commands to all of the applicable processors. You can easily pick and choose individual processor registers, or memory sets of interest by pinning those that should be updated between runs, halts, and steps. This feature also eliminates screen clutter in multiple processor debugging.

- **Background Telemetry Channel Support**

The Background Telemetry Channel (BTC) feature is a mechanism for exchanging data between a host and target application, with minimal intrusion on the target system's "real-time" characteristics and minimal addition on development and debugging time. BTC enables real-time data collection and status messaging, eliminating the overhead involved with halting the target application, getting the desired information, and then restarting the target application. VisualDSP++ 3.5 extends BTC support for SHARC and TigerSHARC processors to provide direct benefit from BTC in the IDDE plot window. In this case, the plot window reads the target's memory contents on a user-defined time interval and upon receipt of the data, converts them to the desired data type, and updates the plot display immediate viewing and analysis.

- **Statistical Profiling**

Statistical profiling allows JTAG emulator debug targets to take advantage of a more generalized form of profiling.. The debugger has the ability to unobtrusively and statically sample the target processors and then present a graphical display of the resultant samples for review. This feature enables you easily and effortlessly identify where your application is spending more of its time.

- **Graphical Plotting**
VisualDSP++ includes numerous graphical plotting options, including Line, Constellation, Eye Diagrams, and 3D waterfall plots that help users to better visualize, analyze, and understand their data. The plotting engine is also capable of doing some simple data processing such as Fast Fourier Transform, 2-D Fast Fourier Transform, and decibel conversion on the data before it is displayed.
- **VisualDSP++ Component Software Engineering (VCSE)**
VCSE supports an Interface Definition Language (IDL) and compiler that allow developers to create and reuse components without having to become familiar with the detail of the model and the mechanisms it involves. Components can easily be integrated into an application and are reusable. VCSE dramatically simplifies the process of incorporating and utilizing components from a variety of developers. The VIDL compiler can automatically generate a test shell for each component that can be used to monitor a component, to validate its actions, and to measure the resources used by it.
- **Pipeline Viewer**
The Pipeline Viewer allows you to easily view the instruction flow through the sequencer's pipeline. Stalls, aborts, and other pipeline events are graphically represented in an easy-to-read format for the developer. Visualization of the pipeline and of the events, which occur within it, allow you to better understand where and why latencies and stalls are being introduced into an executable file. Armed with this knowledge, you can effectively and efficiently optimize an executable file's instruction sequence to minimize the number of undesirable pipeline events.
- **C/C++ Compiler**
The best-in-class C/C++ compiler is a time-saver for developers generating application code. The compiler generates efficient application code optimized for both code density and execution

VisualDSP++ 3.5 Product Highlights

time. C/C++ code modules can be easily interfaced with assembly code modules, allowing users to program in C/C++ and still use assembly for time-critical loops. Beyond that, developers can realize an additional significant decrease in their time to market with the ability to efficiently work with complex data types and take advantage of specialized operations without having to understand the underlying architecture.

Among other notable features, the VisualDSP++ 3.5 compiler offers C++ standard exception handling as defined by the ISO/IEC 14882:1998 standard.

- **Profile-Guided Optimization**

Profile-Guided Optimization (PGO) is an iterative compilation approach which uses information from previous compilations to improve the optimizer's decisions on the code being compiled. Traditionally, a compiler compiles each function only once and attempts to generate code that will perform optimally in most cases by making reasonable default assumptions in the behavior of that code. With PGO, the compiler makes educated assumptions based on data collected during previous executions of the generated code and subsequently makes decisions about the relative importance of parts of the application rather than simply using the default behavior.

This technique enables large gains to be realized in the run-time performance and code density of the program automatically, without additional effort by the users.

- **Expert Linker**

The Expert Linker is a graphical utility that makes it easier for users to produce a Linker Description File (LDF) without having to learn the LDF syntax. The graphical representation of commands in an LDF file also allows the engineer to easily make changes or to generate a new LDF file. In VisualDSP++ 3.5, the Expert Linker

also allows users to easily profile object sections of their program, identify “hotspots” graphically, and optimize code placement in one single step with minimal additional effort.

- **Integrated Source Code Control**

The Source Code Control (SCC) integration in the IDDE enables users to easily connect to SCC applications installed on their machines through the Microsoft Common Source Code Control (MCSCC) interface, which is widely supported by leading SCC vendors. Using the plug-in, you can also access commonly used features (such as getting the latest version, checking out, and removing a selected file from source code control) of these SCC applications, launch the SCC applications, and view a file's source control status in a project window quickly and conveniently without leaving the IDDE.

- **Program Flow Manipulation for the TigerSHARC Cycle-Accurate Simulator.**

The simulator supports manual change of the program counter value in the **PC Register** window to specify which instruction the simulator executes next. When a PC value is changed, the simulator automatically flushes the pipeline and aborts all its instructions. After the pipeline is flushed, normal execution resumes from a memory address indicated by the new PC value.

- **New Data Format Support in Memory and Register Windows**

Data in a **Register** or **Memory** window can be displayed in 8-bit or 32-bit character format. When compiled and loaded into memory, a code `char string[]` can be displayed in a **Memory** window as defined. Select the memory address of the symbol string and choose the display format for which the code was compiled.

VisualDSP++ 3.5 Product Highlights

- **New Select Loader Program Command Added to the Settings→Simulator Menu for TigerSHARC Processors**
A custom loader program can be specified that runs automatically every time before a user program is loaded. When VisualDSP++ starts, the simulator defaults to a standard loader program:
`VisualDSP\Ts\ldr\TS201_prom.dxe.`
- **Splitter Support for TigerSHARC Processors**
VisualDSP++ 3.5 includes a TigerSHARC splitter.
- **Address Bar in Disassembly and Memory Windows**
When enabled, an address bar is displayed in **Disassembly** windows and memory windows. Use the address bar to navigate by address, symbol, or expression. The address bar maintains a most recently used history of visited locations.
- **Menus with Icons**
Icons now appear beside menu commands that have corresponding toolbar buttons.

2 VISUALDSP++ 3.5 MAJOR CHANGES

This chapter summarizes the major changes between the VisualDSP++ 3.5 and 3.0 releases.

The chapter details:

- [“Changes to Installer” on page 2-2](#)
- [“Changes to SHARC Compiler” on page 2-5](#)
- [“Changes to TigerSHARC Compiler” on page 2-14](#)
- [“Generating Map File in .XML Format” on page 2-18](#)

Please note that all obsolete/removed features are listed in Chapter 4.

Changes to Installer

Discrete Installer

VisualDSP++ 3.5 and all future releases of VisualDSP++ will install discretely from other versions of VisualDSP++. Historically, releases installed into a common, shared location. The new installer allows you to switch between different releases of VisualDSP++ more efficiently. Each installation of VisualDSP++ will have its own default installation directory and **Start** menu icons.

 Installing VisualDSP++3.5 into the same location as another version of VisualDSP++ is not supported and is highly discouraged.

Discrete installation relies on certain facilities within Microsoft Windows that are not supported in older versions of the operating system. VisualDSP++ will not be fully discrete under Windows 95, Windows 98, and Window NT 4.0. Users of those operating systems are highly encouraged to migrate to Windows XP. On the named, older versions of Windows, the debug targets to which VisualDSP++ IDDE can connect will not be handled discretely. VisualDSP++ will be able to connect to another installation's targets; however, the build tools will be managed discretely.

When using multiple versions of VisualDSP++ on the same host PC, be aware of these restrictions and guidelines:

1. Multiple product installations have no visibility into each other (by design) to ensure that each installation behaves exactly as it does on a dedicated host PC. One implication of this is that user settings, such as JTAG platforms, debug session, and preferences need to be set under each installation.

VisualDSP++ 3.5 Major Changes

2. License management is likewise discrete. In most cases, licenses already installed and validated for a past release of VisualDSP++ can be imported into VisualDSP++ 3.5. Simply copy the `...\\System\\license.dat` file from the historical release of VisualDSP++ to the `...\\System` directory of the new installation. Alternatively, the license can be reregistered from within VisualDSP++ 3.5.
3. If building from the command line, make sure that the `PATH` environment variable is correctly set. The VisualDSP++ installer places a helper file, `VDSP3_5.BAT`, in the VisualDSP++ directory that correctly sets the `PATH` variable to point to that installation of VisualDSP++ 3.5.
4. Very old releases of the tools had used an environment variable, `ADI_DSP`. This variable should not be set at all when VisualDSP++ 3.5 is being used. Third-party debug targets under older releases can be selected from the **New Session** dialog box by selecting the **Show All Targets** checkbox.
5. If a VisualDSP++ connection is via Automation (for example, from a VBScript), the last registered version of `IDDE.EXE` is invoked. To change the version of VisualDSP++, run the desired version of `IDDE.EXE` with the `/RegServer` switch.
6. VisualDSP++ project file (`.DPJ`) association is a Windows-wide setting and, thus, is not handled discretely; it is generally made to be associated with whichever version of VisualDSP++ was installed most recently. Association can be changed as desired within Windows. The exact procedure depends on a particular Windows version. For example, under Windows XP, select a `.DPJ` file, chose **Properties** from the context (right-click) menu, and click the **Change...** button next to the **Open with** field.

Changes to Installer

Emulation Tools Included in Installer

The emulation tools now are included in the base VisualDSP++ installation. There is no longer a second ICE software installation procedure. To install emulation support, select the appropriate components in the installation wizard. The installer handles the installation and update of the emulator hardware device driver.

Changes to SHARC Compiler

This section summarizes changes to the SHARC compiler environment since VisualDSP++ 3.0. As with any major upgrade, some of the improvements have changed how the compiler is invoked. The compiler's optimizing technology has undergone extensive revision as well, but most of these changes are purely internal.

In general, the changes can be grouped into:

- “[SHARC Command-Line Switches](#)” (some have been removed, some renamed)
- “[Linker Description Files](#)” (the default LDFs have been reorganized to reflect the new libraries)
- “[Start-Up Code and Libraries](#)” (the new libraries require different initialization)
- “[Miscellaneous Changes](#)” (other compiler changes)

SHARC Command-Line Switches

Some command-line switches have been removed, as they are no longer appropriate or enable archaic language dialects no longer supported by the compiler. Other switches have been removed because they historically supported facilities useful for UNIX[®]-targeted compilers but have become inappropriate for compilers targeted at signal processing and embedded platforms.

-swf_screening and -21160rev0

These switches instruct the compiler to work around particular hardware anomalies. They have been replaced by a uniform mechanism under the -workaround switch. Instead of invoking:

```
cc21k -21160 -swf_screening prog.c
```

Changes to SHARC Compiler

you now invoke:

```
cc21k -proc ADSP-21160 -workaround swfa prog.c
```

The other anomaly switches are also passed to `-workaround`, in a similar fashion. This change creates a uniform approach to hardware `-workaround` switches across Analog Devices compilers.

-circbuf

The old `-circbuf` switch has been renamed to `-force-circbuf` to better indicate that it is causing the creation of circular buffers that would not otherwise happen. The `-no-circbuf` switch exists to prevent the automatic creation of circular buffers. Explicit circular buffer usage, via the corresponding intrinsic functions, is still honored.

-alttok

The `-alttok` switch, which directs the compiler to allow digraph sequences in C and C++ source files, is no longer on by default as it was in previous releases.

-analog is renamed to -c89

The `-analog` switch was renamed to `-c89` to reference a particular revision of the ANSI C Standard.

-2106{0|1|2|65L}, -2116{0|1}, -2126{1|2|5|6|7} and -2136{3|4|5}

These switches, which were used to determine which target to compile for, were removed. Instead, use the `-proc processor` switch. For example, `processor` should be `ADSP-21065L` instead.

-inline, -no-inline

These switches are no longer supported because the inliner is integrated into the optimizer. When optimization is on, the inliner operates. When optimization is off, the inliner is disabled. See also `-0a` (auto-inlining).

-xml

This switch specified that map files generated by the linker should be in XML format. In VisualDSP++ 3.5, the linker always generates

XML-format map files, so the switch has been removed. The `-xml` switch is still accepted but has no effect.

Other Switches

The following switches have been removed because they correspond to language options no longer supported in VisualDSP++ 3.5.

`-traditional`, `-dollar`, `-no-dollar`, `-J`, `-force`, `-instantall`,
`-instantlocal`, `-instantused`, `-suppress`, `-tpautooff`.

The following switches have been removed because they were used to select between two compilation modes; only the default mode is available in VisualDSP++ 3.5.

`-bool`, `-explicit`, `-namespace`, `-newforinit`, `-newvec`, `-std`, `-typename`,
and `-wchar`. These switches now are always on.

The `-no-bool`, `-no-explicit`, `-no-namespace`, `-trdforinit`, `-no-newvec`,
`-no-std`, and `-no-wchar` switches are no longer available because their corresponding elections are not supported in VisualDSP++ 3.5.

Linker Description Files

There are many differences between the LDFs in VisualDSP++3.5 and preceding VisualDSP++ releases. Projects that use customized LDFs derived from earlier VisualDSP++ releases almost certainly require some modification in order to link successfully with VisualDSP++ 3.5.

The default linker description files for each processor in VisualDSP++ 3.5 are located within their own processor LDF directory (for example, `21k\LDF`, `211xx\LDF`, `212xx\LDF` and `213xx\LDF`).

The `__ctor_NULL_marker` is now located in a section called “`seg_ctdm1`”. This section is then merged into the normal `seg_ctdm` section, but an extra LDF command is required.

Changes to SHARC Compiler

C++ Exception Support

The compiler's C++ exception support relies on a support library (`libeh*.dll`) and exceptions-enabled versions of the run-time libraries (named with an “eh” suffix). In addition, the exceptions support requires several new input sections (`.fnt`, `.edt`, `.cht`, `.gdt`) that must be mapped to data areas.

-no-std-lib Support

The `-no-std-lib` compiler option indicates that the LDF should not use the default search path to locate libraries by defining linker macro `__NO_STD_LIB` to guard the `SEARCH_DIR` directive.

Start-Up Code and Libraries

Note the following sections to identify changes to start-up files and libraries.

- [“Using the Start-Up Files” on page 2-8](#)
- [“Interrupt Dispatcher Restrictions” on page 2-10](#)
- [“math.h Header File” on page 2-10](#)
- [“stdio.h Header File” on page 2-11](#)
- [“DSP Header Files” on page 2-12](#)
- [“fft_mag Function” on page 2-13](#)

Using the Start-Up Files

The start-up file is an assembly language procedure that initializes the processor and sets up processor features to support the C/C++ run-time environment. The source code for the default start-up files is in:

- `020_hdr.asm` for ADSP-21020 processors
- `06x_hdr.asm` for ADSP-2106x processors

- `16x_hdr.asm` for ADSP-2116x processors
- `26x_hdr.asm` for ADSP-2126x processors
- `36x_hdr.asm` for ADSP-2316x processors

These files are located in the processor-specific directories; for example in `VisualDSP\21k\lib\src\crt_src`.

The start-up files by default include `ARGV` support. The start-up files therefore will be larger than those in VisualDSP++ 3.0. To disable `ARGV` support and make the files smaller, you have to rebuild them without the `ARGV_SUPPORT` macro defined.

To rebuild the start-up files without `ARGV` support amend the processor's makefile within `$(VisualDSP)\$(processor)\lib\src\libc_src` to not define `ARGV_SUPPORT`.

Then from a command line, go to the processor's `libc_src` directory and run `$(VisualDSP)\gmake-378.exe -f Makefile_libc.2106x $(CRT_doj)`.

For example,

```
$(VisualDSP)\gmake-378.exe -f Makefile_libc.2106x 060_hdr.doj.
```

Macro names without preceding underscores (for example, `ADSP21065L`) are deprecated. Macro names with preceding underscores should be used (for example, `__ADSP21065L__`).

Interrupt Dispatcher Restrictions

The following are some restrictions on using interrupt dispatchers.

Interrupt Nesting Restrictions on ADSP-2116x/2126x/2136x DSPs:

On ADSP-2116x, ADSP-2126x and ADSP-2136x platforms, the interrupt vector code explicitly saves the `ASTATx`, `ASTATy` and `MODE1` registers on the status stack using a 'push STS' instruction. For the timer, `VIRPT` and `IRQ0-2` interrupts, the save occurs in addition to the automatic save of these registers. This has the effect of reducing the maximum depth of nested interrupts to between 10 and 15 levels, depending on whether the timer, `VIRPT` and `IRQ0-2` interrupts are used.

Restriction on Use of Super-Fast Dispatcher on ADSP-2106x DSPs:

One of the interrupt dispatchers supplied with VisualDSP++, `interrupts()`, relies on the use of the alternate register set. Therefore, it is not possible to service another interrupt while the current interrupt is being serviced. To ensure restriction, the interrupt enable bit (`IRPTEN`) is cleared by the `interrupts()` dispatcher while it is servicing an interrupt, and is then restored at the end.

Under certain, unusual circumstances on the ADSP-2106x processors, the interrupt enable bit can be set while the `interrupts()` dispatcher is being executed. A nested interrupt can result, possibly causing data corruption or other problems in user code.

The problem occurs when a lower priority interrupt (LPI) occurs and then, immediately afterwards, a higher priority interrupt (HPI) occurs. The problem only appears if the LPI is being serviced using `interrupts()`.

math.h Header File

A number of double-precision math functions have been implemented using only single-precision arithmetic. They are therefore relatively fast, but conversely do not have the precision expected. This issue affects the

64-bit versions of the following functions and has been addressed in VisualDSP++ 3.5.

acos	cos	rsqrt	tan
asin	cosh	sin	tanh
atan2	cot	sinh	
atan	pow	sqrt	

The header file now defines `long double` versions of the math functions. The names of these functions are the same as the equivalent `double` function with `d` appended to their name. For example, the header file contains the following prototypes for the sine function:

```
float sinf (float x);
double sin (double x);
long double sind (long double x);
```

The `long double` functions are independent of the size of type `double`.

In the C++ standard, the header files `math.h` and `cmath` are defined to contain overloaded versions of the `double` functions for the data types `float` and `long double`. This functionality was not supported in previous versions of VisualDSP++, but is now available in VisualDSP++ 3.5. Users should therefore be aware that the behavior of C++ programs may be affected if they pass arguments other than type `double` to the functions defined in the `math.h` and `cmath` header files.

stdio.h Header File

The I/O library, `libio.dlb`, which is associated with the header file `stdio.h` has been enhanced with new features. These features are described in Chapter 3, *New Features and Enhancements*. Applications that link with this library may notice an increase in code size. Applications that do not require some of the new functionality may link with the switch `-flags-link -MD__LIBIO_LITE`. The switch links the application with a

Changes to SHARC Compiler

version of the I/O library that does not support the ability to register alternative device drivers and does not support the %a conversion specifier in `printf`. Linking with this switch results in a smaller executable.

DSP Header Files

In previous releases of VisualDSP++, the header files associated with the DSP run-time library define functions that operate on the `float` data type. Versions of these functions are now available for some of these header files that also support the `double` and `long double` data types. These header files are:

```
complex.h  
cvector.h  
matrix.h  
stats.h  
vector.h
```

The names of the `long double` functions end with `d`, the names of the `float` functions end with `f`, while the names of the `double` functions have no suffix. As an example, the header file `stats.h` contains the following prototypes for the mean function:

```
float meanf (float x[], int n);  
double mean (double x[], int n);  
long double meand (long double x[], int n);
```

This is at variance with previous releases of VisualDSP++ in which the names of DSP run-time library functions with no suffix are defined to operate on data of type `float`. This issue affects those functions that are defined in the `cvector.h`, `stats.h`, and `vector.h` header files only. Applications that use these functions should not notice any difference, provided that they are not compiled with the `-double-size-64` switch. However, any application compiled with this switch should be modified to call the appropriate version of the function defined in `cvector.h`, `stats.h`, and `vector.h`.

fft_mag Function

The `fft_mag` function in the ADSP-211xx DSP run-time library used the incorrect formula to calculate the normalized power spectrum for an FFT. The function has been replaced by two new functions, `cfft_mag` and `rfft_mag`, that compute a normalized power spectrum from the output signal generated by `cfftN` and `rfftN` library functions, respectively. The function `fft_mag` is still supported for backwards compatibility and is equivalent to the `rfft_mag` function.

Miscellaneous Changes

Interprocedural Analysis (IPA)

The IPA optimizer, `ipa.exe`, has been entirely redesigned for VisualDSP++ 3.5. Although there are no user-visible changes, you should be aware that there are likely to be significant changes to the assembly code generated by an IPA-optimized function. You should also ensure that full rebuilds are done to clean out any residual `.opa` or `.ipa` files.

In some cases, the VisualDSP++ 3.5 compiler may produce slower code than the VDSP++ 3.0 compiler when optimization is enabled but IPA is disabled. For this reason, we recommend enabling IPA to obtain maximum performance from the compiler.

Others

- The compiler reorders functions in assembly files (`.s`). Functions do not necessarily appear in the assembly file in the same order in which they are defined in the source file.
- The USTAT registers are now scratch registers. If you are using a specific USTAT register in your application and you are also claiming it in an `gnu asm` statement using the “`u`” constraint, then you should reserve the USTAT register with the `-reserve` compiler switch.

Changes to TigerSHARC Compiler

This section summarizes changes to the TigerSHARC compiler (`ccts.exe`) environment since VisualDSP++ 3.0. As with any major upgrade, some of the improvements have changed how the compiler is invoked. The compiler's optimizing technology has undergone extensive revision for VisualDSP++ 3.5, but most of these changes are purely internal.

In general, the changes can be grouped into:

- [“TigerSHARC Command-Line Switches”](#) (some have been removed, some renamed)
- [“Linker Description Files”](#) (some minor changes to support new features)
- [“Miscellaneous Changes”](#) (other compiler changes)

TigerSHARC Command-Line Switches

Some command-line switches have been removed, as they are no longer appropriate and enable archaic language dialects no longer supported by the compiler. Other switches have been removed because they historically supported facilities useful for UNIX-targeted compilers and became inappropriate for compilers targeted at signal processing and embedded platforms.

-TS101|201

These switches have been deprecated in favor of `-proc ADSP-TS101`, `-proc ADSP-TS201`, `-proc ADSP-TS202`, or `-proc ADSP-TS203`.

-alttok

The `-alttok` switch, which directs the compiler to allow digraph sequences in C and C++ source files, is no longer on by default as it was in previous releases.

-analog is renamed to -c89

The `-analog` switch is renamed to `-c89` to reference a particular revision of the ANSI C Standard.

-cirbuf

The old `-cirbuf` switch has been renamed to `-force-cirbuf` to better indicate that it is causing the creation of circular buffers that would not otherwise happen. The `-no-cirbuf` switch exists to prevent the automatic creation of circular buffers. Explicit circular buffer usage, via the corresponding intrinsic functions, is still honored.

-inline, -no-inline (deprecated switches)

These switches are no longer supported because the inliner is integrated into the optimizer. When optimization is on, the inliner operates. When optimization is off, the inliner is disabled. See also `-Oa` (auto-inlining).

-no-alttok (now default)

The `-no-alttok` switch directed the compiler not to accept alternative operator keywords and digraph sequences in the source files. In VisualDSP++ 3.5, this directive occurs by default.

-xml

This switch specified that map files generated by the linker should be in XML format. In VisualDSP++ 3.5, the linker always generates XML-format map files, so the switch has been removed. The `-xml` switch is still accepted but has no effect.

Other switches

The following switches have been removed because they correspond to language options no longer supported in VisualDSP++ 3.5.

`-traditional`, `-dollar`, `-no-dollar`, `-J`, `-force`, `-installall`,
`-installlocal`, `-instantused`, `-suppress`, `-tpautooff`.

The following switches have been removed because they were used to select between two compilation modes; only the default mode is available in VisualDSP++ 3.5.

Changes to TigerSHARC Compiler

`-bool`, `-explicit`, `-namespace`, `-newforinit`, `-newvec`, `-std`, `-typename`, and `-wchar`. These switches now are always on.

The `-no-bool`, `-no-explicit`, `-no-namespace`, `-trdforinit`, `-no-newvec`, `-no-std`, and `-no-wchar` switches are no longer available because their corresponding elections are not supported in VisualDSP++ 3.5.

Linker Description Files

Default LDFs Support `-no-std-libs`

The compiler has a `-no-std-lib` switch, which directs the compiler not to search the standard library directory for libraries to resolve symbols. To achieve this objective, define the linker preprocessing macro `__NO_STD_LIB`, which guards the `SEARCH_DIR` linker directive. Without this guarding macro, the LDF still directs the linker to search the standard directory.

Default LDFs include `.meminit`

The memory initializer uses a special `“.meminit”` section to determine placement of initialization tables.

C++ Exception Support

The compiler's C++ exception support relies on a support library (`libx*.dll`) and exceptions-enabled versions of the run-time libraries (named with an “x” suffix). In addition, the exceptions support requires several new input sections (`.frt`, `.edt`, `.cht`, `.gdt`) that must be mapped to data areas.

Workaround Libraries

There are versions of the run-time libraries that contain anomaly workarounds for the ADSP-TS20x processors. These workaround versions of the libraries are selected if the `__WORKAROUNDS_ENABLED` macro is defined.

Start-Up Code and Libraries

Changes to the run-time library cause a small number of changes to the start-up code in `ts_hdr.asm`. The start-up code now plants calls to the memory initialization support routine. A call is also made to the program argument support routine.

In the C++ standard, the header files `math.h` and `cmath` are defined to contain overloaded versions of the `double` functions for the data types `float` and `long double`. This functionality was not supported in previous versions of VisualDSP++, but is now available in VisualDSP++ 3.5. Users should therefore be aware that the behavior of C++ programs may be affected if they pass arguments other than type `double` to the functions defined in the `math.h` and `cmath` header files.

Miscellaneous Changes

Interprocedural Analysis (IPA)

The IPA optimizer, `ipa.exe`, has been entirely redesigned for VisualDSP++ 3.5. Although there are no user-visible changes, you should be aware that there are likely to be significant changes to the assembly code generated by an IPA-optimized function. You should also ensure that full rebuilds are done to clean out any residual `.opa` or `.ipa` files.

In some cases, the VisualDSP++ 3.5 compiler may produce slower code than the VDSP++ 3.0 compiler when optimization is enabled but IPA is disabled. For this reason, we recommend enabling IPA to obtain maximum performance from the compiler.

Others

The compiler reorders functions in assembly files (`.s`). Functions do not necessarily appear in the assembly file in the same order in which they are defined in the source file.

Generating Map File in .XML Format

Prior to VisualDSP++ 3.5 release, it could be cumbersome to parse text version of the map file. Any small change to the format on the map file could possibly require changes to a tool that parses the map file.

Starting with VisualDSP++ 3.5 release, the linker generates the map file in XML format *only*. This change was done to standardize the map file format and to make the post-analysis process much easier.

Opening an .xml map file in a Web browser provides an organized view of the map file. By using hyperlinks, it becomes easy to quickly find any relevant information. Since the format of .xml files can be extended between VisualDSP++ releases, the map file is dependent on particular installation of VisualDSP++. Thus, the .xml map file can be used only on the machine on which it was generated. In order to view the map file on a different machine, transform the file to HTML format using the “xmlmap2html.exe” command-line utility. The utility makes it possible to view the map on virtually any machine with any browser.

A small sample script demonstrates the ease of parsing and extracting information from the linker .xml map file.

You can find the MapFileDemo script in:

... \21k\Examples\Scripting Examples\ for ADSP-21xxx processors

... \TS\Examples\Scripting Examples\ for ADSP-TSxxx processors

Run the script using the following command:

```
> cscript MapFileDemo.js <mapfile.xml>
```

To run the script, you might need to install Windows script support. Check the following link:

<http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp>

3 NEW FEATURES AND ENHANCEMENTS

VisualDSP++ 3.5 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 VisualDSP++ 3.5.

The chapter contains:

- [“VisualDSP++ IDDE” on page 3-2](#)
- [“Assembler for TigerSHARC Processors” on page 3-6](#)
- [“Assembler for SHARC Processors” on page 3-10](#)
- [“Compiler and Library for TigerSHARC Processors” on page 3-15](#)
- [“Compiler and Library for SHARC Processors” on page 3-28](#)
- [“Linker and Utilities” on page 3-44](#)
- [“Loaders and Splitter” on page 3-59](#)
- [“VCSE” on page 3-61](#)
- [“VDK” on page 3-63](#)
- [“Simulators for TS101 and TS20x Processors” on page 3-64](#)
- [“Object Protection” on page 3-64](#)
- [“Documentation Changes” on page 3-64](#)

VisualDSP++ IDDE

The VisualDSP++ 3.5 Integrated Development and Debugging Environment (IDDE) introduces:

- “New Processor Support” on page 3-3
- “Multiple Project Support” on page 3-3
- “XML Project File Format” on page 3-3
- “Project Migration” on page 3-4
- “License Management” on page 3-4
- “Data Streaming and Logging” on page 3-4
- “Profile-Guided Optimization” on page 3-4
- “Integrated Source Code Control” on page 3-4
- “Automation Aware Scripting Engine” on page 3-5
- “Address Bar in Memory and Disassembly Windows” on page 3-5
- “Menus with Icons” on page 3-5

For more information about VisualDSP++ IDDE, refer to the *VisualDSP++ 3.5 User’s Guide for 32-Bit Processors* and online Help.

New Processor Support

The following new processors are supported by VisualDSP++ 3.5:

SHARC Processors

ADSP-21261	ADSP-21262	ADSP-21266	ADSP-21267
ADSP-21363	ADSP-21364	ADSP-21365	

TigerSHARC Processors

ADSP-TS202	ADSP-TS203
------------	------------

For details, refer to Hardware References and data sheets of appropriate target processors.

Multiple Project Support

VisualDSP++ provides the ability to switch among multiple open projects in the same IDDE session. The **Project** window displays active projects.

XML Project File Format

The binary project file format used in previous releases of VisualDSP++ has been replaced with a new text-based XML format. The XML format has several benefits:

- Forward and backward compatibility in successive releases
- Better version control and comparison in source code control environments
- Better readability

Project Migration

Projects migrated to VisualDSP++ 3.5 cannot be opened in previous versions. However, a backup copy of the project, automatically made before conversion, can be opened for backwards compatibility.

License Management

License installation and validation has been integrated into the VisualDSP++ IDDE. Installing a FlexLM license server is still handled by the separate installation application. Two licensing options are available: *single-user* and *client*. A *server* license is required before you can install a client license.

Data Streaming and Logging

VisualDSP++ now offers the ability to stream from a target DSP without halting the DSP. The IDDE takes advantage of this capability in plot windows. If the target supports BTC, the plot window is updated without halting the target.

Profile-Guided Optimization

The VisualDSP++ IDDE includes facilities to run common Profile-Guided Optimization (PGO) scenarios simply and also provides a mechanism for advanced users who require more control over the profiling process via scripting. The technique relies on setting up and executing data sets to produce an optimized application. See [“Optimization Control” on page 3-19](#) for more information.

Integrated Source Code Control

VisualDSP++ provides integration between the IDDE and Integrated Source Code Control (SCC) applications (such as Visual SourceSafe,

PVCS Version Manager, and Concurrent Versions System (CVS)) installed on your machine through Microsoft Common Source Code Control (MCSCC) interface. You can conveniently access commonly-used SCC features from VisualDSP++ without leaving the IDDE. Application-specific and advanced SCC features not available from the IDDE must be run directly from the SCC applications.

Automation Aware Scripting Engine

VisualDSP++ includes a scripting engine that utilizes the Microsoft ActiveX script host framework. The engine allows you to use multiple scripting languages, such as VBScript, JavaScript and others, to access the VisualDSP++ Automation API.

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

Address Bar in Memory and Disassembly Windows

When enabled, **Disassembly** and **Memory** windows display an address bar. Use the address bar to navigate by address, symbol, or expression. The address bar maintains a most recently used history of visited locations.

Menus with Icons

Icons now appear beside menu commands that have equivalent toolbar buttons.

Assembler for TigerSHARC Processors

For TigerSHARC processors, the assembler's most notable new features and enhancements are:

- [“Assembler Macros” on page 3-6](#)
- [“VCSE Optimization Directives” on page 3-7](#)
- [“Assembler Command-Line Switch” on page 3-7](#)
- [“Preprocessor Macros” on page 3-8](#)
- [“Preprocessor Command-Line Switches” on page 3-8](#)
- [“Include Path Search Algorithm Matches Compiler” on page 3-9](#)

For more information, refer to the *VisualDSP++ 3.5 Assembler and Preprocessor Manual for TigerSHARC Processors* and online Help.

Assembler Macros

The new feature macros are:

<code>-D__ADSP_TS202__ =1</code>	Present when running <code>easmts -proc ADSP-TS202</code> with ADSP-TS202 processors
<code>-D__ADSP_TS203__ =1</code>	Present when running <code>easmts -proc ADSP-TS201</code> with ADSP-TS203 processors
<code>-D__ADSP_TS20x__ =1</code>	Present when running <code>easmts -proc ADSP-TS201</code> with ADSP-TS201 processors <code>easmts -proc ADSP-TS202</code> with ADSP-TS202 processors <code>easmts -proc ADSP-TS203</code> with ADSP-TS203 processors.

VCSE Optimization Directives

The new `.VCSE_` directives are the optimization directives for VCSE components. The `.VCSE_METHODCALL_START` and `.VCSE_METHODCALL_END` directives mark VCSE methods for linker code/data elimination. The linker is provided with the interface name and actual offset of the corresponding entry in the method table. The `.VCSE_RETURNS` directive is used for marking VCSE constant methods.

PAGEWIDTH Directive

The `.PAGEWIDTH expression` directive sets the page width of the listing file produced by the assembler. Depending on the setting of the `.LEFTMARGIN` directive, the page width setting is now at least equal to the `LEFTMARGIN` value plus 49. The setting cannot be less than 49. There is no upper limit. If `LEFTMARGIN = 0` and the `.PAGEWIDTH` value is not specified, the actual page width is set to any number over 49.

Assembler Command-Line Switch

The new assembler command-line switches are:

Switch Name	Purpose
<code>-no-source-dependency</code>	This switch directs the assembler not to print anything about dependency between the <code>.asm</code> source file and the <code>.doj</code> object file when outputting dependency information. This option must be used in conjunction with the <code>-M</code> or <code>-MM</code> options.

Assembler for TigerSHARC Processors

Switch Name	Purpose
-save-temps	This switch directs the assembler to retain intermediate files generated and normally removed as part of the assembly process.
-si-revision <i>version</i>	This switch defines the silicon revision for a particular processor. The parameter “<i>version</i>” represents a silicon revision of the processor specified by the -proc switch. The supported revisions are “0.0”, “0.1”, “1.0” and “none”. The assembler uses this information when dealing with issues which are specific to a particular silicon revision of a processor. If “none” is used, these issues are ignored. If the -si-revision option is not used, the assembler assumes the latest silicon revision known to it. The assembler sets the <code>__SILICON_REVISION__</code> macro to 0x0 for revision “0.0”, 0x1 for revision “0.1” and 0x100 for revision “1.0”. The assembler does not set the <code>__SILICON_REVISION__</code> macro for “none”.

Preprocessor Macros

The new preprocessor predefined macro, `__LASTSUFFIX__`, specifies the last value of suffix that was used to build preprocessor generated labels.

The new preprocessor feature macros are `__ADSPTS202__` and `__ADSPTS203__`.

Preprocessor Command-Line Switches

The following preprocessor command-line switches have been introduced in VisualDSP++ 3.5.

Switch Name	Description
-cstring	Enables the stringization operator and provides “C compiler” style preprocessor behavior

<code>-i</code>	Outputs only makefile dependencies for <code>include</code> files specified in double quotes
<code>-tokenize-dot</code>	Treats “.” (dot) as an operator when parsing identifiers
<code>-Uname</code>	Undefines a macro on the command line
<code>-version</code>	Displays version information for preprocessor
<code>-w</code>	Removes all preprocessor-generated warnings
<code>-Wnumber</code>	Suppresses any report of the specified warning
<code>-warn</code>	Prints warning messages (default)

Include Path Search Algorithm Matches Compiler

In VisualDSP++ 3.5, the `include` path semantics used by the linker/assembler preprocessor and the compiler for user and system header files are the same.

The `#include <file>` (system header file) search order is:

1. include path specified by the `-I` switch
2. `...\VisualDSP\processor\include` folders

The `#include "file"` (user header file) search order is:

1. local directory — the directory in which the source file resides
2. include path specified by the `-I` switch
3. `...\VisualDSP\processor\include` folders

When both `-I` and `-I-` appear on the command line, the system search path (`#include < >`) is modified in such a manner that search directories specified with the `-I` switch that appear before the directory specified with the `-I-` switch are ignored.

Assembler for SHARC Processors

For SHARC processors, the assembler's most notable new features and enhancements are:

- “New Assembler Directives” on page 3-10
- “Assembler Command-Line Switches” on page 3-11
- “Preprocessor Macros” on page 3-12
- “Preprocessor Command-Line Switches” on page 3-13
- “Include Path Search Algorithm Matches Compiler” on page 3-13

For more information, refer to the *VisualDSP++ 3.5 Assembler and Preprocessor Manual for SHARC Processors* and online Help.

New Assembler Directives

The new assembler directives are:

.TYPE, Change Default Symbol Type

The `.TYPE` directive directs the assembler to change the default symbol type of an object. This directive may appear in the compiler-generated assembly source file (`.S`).

VCSE Optimization Directives

The `.VCSE_` directives are the optimization directives for VCSE components. The `.VCSE_METHODCALL_START` and `.VCSE_METHODCALL_END` directives mark VCSE methods for linker code/data elimination. The linker is provided with the interface name and actual offset of the corresponding entry in the method table. The `.VCSE_RETURNS` directive is used for marking VCSE constant methods.

PAGEWIDTH Directive

The `.PAGEWIDTH expression` directive sets the page width of the listing file produced by the assembler. Depending on setting of the `.LEFTMARGIN` directive, the page width setting is now at least equal to the `LEFTMARGIN` value plus about 66. The setting cannot be less than 66. There is no upper limit. If `LEFTMARGIN = 0` and the `.PAGEWIDTH` value is not specified, the actual page width is set to any number over 66.

Assembler Command-Line Switches

The new assembler command-line switches are:

Switch Name	Purpose
<code>-no-source-dependency</code>	This switch directs the assembler not to print anything about dependency between the <code>.asm</code> source file and the <code>.doj</code> object file when outputting dependency information. This option must be used in conjunction with the <code>-M</code> or <code>-MM</code> options.
<code>-save-temps</code>	This switch directs the assembler to retain intermediate files generated and normally removed as part of the assembly process.
<code>-si-revision <i>version</i></code>	This switch defines the silicon revision for a particular processor. The parameter “<i>version</i>” represents a silicon revision of the processor specified by the <code>-proc</code> switch. The supported revisions are “0.0”, “0.1”, “1.0” and “none”. The assembler uses this information when dealing with issues which are specific to a particular silicon revision of a processor. If “none” is used, these issues are ignored. If the <code>-si-revision</code> option is not used, the assembler assumes the latest silicon revision known to it. The assembler sets the <code>__SILICON_REVISION__</code> macro to 0x0 for revision “0.0”, 0x1 for revision “0.1” and 0x100 for revision “1.0”. The assembler does not set the <code>__SILICON_REVISION__</code> macro for “none”.

Assembler Macros

The new assembler feature macros are:

<code>-D__ADSP21261__=1</code> <code>-D__2126x__=1</code>	Present when running <code>easm21K -proc ADSP-21261</code> with ADSP-21261 DSPs
<code>-D__ADSP21262__=1</code> <code>-D__2126x__=1</code>	Present when running <code>easm21K -proc ADSP-21262</code> with ADSP-21262 DSPs
<code>-D__ADSP21266__=1</code> <code>-D__2126x__=1</code>	Present when running <code>easm21K -proc ADSP-21266</code> with ADSP-21266 DSPs
<code>-D__ADSP21267__=1</code> <code>-D__2126x__=1</code>	Present when running <code>easm21K -proc ADSP-21267</code> with ADSP-21267 DSPs
<code>-D__ADSP21363__=1</code> <code>-D__2136x__=1</code>	Present when running <code>easm21K -proc ADSP-21363</code> with ADSP-21363 DSPs
<code>-D__ADSP21364__=1</code> <code>-D__2136x__=1</code>	Present when running <code>easm21K -proc ADSP-21364</code> with ADSP-21364 DSPs
<code>-D__ADSP21365__=1</code> <code>-D__2136x__=1</code>	Present when running <code>easm21K -proc ADSP-21365</code> with ADSP-21365 DSPs

For the `.IMPORT` headers, the assembler calls the compiler driver with the appropriate processor option and the compiler sets the machine constants accordingly (and defines `-D__LANGUAGE_C=1`). This macro is present when used for C compiler calls to specify headers. It replaces `-D__LANGUAGE_ASM`.

Preprocessor Macros

The new preprocessor predefined macro `__LASTSUFFIX__` specifies the last value of suffix used to build preprocessor generated labels.

For the list of the new feature macros, see [“Assembler Macros”](#) above,

Preprocessor Command-Line Switches

The following preprocessor command-line switches have been introduced in VisualDSP++ 3.5.

Switch Name	Description
-cstring	Enables the stringization operator and provides “C compiler” style preprocessor behavior
-i	Outputs only makefile dependencies for <code>include</code> files specified in double quotes
-tokenize-dot	Treats “.” (dot) as an operator when parsing identifiers
-Uname	Undefines a macro on the command line
-version	Displays version information for preprocessor
-w	Removes all preprocessor-generated warnings
-Wnumber	Suppresses any report of the specified warning
-warn	Prints warning messages (default)

Include Path Search Algorithm Matches Compiler

In VisualDSP++ 3.5, the `include` path semantics used by the linker/assembler preprocessor and the compiler for user and system header files are the same.

The `#include <file>` (system header file) search order is:

1. include path specified by the `-I` switch
2. ... \VisualDSP\processor\include folders

Assembler for SHARC Processors

The `#include` “file” (user header file) search order is:

1. local directory — the directory in which the source file resides
2. include path specified by the `-I` switch
3. ...VisualDSP\processor\include folders

When both `-I` and `-I-` appear on the command line, the system search path (`#include < >`) is modified in such a manner that search directories specified with the `-I` switch that appear before the directory specified with the `-I-` switch are ignored.

Compiler and Library for TigerSHARC Processors

For TigerSHARC processors, the most notable new features and enhancements of the C/C++ compiler are:

- “File Extensions” on page 3-16
- “Compiler Command-Line Switches” on page 3-16
- “Optimization Control” on page 3-19
- “Compiler Built-In Functions” on page 3-19
- “Pragmas” on page 3-22
- “Predefined Compiler Macros” on page 3-23
- “GCC Compatibility Extensions” on page 3-24
- “File I/O Support” on page 3-25
- “C/C++ Libraries and Start-Up Files” on page 3-25
- “C Library Functions” on page 3-26
- “DSP Run-Time Library” on page 3-27

For detailed information on these features, refer to the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for TigerSHARC Processors* and online Help.

File Extensions

The compiler supports new file extensions.

File Extension	Description
.cc	C++ source code
.idl	Interface definition language files for VCSE
.ipa, .opa	Interprocedural analysis files used internally by the compiler when performing interprocedural analysis.
.xml	Processor memory map file output

Compiler Command-Line Switches

This section summarizes C/C++ compiler command-line switches introduced or enhanced in VisualDSP++ 3.5. [Table 3-1](#) through [Table 3-3](#) list and briefly describe each switch:

- [Table 3-1, “C or C++ Mode Selection Switches” on page 3-16](#)
- [Table 3-2, “C/C++ Compiler Common Switches” on page 3-16](#)
- [Table 3-3, “C++ Mode Compiler Switches” on page 3-18.](#)

Table 3-1. C or C++ Mode Selection Switches

Switch Name	Description
-c89	Supports programs that conform to the ISO/IEC 9899:1990 standard

Table 3-2. C/C++ Compiler Common Switches

Switch Name	Description
-bss	Causes the compiler to place global zero-initialized data into a separate BSS-style section.
-ED	Preprocesses and sends all output to a file.

Table 3-2. C/C++ Compiler Common Switches (Cont'd)

Switch Name	Description
-force-circbuf	Treats array references of the form <code>array[i%n]</code> as circular buffer operations.
-fp-associative	Treats floating point multiply and addition as an associative.
-i	Outputs only header details or makefile dependencies for <code>include</code> files specified in double quotes.
-MD	Generates <code>make</code> rule, compiles, and prints to a file.
-Mo <i>filename</i>	Writes dependency information to <i>filename</i> . This switch is used in conjunction with the <code>-ED</code> or <code>-MD</code> options.
-mem	Enables memory initialization.
-no-alttok	Does not allow alternative keywords and sequences in sources.
-no-annotate	Disables the annotation of assembly files.
-no-bss	Causes the compiler to group global zero-initialized data into the same section as global data with non-zero initializers. Set by default.
-no-circbuf	Disables the automatic generation of circular buffering code.
-no-const-strings	Indicates that strings literals should not be qualified as <code>const</code> .
-no-fp-associative	Does not treat floating point multiply and addition as an associative.
-no-mem	Disables memory initialization.
-no-saturation	Causes the compiler not to introduce saturation semantics when optimizing expressions.
-no-std-ass	Disables any predefined assertions and system-specific macro definitions.
-Oa	Enables automatic function inlining.

Compiler and Library for TigerSHARC Processors

Table 3-2. C/C++ Compiler Common Switches (Cont'd)

Switch Name	Description
-pguide	Add instrumentation for the gathering of a profile as the first stage of performing profile-guided optimization.
-signed-bitfield	Makes the default type for <code>int</code> bitfields signed.
-si-revision <i>version</i>	Specifies a silicon revision of the specified processor. The default setting is the latest silicon revision.
-unsigned-bitfield	Makes the default type for plain <code>int</code> bitfields unsigned.
-vcse-add <filename>	If the <code>-vcse-enforce</code> switch has been supplied, <filename> is passed to the <code>vcse_enforce</code> utility as the argument for the <code>-add</code> switch.
--vcse-cname <component name>	If the <code>-vcse-enforce</code> switch has been supplied, <component name> is passed to the <code>vcse_enforce</code> utility as the argument for the <code>-cname</code> switch.
-vcse-enforce	If the library builder is invoked, <code>ccts</code> invokes the <code>vcse_enforce</code> utility to process the generated library.
-vcse-names <filename>	If the <code>-vcse-enforce</code> switch has been supplied, <filename> is passed to the <code>vcse_enforce</code> utility as the argument for the <code>-names</code> switch.

Table 3-3. C++ Mode Compiler Switches

Switch Name	Description
-anach	Supports some language features (anachronisms) that are prohibited by the C++ standard but are still in common use.
-eh	Enables exception handling.
-no-anach	Disallows the use of anachronisms that are prohibited by the C++ standard.
-no-eh	Disables exception-handling.
-no-rtti	Disables run-time type information.
-rtti	Enables run-time type information.

Optimization Control

The following list identifies several new optimization levels. Refer to Chapter 2, *Achieving Optimal Performance from C/C++ Source Code*, of the compiler manual for detailed information on how to obtain maximal code performance from the compiler.

The new and enhanced optimization features are:

- **Profile-Guided Optimizations**

The compiler performs advanced aggressive optimizations using profiler statistics generated from running the application using representative training data. PGO can be used in conjunction with interprocedural optimizations (IPA) and automatic inlining. The PGO operation is handled via a new PGO submenu added to the top-level **Tools** menu: **Tools -> PGO -> Manage Data Sets**.

- **Automatic Inlining**

The compiler automatically inlines C/C++ functions which are not necessarily declared as inline in the source code. The compiler determines when the inlining will reduce execution time. How aggressively the compiler performs automatic inlining is controlled by the `-Ov` switch. Automatic inlining, enabled using the `-Oa` switch, additionally enables Procedural Optimizations (`-O`).

Program Argument Support

Support is included in this release for passing arguments to the main routine (`ARGV/ARGC` support) by means of setting a specific variable. Refer to the VisualDSP++ 3.5 C/C++ Compiler and Library Manual for details.

Compiler Built-In Functions

The compiler supports intrinsic (built-in) functions that enable you to make efficient use of hardware resources.

builtins.h Header File

The `builtins.h` header file provides user-visible prototypes and C reference code for the built-in functions. This file can enable compilation and execution of code that makes use of the built-in functions.

When using a compiler other than the one provided with VisualDSP++ for TigerSHARC processors, make sure your compiler supports inline functions as the C reference versions of the built-ins use the `inline` keyword. These implementations of the built-in functions also require support for 64-bit integers.

Optimization Guidance Built-in Functions

The `__builtin_aligned` intrinsic is an assertion that the first parameter points to an argument that is aligned on a multiple of the second argument.

```
void __builtin_aligned(const void *p, int align);
```

The second argument is in units of `char` size, which is 8-bit bytes in byte-addressing mode and 32-bit words in word-addressing mode. Ideally all data processed in the first iteration of the loop should be aligned on a four-word boundary.

Data Alignment Buffer (DAB) Built-in Functions

The DAB built-in functions provide access to the Data Alignment Buffer functionality. Two sets of built-in functions are provided. The first set covers 32-bit data items, and the built-ins are the same whether byte-addressing mode is enabled (with the `-char-size-8|32` switch) or not. The second set covers 16-bit data items, and the details for using these functions depend on whether byte-addressing mode is enabled or not.

Circular Buffer Data Alignment Buffer (DAB) Built-in Functions

The circular buffer DAB intrinsics are like the DAB intrinsics with extra “*base*” and “*length*” parameters. The automatic post-increments now become post-increment with wrap-around according to the circular buffering. Unoptimized, the code generated performs two DAB accesses (the prime and the read) as well as initialization and reset of the length and base registers for each DAB intrinsic. When optimization is turned on, the unnecessary setting of length and base registers and the DAB prime operation is removed or hoisted out of loops to generate the optimal code.

C++ Exception Handling

When in C++ mode, the compiler supports run-time type information (RTTI) and exception handling as defined by the ISO/IEC 14882:1998 standard and modified by Technical Corrigendum 1. Exceptions are enabled using the `-eh` switch, and the RTTI is enabled using the `-rtti` switch.

Exceptions also require modifications to the Linker Description Files; these modifications link against versions of the C++ run-time library that have been built with exceptions support enabled, and link the exception-handling library. Several new data sections must be mapped into the `.dxe` file (`.fnt`, `.edt`, `.cht`, and `.gdt`). See the default LDFs included with the release for example modifications.

Pragmas

The VisualDSP++ 3.5 C/C++ compiler supports a number of new pragmas. Pragmas are implementation-specific directives that modify the compiler's behavior. The new and enhanced pragmas are described briefly below.

Pragma	Function
<code>#pragma all_aligned</code>	Asserts that all pointers are initially aligned on the most desirable boundary; applies to the subsequent loop.
<code>#pragma no_vectorization</code>	Turns off all vectorization for the loop on which it is specified.
<code>#pragma different_banks</code>	Allows the compiler to assume that groups of memory accesses based on different pointers within a loop reside in different memory banks.
<code>#pragma interrupt</code>	Defines a non-reentrant interrupt handler (one which cannot be interrupted by a subsequent interrupt).
<code>#pragma interrupt_reentrant</code>	Defines a reentrant interrupt handler (one which can be interrupted by higher-priority interrupts).
<code>#pragma weak_entry</code>	Causes the compiler to generate the function or variable definition with weak linkage.
<code>#pragma pure</code>	Tells the compiler that the function does not write to any global variables, and does not read or write any volatile variables.
<code>#pragma const</code>	This is a more restrictive form of the <code>pure</code> pragma. It tells the compiler that the function does not read from or write to global variables and does not read or write volatile variables. The result of the function is therefore a function of its parameters.
<code>#pragma regs_clobbered <i>string</i></code>	Used with a function declaration or definition to specify which registers are modified (or clobbered) by that function.
<code>#pragma instantiate <i>instance</i></code>	Requests the compiler to instantiate <i>instance</i> in the current compilation.

Pragma	Function
<code>#pragma do_not_instantiate <i>instance</i></code>	Directs the compiler not to instantiate <i>instance</i> in the current compilation.
<code>#pragma can_instantiate <i>instance</i></code>	Tells the compiler that if <i>instance</i> is required anywhere in the program, it should be instantiated in this compilation. This pragma has the same effect as <code>#pragma instantiate</code> .
<code>#pragma once</code>	Appears at the beginning of a header file and tells the compiler that the header is written in such a way that including it several times has the same effect as including it once.

Predefined Compiler Macros

The new predefined compiler macros are:

- The `__ADSPTS202__` macro is defined as 1 when you compile with the `-proc ADSP-TS202` command-line switch.
- The `__ADSPTS203__` macro is defined as 1 when you compile with the `-proc ADSP-TS203` command-line switch.
- The `__ADSPTS20x__` macro is defined as 1 when you compile with either the `-proc ADSP-TS201`, `-proc ADSP-TS202`, or `-proc ADSP-TS203` command-line switch.
- The `__SILICON_REVISION__` macro is defined when the `-si-revision` switch is specified with a value other than `none`. The value it is defined to is the major revision number shifted left by 8 logically OR'd against the minor revision number. An example would be that if `"-si-revision 2.1"` were to be specified, then the value of the define for the `__SILICON_REVISION__` macro would be `0x201`.
- The `__VERSION__` macro: The preprocessor expands this macro into a string constant containing the current compiler version.

- The `__WORKAROUNDS_ENABLED` macro is defined as 1 if any hardware workarounds are implemented by the compiler. This macro is set if the `-si-revision` switch option has a value other than “none” or if any specific workaround is selected by means of the `-workaround` compiler command-line switch.

GCC Compatibility Extensions

The compiler provides compatibility with the C dialect accepted by version 3.2 of the GNU C Compiler. Many of these extensions are available in the C99 ANSI Standard. A brief description of the following extensions is included in the compiler manual:

- Statement expressions
- Type reference support keyword (`typeof`)
- GCC generalized `Lvalues`
- Conditional expressions with missing operands
- Hexadecimal floating-point numbers
- Arithmetic on pointers to `void` and pointers to functions
- Cast to `union`
- Ranges in `case` labels
- Declarations mixed with code
- Zero-length arrays
- Variable argument macros
- Line breaks in string literals
- Escape character constant
- Alignment inquiry keyword (`__alignof__`)
- Keyword for specifying names in generated assembler (`asm`)
- Function, variable and type attribute keyword (`__attribute__`)

File I/O Support

In earlier VisualDSP++ releases, the implementation of the functions defined in the header file `stdio.h` was based on a device driver, known as `primIO`, provided by the VisualDSP++ simulator and EZ-KIT Lite systems. This device driver, however, only provides access to the host file system.

VisualDSP++ 3.5 permits alternative device drivers to be registered and used through the normal `stdio` functions. This feature enables the routines to interact with a device other than the host file system. By default, the `stdio` functions will continue to use the device driver provided by the VisualDSP++ simulator and EZ-KIT Lite systems, and users should not notice any difference in functionality. Full details of the new extensible driver mechanism are available in the C/C++ compiler manual.

C/C++ Libraries and Start-Up Files

This release of VisualDSP++ contains a file for memory initialization support and several new variants of the C/C++ run-time libraries and start-up files. These new binaries files are summarized in the following table.

Table 3-4. C and C++ Library Files

Library – Startup Files	Description
<code>meminit_TS101.doj</code> <code>meminit_TS20x.doj</code>	Memory initialization support
<code>*_w.dlb</code> <code>*_w.doj</code>	Binary files that incorporate all available workarounds for known ADSP-TS20x hardware errata
<code>*_x_*.dlb</code>	Run-time libraries with C++ exception handling enabled

The start-up routines now call the memory initializer support routine and set up the arguments for the main routine.

C Library Functions

The C run-time library has been extended with the addition of some new functions and enhanced functionality.

The formatted input/output functions defined in `stdio.h` (such as, `printf`, `scanf`, `fprintf`, . . .) now support the `%a` conversion specifier. The `%a` specifier is similar both in form and meaning to the `%e` specifier, except that the `%e` specifier is used to input and output decimal floating-point numbers, while the `%a` specifier is used to input and output hexadecimal floating-point numbers. Support for the `ll` length modifier has also been added and may be used with the `%d`, `%i`, `%o`, `%u`, and `%x` conversion specifiers to input and output data of type `long long int`.

Additional functions defined in the `stdio.h` header file are supported in the VisualDSP++ 3.5 release. These functions are:

`fgetpos`, `fseek`, `fsetpos`, `ftell`, `remove`, `rename`, `rewind`

 The C standard `stdio.h` functions `tmpfile` and `tmpnam` are not supported in this release.

The library functions `atof`, `atoff`, `atold`, `strtof`, `strtod`, and `strtold` defined in `stdlib.h` have been modified to support hexadecimal floating-point numbers. The header file also defines the new functions `atoll`, `llabs`, `strtoll`, and `strtoull` for the `long long int` data type. The function documentation has been updated to reflect this new functionality.

DSP Run-Time Library

New functions have been added to the DSP run-time library. These functions are identified in the following table.

Library Function	Description
<code>alog</code>	Calculates the natural (base e) anti-log of its argument
<code>alog10</code>	Calculates the base 10 anti-log of its argument
<code>cartesian</code>	Transforms a complex number from Cartesian notation to polar notation
<code>cfft_mag</code>	Computes the power spectrum of a complex FFT
<code>rfft_mag</code>	Computes the power spectrum of a real FFT
<code>rfftf_mag</code>	Computes the power spectrum generated by the <code>rfftf</code> function



The new functions are fully documented in the compiler manual.

Compiler and Library for SHARC Processors

For SHARC processors, the most notable new features and enhancements of the C/C++ compiler are:

- “File Extensions” on page 3-29
- “Compiler Command-Line Switches” on page 3-29
- “Optimization Control” on page 3-32
- “Pragmas” on page 3-33
- “GCC Compatibility Extensions” on page 3-35
- “Predefined Compiler Macro” on page 3-36
- “seg_int_code Memory/Section Name” on page 3-37
- “Support for argv/argc” on page 3-38
- “File I/O Support” on page 3-38
- “Run-Time Libraries and Start-Up Files” on page 3-39
- “C Run-Time Library Functions” on page 3-39
- “DSP Run-Time Library Functions” on page 3-40

For more information about these features, refer to the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for SHARC Processors* and online Help.

File Extensions

The compiler supports new file extensions:

File Extension	Description
.cc	C++ source code
.ipa, .opa	IPA files used internally by the .pch header file
.pch	Precompiled header file
.xml	Processor memory map file output

Compiler Command-Line Switches

This section summarizes C/C++ compiler command-line switches introduced or enhanced in VisualDSP++ 3.5.

- [Table 3-5, “C or C++ Mode Selection Switches” on page 3-29](#)
- [Table 3-6, “C/C++ Compiler Common Switches” on page 3-29](#)
- [Table 3-7, “C++ Mode Compiler Switches” on page 3-31](#)

Table 3-5. C or C++ Mode Selection Switches

Switch Name	Description
-c89	Supports programs that conform to the ISO/IEC 9899:1990 standard.

Table 3-6. C/C++ Compiler Common Switches

Switch Name	Description
-bss	Causes the compiler to put global zero-initialized data into a separate BSS-style section.
-const-read-write	Specifies that data accessed via a pointer to <code>const</code> data may be modified elsewhere.
-ED	Preprocesses and sends all output to a file.

Compiler and Library for SHARC Processors

Table 3-6. C/C++ Compiler Common Switches (Cont'd)

Switch Name	Description
-force-circbuf	Treats array references of the form <code>array[i%n]</code> as circular buffer operations.
-fp-associative	Treats floating-point multiplication and addition as associative.
-i	Outputs only header details or makefile dependencies for <code>include</code> files specified in double quotes.
-ipa	Specifies that interprocedural analysis should be performed for optimization between translation units.
-MD	Generates <code>make</code> rule, compiles, and prints to a file.
-Mo <i>filename</i>	Writes dependency information to <i>filename</i> . This switch is used in conjunction with the <code>-ED</code> or <code>-MD</code> options.
-no-annotate	Disables the annotation of assembly files.
-no-bss	Causes the compiler to group global zero-initialized data into the same section as global data with non-zero initializers.
-no-circbuf	Disables the automatic generation of circular buffering code.
-no-fp-associative	Does not treat floating-point multiply and addition as an associative.
-no-saturation	Causes the compiler not to introduce saturation semantics when optimizing expressions.
-no-std-ass	Disables any predefined assertions and system-specific macro definitions.
-Oa	Enables automatic function inlining.
-Ov <i>num</i>	Controls speed versus size optimizations.
-path- <i>tool</i>	Enhanced. Uses the specified directory as the location of the specified compilation <i>tool</i> (assembler, compiler, library builder, linker, or memory initializer).
-pguide	Adds instrumentation for the gathering of a profile as the first stage of performing profile-guided optimization.

Table 3-6. C/C++ Compiler Common Switches (Cont'd)

Switch Name	Description
<code>-restrict-hardware-loops <maximum></code>	Restrict the number of levels of loop nesting used by the compiler.
<code>-signed-bitfield</code>	Makes the default type for <code>int</code> bitfields signed.
<code>-signed-char</code>	Makes the default type for <code>char</code> signed.
<code>-si-revision <i>version</i></code>	Specifies a silicon revision of the specified processor. The default setting is the latest silicon revision.
<code>-vcse-add <filename></code>	If the <code>-vcse-enforce</code> switch has been supplied, <code><filename></code> is passed to the <code>vcse_enforce</code> utility as the argument for the <code>-add</code> switch.
<code>-vcse-cname <component name></code>	If the <code>-vcse-enforce</code> switch has been supplied, <code><component name></code> is passed to the <code>vcse_enforce</code> utility as the argument for the <code>-cname</code> switch.
<code>-vcse-enforce</code>	If the library builder is invoked, <code>cc21k</code> invokes the <code>vcse_enforce</code> utility to process the generated library.
<code>-vcse-names <filename></code>	If the <code>-vcse-enforce</code> switch has been supplied, <code><filename></code> is passed to the <code>vcse_enforce</code> utility as the argument for the <code>-names</code> switch.
<code>-unsigned-bitfield</code>	Makes the default type for plain <code>int</code> bitfields unsigned.
<code>-workaround <workaround></code>	Enables code generator workaround for specific hardware errata.

Table 3-7. C++ Mode Compiler Switches

Switch Name	Description
<code>-anach</code>	Supports some language features (anachronisms) that are prohibited by the C++ standard but still in common use.
<code>-eh</code>	Enables exception handling.
<code>-no-anach</code>	Disallows the use of anachronisms that are prohibited by the C++ standard.
<code>-no-eh</code>	Disables exception handling.

Table 3-7. C++ Mode Compiler Switches (Cont'd)

Switch Name	Description
<code>-no-rtti</code>	Disables run-time type information.
<code>-rtti</code>	Enables run-time type information.

Optimization Control

The following list identifies several new optimization levels. Refer to Chapter 2, *Achieving Optimal Performance from C/C++ Source Code*, of the compiler manual for detailed information on how to obtain maximal code performance from the compiler. The new and enhanced optimization features are:

- **Automatic Inlining**
The compiler automatically inlines C/C++ functions which are not necessarily declared as inline in the source code. The compiler determines when the inlining will reduce execution time. How aggressively the compiler performs automatic inlining is controlled using the `-Ov` switch. Automatic inlining, enabled using the `-Oa` switch, additionally enables procedural optimizations (`-O`).
- **Profile-Guided Optimizations (PGO)**
The compiler performs advanced aggressive optimizations using profiler statistics generated from running the application using representative training data. PGO can be used in conjunction with IPA and automatic inlining.
- **C++ Standard Exceptions Support**
When in C++ mode, the compiler supports run-time type information (RTTI) and exception handling as defined by the ISO/IEC 14882:1998 standard and modified by Technical Corrigendum 1. Exceptions are enabled using the `-eh` switch, and the RTTI is enabled using the `-rtti` switch.

Pragmas

The VisualDSP ++ 3.5 C/C++ compiler supports a number of new pragmas. Pragmas are implementation-specific directives that modify the compiler's behavior. The new and enhanced pragmas are described briefly in [Table 3-8](#).

Table 3-8. C/C++ Compiler Pragmas

Pragma	Function
<code>#pragma all_aligned</code>	Applies to the subsequent loop. This pragma tells that compiler that all pointer induction variables in the loop are initially aligned to the maximum permitted alignment of the processor architecture. For ADSP-2106x DSPs, it is word aligned; for ADSP-2116x/2126x/2136x DSPs, it is double-word aligned.
<code>#pragma vector_for</code>	Notifies the optimizer that it is safe to execute two iterations of the loop in parallel. The pragma does not force the compiler to vectorize the loop; the optimizer checks various properties of the loop and does not vectorize it if it believes it is unsafe.
<code>#pragma no_vectorization</code>	Used with ADSP-2116x/2126x/2136x DSPs. It ensures the compiler will not generate vectorized SIMD code for the loop on which it is specified.
<code>#pragma loop_count(min, max, modulo)</code>	Asserts that the loop will iterate at least <i>min</i> times, no more than <i>max</i> times, and a multiple of <i>modulo</i> times. This information enables the optimizer to omit loop guards and to decide whether the loop is worth completely unrolling and whether code need be generated for odd iterations.
<code>#pragma optimize_as_cmd_line</code>	Resets the optimization settings to be those specified on the command line when the compiler was invoked.
<code>#pragma no_alias</code>	Tells the compiler the following has no loads or stores that conflict because of references to the same location through different pointers, known as "aliases".

Table 3-8. C/C++ Compiler Pragas (Cont'd)

Pragma	Function
<code>#pragma alloc</code>	Tells the compiler that the function behaves like the library function “malloc”, returning a pointer to a newly allocated object.
<code>#pragma pure</code>	Tells the compiler that the function does not write to any global variables, and does not read or write any volatile variables.
<code>#pragma const</code>	This is a more restrictive form of the <code>pure</code> pragma. It tells the compiler that the function does not read from or write to global variables and does not read or write volatile variables. The result of the function is therefore a function of its parameters.
<code>#pragma regs_clobbered <i>string</i></code>	Used with a function declaration or definition to specify which registers are modified (or clobbered) by that function.
<code>#pragma result_alignment (<i>n</i>)</code>	Asserts that the pointer or integer returned by the function has a value that is a multiple of <i>n</i> .
<code>#pragma instantiate <i>instance</i></code>	Requests the compiler to instantiate <i>instance</i> in the current compilation.
<code>#pragma do_not_instantiate <i>instance</i></code>	Directs the compiler not to instantiate <i>instance</i> in the current compilation.
<code>#pragma can_instantiate <i>instance</i></code>	Tells the compiler that if <i>instance</i> is required anywhere in the program, it should be instantiated in this compilation. This pragma has the same effect as <code>#pragma instantiate</code> .
<code>#pragma hdrstop</code>	Used in conjunction with the <code>-pch</code> (precompiled header) switch. The switch tells the compiler to look for a precompiled header (.pch file), and, if it cannot find one, to generate a file for use on a later compilation.
<code>#pragma no_pch</code>	Overrides the <code>-pch</code> (precompiled headers) switch for a particular source file and directs the compiler not to look for a .pch file and not to generate one for the specified source file.

Table 3-8. C/C++ Compiler Pragmas (Cont'd)

Pragma	Function
<code>#pragma once</code>	Appears at the beginning of a header file and tells the compiler that the header is written in such a way that including it several times has the same effect as including it once.
<code>#pragma system_header</code>	Appears in a header file and identifies the file as one that is supplied with VisualDSP++.
<code>#pragma avoid_anomaly_45 on/off</code>	When executing code from external SDRAM on the ADSP-21161 DSP, conditional instructions containing a DAG1 data access may not be performed correctly. This pragma initiates (or avoids) the generation of such instructions on a function-by-function basis. It is used before a function definition and remains in effect until another variant of the pragma is seen.

GCC Compatibility Extensions

The compiler provides compatibility with the C dialect accepted by version 3.2 of the GNU C Compiler. Many of these extensions are available in the C99 ANSI Standard. A brief description of the following extensions is included in the compiler manual (see Chapter 1):

- Statement expressions
- Type reference support keyword (`typeof`)
- GCC generalized `Lvalues`
- Conditional expressions with missing operands
- Hexadecimal floating-point numbers
- Arithmetic on pointers to `void` and pointers to functions
- Cast to `union`
- Ranges in `case` labels

Compiler and Library for SHARC Processors

- Declarations mixed with code
- Zero-length arrays
- Variable argument macros
- Line breaks in string literals
- Escape character constant
- Alignment inquiry keyword (`__alignof__`)
- Keyword for specifying names in generated assembler (`asm`)
- Function, variable and type attribute keyword (`__attribute__`)

Predefined Compiler Macro

The new predefined macros that `cc21k` provides are listed below.

Macro	Function
<code>__2126x__</code>	When compiling for the ADSP-21261/21262/21266 or ADSP-21267 DSPs, <code>cc21k</code> defines <code>__2126x__</code> as 1.
<code>__2136x__</code>	When compiling for the ADSP-21363, ADSP-21364, or ADSP-21365 DSPs, <code>cc21k</code> defines <code>__2136x__</code> as 1.
<code>__ADSP21261__</code>	<code>cc21k</code> defines <code>__ADSP21261__</code> as 1 when you compile with the <code>-21261</code> command-line switch.
<code>__ADSP21262__</code>	<code>cc21k</code> defines <code>__ADSP21262__</code> as 1 when you compile with the <code>-21262</code> command-line switch.
<code>__ADSP21266__</code>	<code>cc21k</code> defines <code>__ADSP21266__</code> as 1 when you compile with the <code>-21266</code> command-line switch.
<code>__ADSP21267__</code>	<code>cc21k</code> defines <code>__ADSP21267__</code> as 1 when you compile with the <code>-21267</code> command-line switch.
<code>__ADSP21363__</code>	<code>cc21k</code> defines <code>__ADSP21363__</code> as 1 when you compile with the <code>-21363</code> command-line switch.

New Features and Enhancements

Macro	Function
<code>__ADSP21364__</code>	cc21k defines <code>__ADSP21364__</code> as 1 when you compile with the <code>-21364</code> command-line switch.
<code>__ADSP21365__</code>	cc21k defines <code>__ADSP21365__</code> as 1 when you compile with the <code>-21365</code> command-line switch.
<code>__EXCEPTIONS</code>	The <code>__EXCEPTIONS</code> macro is defined as 1 when C++ exception handling is enabled (using the <code>-eh</code> switch).
<code>__LANGUAGE_C</code>	The <code>__LANGUAGE_C</code> macro is always defined as 1. This macro is present when used for C compiler calls to specify headers.
<code>__RTTI</code>	The <code>__RTTI</code> macro is defined as 1 when C++ run-time type information is enabled (using the <code>-rtti</code> switch).
<code>__SILICON_REVISION__</code>	The <code>__SILICON_REVISION__</code> macro is defined to a value specific to each silicon revision. It is defined when the <code>-si-revision</code> switch is used.
<code>__WORKAROUNDS_ENABLED</code>	cc21k defines this macro to be 1 if any hardware workarounds are implemented by the compiler. This macro is set if the <code>-si-revision</code> option has a value other than “none” or if any specific workaround is selected by means of the <code>-workaround</code> compiler switch.

`seg_int_code` Memory/Section Name

The cc21k C/C++ run-time environment requires that a specific set of memory section names to be used for placing code in memory.

The `seg_int_code` section must always be located in internal memory and contains library code that modifies the interrupt latch registers (IMASKP and IRPTL). A hardware anomaly on a number of SHARC DSPs means that it is unsafe for code located in external memory to modify these registers. This section is used to locate the affected library code in internal memory without restricting the location of the rest of the library code.

Support for argv/argc

By default, the facility to specify arguments that get passed to your `main()` (`argv/argc`) at run-time is not enabled. To enable it, you need to modify application builds in the following ways:

1. Define your command-line arguments in C by defining a variable called “`__argv_string`”. When linked, your new definition will over-ride the default zero definition otherwise found in the C run-time library. For example,

```
const char __argv_string[] = "-in x.gif -out y.jpeg";
```
2. To use command-line arguments as part of Profile-Guided Optimizations (PGO), it is necessary to define `__argv_string` within a memory section called `seg_argv`. Therefore, define a memory section called `seg_argv` in your `.LDF` file and include the definition of `__argv_string` in it if you use PGO. The default `.LDF` files will complete this task for you if macro `IDDE_ARGS` is defined at link time.

File I/O Support

In earlier VisualDSP++ releases, the implementation of the functions defined in the header file `stdio.h` was based on a device driver, known as `primIO`, provided by the VisualDSP++ simulator and EZ-KIT Lite systems. This device driver, however, only provides access to the host file system.

VisualDSP++ 3.5 permits the registration of alternative device drivers. These drivers can then be used through the normal `stdio` functions and, therefore, enable the routines to interact with a device other than the host file system. By default, the `stdio` functions will continue to use the device driver provided by the VisualDSP++ simulator and EZ-KIT Lite systems, and users should not notice any difference in functionality. Full details of the new extensible driver mechanism are available in the C/C++ compiler manual.

Run-Time Libraries and Start-Up Files

A new set libraries supplied with this version of VisualDSP++ contain run-time support for C++ exception handling. These libraries are required by any program compiled with the switches `-eh -rtti`, and are included in an executable file when the default LDF file is used. The new C++ run-time libraries are described in [Table 3-9](#).

Table 3-9. C++ Run-time Library Files

Library – Startup Files	Description
<code>libcppeh.dlb</code> <code>libcppehmt.dlb</code>	C++ run-time library with exception handling
<code>libcpprteh.dlb</code> <code>libcpprteht.dlb</code>	C++ run-time support library with exception handling
<code>libeh.dlb</code> <code>libehmt.dlb</code>	C++ exception handling support library

Variants of the I/O library (`libio.dlb`) are available that do not support all the new features of VisualDSP++ 3.5. These libraries may be of benefit to applications that do not require the new functionality and would prefer a smaller executable file. The file name of the alternative I/O libraries has the form `libio*_lite.dlb`. When using the default LDF file, you may use the `-flags-link -MD__LIBIO_LITE` switch to link this file name with an application. Refer to Chapter 3 in the VisualDSP++ 3.5 C/C++ compiler manual for a more detailed explanation.

C Run-Time Library Functions

The C run-time library has been extended with the addition of some new functions and enhanced functionality.

The formatted input/output functions defined in `stdio.h` (such as, `printf`, `scanf`, `fprintf`, ...) now support the `%a` conversion specifier. The `%a` specifier is similar both in form and meaning to the `%e` specifier,

except that the `%e` specifier is used to input and output decimal floating-point numbers, while the `%a` specifier is used to input and output hexadecimal floating-point numbers.

Additional functions defined in the `stdio.h` header file are supported in the VisualDSP++ 3.5 release. These functions are:

`fgetpos`, `fseek`, `fsetpos`, `ftell`, `remove`, `rename`, `rewind`

 The C standard `stdio.h` functions `tmpfile` and `tmpnam` are not supported in this release.

The `math.h` header file has been extended to provide support for the `long double`, `double` and `float` data types. The names of the `long double` functions are the same as those for the equivalent `double` function with `d` appended to their name. For example, the header file contains the following prototypes for the sine function:

```
float sinf (float x);  
double sin (double x);  
long double sind (long double x);
```

The `long double` functions are independent of the size of type `double`.

The library functions `atof`, `atoff`, `strtod`, and `strtodf` defined in `stdlib.h` have been modified to support hexadecimal floating-point numbers. The header file also defines the new functions `atold` and `strtold` for the `long double` data type. The function documentation has been updated to reflect this new functionality.

DSP Run-Time Library Functions

The DSP run-time library has been extended with new functions and with support for the `double` and `long double` data types, including the `complex_double` and `complex_long_double` data types. These new features are reflected in the DSP header files and are described in the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual for SHARC® Processors*.

Several header files for the DSP run-time library now define additional functions that support the `long double` and `double` data types; the `complex.h` header file has also been extended with definitions of the `complex_long_double` and `complex_double` data types. In general, the names of the functions that support the `float` (or `complex_float`) data type have the suffix `f`, functions that support the `long double` (or `complex_long_double`) data type have the suffix `d`, while functions that support the `double` (or `complex_double`) data type have no suffix. The header files that have been extended with this functionality are:

```
cmatrix.h
complex.h
cvector.h
matrix.h
stats.h
vector.h
```

The newly added functions perform basic arithmetic between two complex matrices and also between a complex matrix and a complex scalar. The names of these functions, which are defined in the header file `cmatrix.h`, are:

Library Function	Description
<code>cmatmadd</code>	adds two complex matrices
<code>cmatmmlt</code>	multiplies two complex matrices
<code>cmatmsub</code>	subtracts two complex matrices
<code>cmatsadd</code>	adds a complex scalar to a complex matrix
<code>cmatsmmlt</code>	multiplies a complex matrix with a complex scalar
<code>cmatssub</code>	subtracts a complex scalar from a complex matrix

Besides defining the data types `complex_long_double` and `complex_double`, the header file `complex.h` also includes prototypes for the following new functions in VisualDSP++ 3.5:

Library Function	Description
<code>arg</code>	computes the phase associated with a Cartesian number
<code>cadd</code>	adds two complex scalars
<code>cartesian</code>	transforms a complex number in Cartesian notation to polar notation
<code>cdiv</code>	divides two complex scalars
<code>cm1t</code>	multiplies two complex scalars
<code>conj</code>	computes the conjugate of a complex scalar
<code>csub</code>	subtracts two complex scalars
<code>norm</code>	normalizes a complex scalar
<code>polar</code>	transforms a polar coordinate into Cartesian notation

The set of DSP filters and transformations for the ADSP-211xx and ADSP-212xx processors that are defined in the header file `filter.h` has been extended with four new functions—a function that calculates a biquad filter for a vector, a convolution function, and two functions that calculate the power spectrum of a real and complex FFT respectively:

Library Function	Description
<code>biquad</code>	cascaded biquad filter for vectors
<code>convolve</code>	convolves two vectors
<code>cfft_mag</code>	computes the power spectrum of a complex FFT
<code>rfft_mag</code>	computes the power spectrum of a real FFT



The `convolve` function is also available for ADSP-2106x processors and is defined in the header file `filters.h`.

New Features and Enhancements

Some other new functions have been added to the DSP run-time library; these functions are identified as:

Library Function	Description
<code>a log</code>	calculates the natural (base e) anti-log of its argument
<code>a log10</code>	calculates the base 10 anti-log of its argument
<code>count_ones</code>	counts the number of bits that are set to 1 in its argument

Linker and Utilities

The VisualDSP++ 3.5 linker and utility programs are upgraded to operate more efficiently on TigerSHARC and SHARC processors.

For the linker and utilities, the most notable new features and enhancements are:

- “Modified Link Page in Project Options Dialog Box” on page 3-44
- “Migrating LDFs from Previous Installations” on page 3-46
- “Linker Command-Line Switches” on page 3-48
- “Updated List of LDF Keywords” on page 3-50
- “Modifications to LDF Commands” on page 3-51
- “Expert Linker” on page 3-53
- “Expert Linker” on page 3-53
- “Memory Map File (.XML)” on page 3-55
- “Archiver” on page 3-55

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

Modified Link Page in Project Options Dialog Box

Within VisualDSP++, the **Link** tab of the **Project Options** dialog box was modified to have more flexibility for specifying tool settings for project builds. Choosing a **Category** from the pull-down list at the top of the **Link** tab presents several different pages of options.

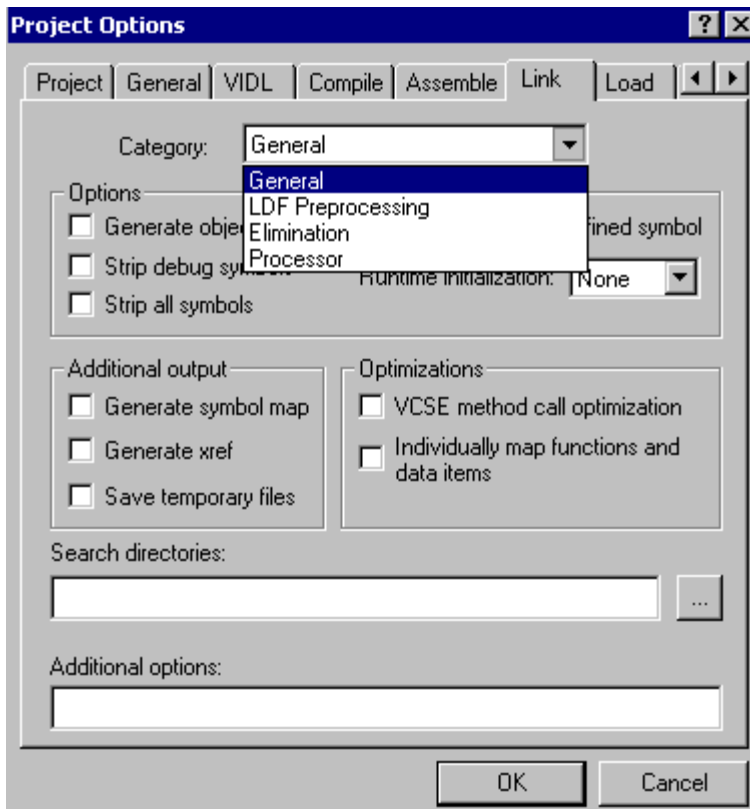


Figure 3-1. Main Link Tab with Category Selections

There are four sub-pages you can access—**General**, **LDF Preprocessing**, **Elimination**, and **Processor**. Almost every setting option has a corresponding compiler command-line switch. For more information, refer to the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors* and online Help.

Migrating LDFs from Previous Installations

Migrating LDF files from previous VisualDSP++ installations involves changes to:

- Handling of C++ global constructors and destructors, and support for run-time argument processing used with Profile-Guided Optimization (PGO)
- Support of C++ exceptions, zero-initialized data, and run-time initialization (TigerSHARC architecture only)
- Support for a separate code section, located in internal memory, and for code which modifies interrupt latch registers (SHARC processors only)

C++ Global Constructors and Destructors

In VDSP++ 3.0, the end-of-list markers for global constructor and destructor lists were located in a separate library, which meant that the correct link order for C++ libraries was crucial. In VDSP++ 3.5, the mechanism for locating these markers has been changed. The end markers are now located in a separate input section (“ctor1” on TigerSHARC processors, “seg_ctdm1” on SHARC processors), which is linked into the same data section as the constructors and destructors. The linker ensures that the end markers are placed correctly.

For TigerSHARC processors, the new input section is handled as follows.

```
ctor
{
    INPUT_SECTIONS( $OBJECTS(ctor0) $LIBRARIES(ctor0) )
    INPUT_SECTIONS( $OBJECTS(ctor1) $LIBRARIES(ctor1) )
    INPUT_SECTIONS( $OBJECTS(ctor2) $LIBRARIES(ctor2) )
    INPUT_SECTIONS( $OBJECTS(ctor3) $LIBRARIES(ctor3) )
    INPUT_SECTIONS( $OBJECTS(ctor4) $LIBRARIES(ctor4) )
```

```
INPUT_SECTIONS( $OBJECTS(ctor) $LIBRARIES(ctor) )  
INPUT_SECTIONS( $OBJECTS(ctor1) $LIBRARIES(ctor1) )  
} >M1Data
```

For SHARC processors, the new input section is handled as follows.

```
seg_ctdm  
{  
    __ctors = .;    /* points to the start of the section */  
    INPUT_SECTIONS( $OBJECTS(seg_ctdm) $LIBRARIES(seg_ctdm))  
    INPUT_SECTIONS( $OBJECTS(seg_ctdm1) $LIBRARIES(seg_ctdm1))  
} > seg_ctdm
```

Run-Time Argument Processing for PGO

The PGO feature of VDSP++ 3.5 requires that the argument string passed to `main()` is placed in a separate data section (“`seg_argv`” on SHARC processors, “`MEM_ARGV`” on TigerSHARC processors). This data section should be defined in the `.ldf` file, and the argument string (called “`__argv_string`”) should be placed in this section using the `RESOLVE()` linker command.

C++ Exceptions

The compiler’s C++ exception support relies on a support library and exceptions-enabled versions of the run-time libraries. In addition, the exceptions support requires several new input sections (`.frt`, `.edt`, `.cht`, `.gdt`) that must be mapped to data areas. Note that exceptions are disabled by default.

Zero-Initialized Data

The generation of zero-initialized data relies on the creation of new input sections (`.bss` on SHARC processors and `.bsz` on TigerSHARC processors) that are mapped to existing data areas. Note that this feature is disabled by default.

Run-Time Initialized Data (TigerSHARC Processors Only)

On TigerSHARC processors only, variables that are to be initialized at run-time are placed in a separate input section, “bsz_init”, which is processed by the Meminit utility.

Modifying Interrupt Registers (SHARC Processors Only)

A hardware anomaly on SHARC processors means that code located in external memory and used to modify interrupt latch registers may cause incoming interrupts to be ignored. To avoid this anomaly, all affected code in the run-time library is placed in a separate code section, “seg_int_code”, which is located in internal memory.

Others

The key points to note are:

- The output sections “bss” and “bsz” have the ZERO_INIT flag specified.
- The special output section named “.meminit” is included.

For more detail on the changed and new functionality for run-time initialization (ZERO_INIT and RUNTIME_INIT), refer to the SECTIONS description in the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Linker Command-Line Switches

Table 3-10 lists the new or modified linker command-line switches.

Files created during the link are named with an appropriate extension and are formatted accordingly. A map file is generated in XML format only and is given an .XML extension. An executable file is written in the ELF format and is given a .DXX extension.

Table 3-10. Linker Command-Line Switches

Switch	Description
<code>-DprocessorID</code>	Modified syntax and description. Specifies the target processor ID. The use of the <code>-proc processor</code> switch is recommended.
<code>-ek secName</code>	Specifies a section name in which elimination should not take place.
<code>-Ovcse</code>	Enables VCSE method call optimization.
<code>-MUD</code>	Undefines the preprocessor macro.
<code>-Wnumber</code>	Selectively disables warnings by one or more message numbers. For example, <code>-W1010</code> disables warning message 11010.
<code>-Wwarn number</code>	Demotes the specified error message to a warning.
<code>-flags-meminit</code>	Passes each comma-separated option to the Memory Initializer utility.
<code>-flags-pp</code>	Passes each comma-separated option to the preprocessor.
<code>-ip</code>	Modified syntax and description. Fills fragmented memory with individual data objects that fit and requires that objects have been assembled with the assembler's <code>-ip</code> switch.
<code>-meminit</code>	Directs the linker to post-process the <code>.DXE</code> file through the Memory Initializer utility. This action will cause the sections specified in the <code>.LDF</code> file to be “run-time” initialized by the C run-time library. By default, if this flag is not specified, all sections are initialized at “load” time (for example, via the VisualDSP++ IDDE or the boot loader).
<code>-save-temps</code>	Saves temporary output files.
<code>-si-revision version</code>	Specifies a silicon revision of the specified processor.
<code>-v -verbose</code>	Verbose—outputs status information.
<code>-xref filename</code>	Produces a cross-reference file (<code>xref.xml</code> format).

Updated List of LDF Keywords

Table 3-11 lists .LDF file keywords that apply to all 32-bit processors.

 Keywords are case sensitive; the linker recognizes a keyword only when the *entire* word is UPPERCASE.

Table 3-11. LDF File Keywords Summary

ABSOLUTE	ADDR	ALGORITHM
ALIGN	ALL_FIT	ARCHITECTURE
BEST_FIT	BM ¹	BOOT
DEFINED	DM ¹	ELIMINATE
ELIMINATE_SECTIONS	END	FALSE
FILL	FIRST_FIT	INCLUDE
INPUT_SECTION_ALIGN	INPUT_SECTIONS	KEEP
LENGTH	LINK_AGAINST	MAP
MEMORY	MEMORY_SIZEOF	MPMEMORY
NUMBER_OF_OVERLAYS	OUTPUT	OVERLAY_GROUP
OVERLAY_ID	OVERLAY_INPUT	OVERLAY_OUTPUT
PACKING	PLIT	PLIT_SYMBOL_ADDRESS
PLIT_SYMBOL_OVERLAYID	PM ¹	PROCESSOR
RAM	RESOLVE	RESOLVE_LOCALLY
ROM	SEARCH_DIR	SECTIONS
SHARED_MEMORY	SHT_NOBITS	SIZE
SIZEOF	SROM ¹	START
TYPE	VERBOSE	WIDTH
XREF		

¹ Supported on ADSP-21xxx processors only.

Modifications to LDF Commands

Some LDF commands are enhanced to support better linking and memory management in VisualDSP++ 3.5.

KEEP() LDF Command

For the C and C++ run-time libraries to work properly, use `KEEP` to retain the following symbols:

`__ctor_NULL_marker` and `__lib_end_of_heap_descriptions`

A symbol specified in *keepList* must be a global symbol.

KEEP_SECTIONS()

The linker uses the `KEEP_SECTIONS()` command to specify a section name in which elimination should *not* take place. This new command can appear anywhere the `ELIMINATE_SECTION` command appears. You may either use the `KEEP_SECTIONS()` command or the `-ek` linker switch.

MAP()

The `MAP(filename)` command was modified to output a map file in the `.XML` format only.

MEMORY{} LDF Command

The `MEMORY{}` command specifies the memory map for the target system. A *segment declaration* declares a memory segment on the target processor.

`TYPE()` in the *segment declaration* identifies the architecture-specific type of memory within the memory segment.

Note: Not all target processors support all types of memory. The linker

stores this information in the executable file for use by other development tools.

- For TigerSHARC processors, use `TYPE()` to specify the functional or hardware locus (RAM or ROM). The RAM declarator specifies segments that need to be booted. ROM segments are not booted; they are executed/loaded directly from off-chip PROM space.
- For ADSP-21xxx processors, use `TYPE()` to specify two parameters: memory usage (PM for program memory or DM for data memory), and functional or hardware locus (RAM or ROM, as described above).

SECTIONS{} LDF Command

The `SECTIONS{} command` uses memory segments (defined by `MEMORY{} commands`) to specify the placement of output sections in memory.

The `section_declaration` of the `SECTIONS{} command` is enhanced to use the special section name `.MEMINIT` to indicate where to place the “run-time” initialization structures to be used by the C run-time library. The linker “places” this section into the largest available unused memory at the specified memory segment. The Memory Initializer post-processor fills this space with the data needed by the C run-time library for run-time initialization. The `.MEMINIT` section should be placed in non-overlay memory.

The *init_qualifier* specifies run-time initialization type (optional). The qualifiers are:

- `NO_INIT` – The section type contains un-initialized data. No data is stored in the `.DXX` file for this section (equivalent to the `SHT_NOBITS` legacy qualifier).

- `ZERO_INIT` – The section type contains only “zero-initialized” data. If invoked with the `-meminit` switch, the “zeroing” of the section is done at runtime by the C run-time library. If `-meminit` is not specified, the “zeroing” is done at “load” time.
- `RUNTIME_INIT` – If the linker is invoked with the `-meminit` switch, this section is filled at runtime. If `-meminit` is not specified, the section is filled at “load” time.

The `OVERLAY_INPUT{}` command was modified to include the `DEFAULT_OVERLAY()` command. When this command is used, linker places the overlay in the run-time space initially (that is, without running the overlay manager).

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

OVERLAY_GROUP{} Command

The `OVERLAY_GROUP{}` LDF command is deprecated and is not recommended for use. Though in VisualDSP++ 3.5, the `OVERLAY_GROUP{}` can still be used to group overlays, it is recommended that you revise any LDF files which include the command. The linker of the current release processes all overlay groups and explicit `OVERLAY_GROUP{}` commands, producing a warning. The preferable way to manage overlays is to create a separate output section for each overlay group.

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`. For more information on the Expert Linker, refer to the corresponding chapter of the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors* and online Help.

Expert Linker Menu Updates

The VisualDSP++ 3.5 Expert Linker GUI and context menus have been enhanced to provide better processing for graphical data. The most significant menu changes are as follows.

In the **Input Sections** context menu, new selections are:

- **Remove** – Removes an LDF macro from another LDF macro but does not delete the input section mappings that contain the removed macro. The difference between **Delete** and **Remove** is that **Delete** deletes the input section macros that contain the deleted macro. The **Remove** option becomes available only if you right-click on an LDF macro that is part of another LDF macro.
- **Expand All LDF Macros** – Expands all the LDF macros in the input sections pane to display the contents of all the LDF macros.

In the **Memory Map** context menu, the new **Expand All** selection expands all items in the memory map tree so that their contents are visible.

Profiling Object Sections

The Expert Linker has been enhanced to give you the option of profiling object sections. If this feature is enabled, the Expert Linker uses the profiler to collect profiling information while your program is running. When the program halts, the Expert Linker graphically displays how much time was spent in each object section so that you can see “hotspots” in the code and move that code to faster internal memory.

To profile a program with the Expert Linker:

1. Enable profiling in the **Global Properties** dialog box.
2. Load the program into the current project. After the program is loaded, the Expert Linker sets up the profiling bins to collect the profiling information.

3. Run the program. When the program halts, the Expert Linker colors each object section with a different shade of red to indicate how much time was spent executing that section.

From the Expert Linker, you can view PC sample counts for object sections. To view an actual PC sample count, move the mouse pointer over an object section and view the PC sample count. To view sample counts for functions located within an object section, double-click on the object section. You can view detailed profile information, such as the sample counts, for each line in the function.

Memory Map File (.XML)

The linker can output memory map files that contain memory and symbol information for your executable file(s). The map contains a summary of memory defined with `MEMORY{ }` commands in the `.LDF` file and provides a list of the absolute addresses of all symbols. For VisualDSP++ 3.5, the file format has been changed from plain text to XML, and the extension has been changed from `.MAP` to `.XML`.

Archiver

For the VisualDSP++ 3.5 release, the archiver (`elfar`) provides several improvements. For more information about the archiver, refer to “*Archiver*” chapter in the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Archiver Switches

[Table 3-12](#) summarizes new archiver command-line switches.

Table 3-12. New Archiver Command-Line Switches

Item	Description
-anv	Appends one or more object files and clears version information
-dnv	Removes the listed object file(s) from the specified library file and clears version information
-pv	Prints only version information in library to standard output
-pva	Prints all version information in library to standard output
-t <i>verno</i>	Tags the library with version information in string
-tx <i>filename</i>	Tags the library with version information in the file
-twc <i>ver</i>	Tags the library with version information in the <i>num.num.num</i> form
-tnv	Clears version information from a library
-version	Prints the archiver (elfar) version to standard output
-w	Removes all archiver-generated warnings
-Wnnnn	Selectively disables warnings specified by one or more message numbers. For example, -W0023 disables warning message ar0023.

Warnings for Duplicate Library Entries

If two objects with the same name are put into a library, the archiver in VisualDSP++ 3.5 issues the warning:

```
[Warning ea0079] An object file named "<fname>" already exists in  
this library
```

Improved Support for File Specifications

In VisualDSP++ 3.5, the archiver accepts command lines with wildcard specification of the files for inclusion:

```
elfar -c mylib.dlb *.doj
```

The archiver now accepts UNC filename specification:

```
elfar -r fruit.dlb \\c\tests\strawberry.doj
```

or

```
elfar -r fruit.dlb //c/tests/strawberry.doj
```

Tagging an Archive with Version Information

The archiver supports embedding version information into a library built with `elfar`. The following is a list of version information tagging features provided by the archiver.

Basic Version Information

You can “tag” an archive with a version. The easiest way to tag an archive is to use the `-t` switch, which can be used with any other `elfar` switch. To highlight the version information, precede it with “:.”.

User-Defined Version Information

Any number of user-defined version values can be provided by supplying a text with those values. Each line in the text file begins with a name, followed by a space and then the value associated with that name.

Linker and Utilities

Printing Version Information

Use the `-p` switch to print version information. The `-pv` switch prints only version information and does not print the contents of the archive. The `-pva` switch prints all version information. Version names without values are not be printed with `-p` or `-pv` but are shown with `-pva`.

Removing Version Information from an Archive

Adding “`nv`” to a switch strips version information. In addition, a special form of the `-t` switch, which takes no argument, can be used for stripping version information from an archive.

Checking Version Number

The `-twc` switch causes the archiver to raise a warning if the version number is not provided. The check ensures that the version number starts with a number in that format.

Adding Text to Version Information

You can add additional text to the end of the version information.

Loaders and Splitter

The loader program (`elfloader.exe`) for TigerSHARC and SHARC processors has been modified to support new processors.

TigerSHARC Loader Features

The TigerSHARC loader modifications are:

- The loader has been updated to support new ADSP-TS202 and ADSP-TS203 processors. Refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors* for details.
- The `-version` switch has been added to show the version number of the loader.
- The `-si-revision` switch has been added to provide a silicon revision number to the loader.
- The SPI boot mode now supports Intel HEX format.
- The `-fs3`, `-fs2`, and `-fs1` switches have been added to generate a loader file in Motorola S-record format.

SHARC Loader Features

The TigerSHARC loader modifications are:

- The loader has been updated to support new ADSP-2126x and ADSP-2136x processors. Refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors* for details.
- The `-version` switch has been added to show the version number of the loader.
- The `-si-revision` switch has been added to provide a silicon revision number to the loader.

Loaders and Splitter

- The `-fs3`, `-fs2`, and `-fs1` switches have been added to generate a loader file in Motorola S-record format.
- The `-id#exe=N` switch has been added to identify an `N` loader jump table entry for the `ID( # )` image. This switch is for the use with MP systems only.

For details, see the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors*.

Splitter for TigerSHARC Processors

The splitter (`elfspl21k.exe`) has been added to support TigerSHARC processors. Refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors* for details.

HEXUTIL Command Line Tool

The TigerSHARC and SHARC loader uses the new command line tool “`hexutil`”, which converts an Intel HEX file to a different format. Refer to the *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors* for details.

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
- Automatically generate a set of HTML pages that document a component and the interfaces it provides or requires. The generated documentation include a table of contents and an index.

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 can automatically generate a test shell component from the VIDL definition of a component. The generated test can include code to validate the argument values passed to and from the component, carry out array bound checking of arguments, ensure that methods of an interface are called in the correct sequence, and measure the resources used by a component.

The level of checking effected by the test shell can be controlled by the developer. The generated test shells can be used by the developer for testing his component and by the user to ensure the use of the component is valid.

VDK

The following VisualDSP++ Kernel (VDK) features have been added to the VisualDSP++ 3.5 release.

- Support for configuring and using multiple heaps.
- Ability to specify, at build time, the heap from which each object (such as semaphores, device flags, and so on) is allocated.
- Ability to specify, at build time, the heap from which the stack and thread structure for each thread type are allocated.
- Ability to specify the size of the stack for the Idle thread and the heap from which it is allocated.
- Support for inter-processor messaging, which uses the same API as intra-processor messaging.
- Out-of-the-box support for transporting messages between ADSP-TS101, ADSP-TS201 and ADSP-2116x processors over link ports.
- Support for marshalling the payloads of inter-processor messages which have been allocated from heaps and memory pools.
- Support for user-defined marshalling of message payloads.
- Ability to specify the routing between processors to allow messages to be passed between processors that are not directly connected.
- Support for importing projects that will be used on other processors to simplify the configuration of multi-processor messaging.

Simulators for TS101 and TS20x Processors

Simulators for TS101 and TS20x processors allow manual changes to the program counter value in the PC register window.

Object Protection

The VisualDSP++ 3.5 tool chain now supports the encryption and decryption of object files. This feature enables algorithm providers to distribute objects and libraries under license protection.

Documentation Changes

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

Compiler Manuals

In VisualDSP++ 3.5, each compiler manual has a new chapter, called “*Achieving Optimal Performance from C/C++ Source Code*”. The text focuses on how to fine-tune your program code to obtain maximal code performance from the compiler while optimizing for minimum code size.

The chapter starts with discussing some general optimization principles and explains how the compiler can lend the most help to your optimization effort. Optimal coding styles are described in detail. Special features, such as compiler switches, built-in functions, and pragmas are also discussed. The chapter ends with a short example to demonstrate how the optimizer works.

The new chapter (Chapter 2) is available in the following manuals:

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

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

In addition to the new chapter, each compiler manual in the VisualDSP++ 3.5 documentation set has been enhanced with new or existing run-time libraries, header files, and function descriptions. Notably, the `stdio.h` header file's description now provides additional information about alternative device drivers, software restrictions, and I/O support using `_primIO()`.

VisualDSP++ 3.5 User's Guide

The legacy Tcl documentation is removed from the User's Guide but made available through the technical support as requested.

Online Help

An ongoing effort to enhance the VisualDSP++ online Help has yielded a more detailed error message presentation, a unified index for all online manuals in the documentation set, and the ability to search for keywords across the documentation set.

Error Messages

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

Documentation Changes

Merged Index

Index entries from all the VisualDSP++ 3.5 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++ 3.5 publications for the target processor family. The Help window lists each occurrence and its location (manual).

Online Manuals

All manuals in the VisualDSP++ 3.5 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++ 3.5 installation CD contains a complete set of documentation for processors that are supported by your development tools suite. The set comprises 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 book or for each chapter in the manual.

4 OBSOLETE OR REMOVED FEATURES

This chapter describes the features that have been deprecated or removed since VisualDSP++ 3.5. Read this chapter if you are 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 into VisualDSP++ 3.0. Similarly, new projects created using VisualDSP++ 3.5 cannot be used by earlier versions of the tools.

This chapter contains listings of obsolete or removed features:

- [“Assembler and Preprocessor” on page 4-2](#)
- [“Compiler and Library for SHARC Processors” on page 4-3](#)
- [“Compiler and Library for TigerSHARC Processors” on page 4-5](#)
- [“Linker” on page 4-7](#)
- [“Tcl Scripting Engine” on page 4-7](#)

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

For VisualDSP++ 3.5 assembler and preprocessor for both TigerSHARC and SHARC processors, the deprecated or removed features are grouped as follows.

Preprocessor Command-Line Switch

Switch	Description
-cpredef	Deprecated; replaced with the -cstring switch

Compiler and Library for SHARC Processors

The compiler features that were either renamed, removed, or became obsolete in VisualDSP++ 3.5 (as compared to VisualDSP++ 3.0) are listed in [“C/C++ Compiler Command-Line Switches”](#).

Refer to Chapter 3, *“New Features and Enhancements”*, of this manual for more information about the changes to the C/C++ compiler and run-time libraries.

C/C++ Compiler Command-Line Switches

The following switches have been deprecated or removed:

Table 4-1. Obsolete/Deprecated C or C++ Mode Selection Switches

Switch Name	Description
-analog	Renamed to -c89
-traditional	Removed

Table 4-2. Obsolete/Deprecated C/C++ Compiler Common Switches

Switch Name	Description
-21xxx	All -21xxx switches are deprecated, use proc -ADSP-21xxx switch instead
-auto-inline factor	Removed
-dollar	Removed
-line-debug	Removed
-no-dir-warnings	Removed
-no-dollar	Removed
-no-inline	Removed

Table 4-2. Obsolete/Deprecated C/C++ Compiler Common Switches

Switch Name	Description
-no-restrict	Removed; use -no-extra-keywords instead
-swf-screening	Removed
-restrict	Removed; now the default mode
-traditional	Removed
-xml	Removed

Table 4-3. Obsolete/Deprecated C++ Mode Compiler Switches

Switch Name	Description
-explicit	Removed
-instant[all local used]	Removed
-namespace	Removed
-newforinit	Removed
-newvec	Removed
-no-explicit	Removed
-no-namespace	Removed
-no-newvec	Removed
-notstrict	Removed; now is the default mode
-no-wchar	Removed
-strict	Removed; use -pedantic-errors instead
-strictwarn	Removed; use -pedantic instead
-tpautooff	Removed
-trdforinit	Removed
-typename	Removed
-wchar	Removed

Compiler and Library for TigerSHARC Processors

The compiler features that were either renamed or became obsolete in VisualDSP++ 3.5 (compared to VisualDSP++ 3.0) are described in [Table 4-4](#) through [Table 4-6](#).

Table 4-4. Obsolete/Deprecated C or C++ Mode Selection Switches

Switch Name	Description
-analog	Renamed to -c89
-traditional	Removed

Table 4-5. Obsolete/Deprecated C Compiler Switches

Switch Name	Description
-TS101 201	Removed; use <i>-proc processor</i> instead
-dollar	Removed
-inline	Removed
-J	Removed
-no-dollar	Removed
-no-dir-warnings	Removed
-no-inline	Removed
-no-restrict	Removed; use <i>-no-extra-keywords</i> instead
-restrict	Removed; now the default mode
-xml	Removed

Table 4-6. Obsolete/Deprecated C++ Mode Compiler Switches

Switch Name	Description
<code>-explicit</code>	Removed
<code>-instant[all local used]</code>	Removed
<code>-namespace</code>	Removed
<code>-newforinit</code>	Removed
<code>-newvec</code>	Removed
<code>-no-explicit</code>	Removed
<code>-no-namespace</code>	Removed
<code>-no-newvec</code>	Removed
<code>-notstrict</code>	Removed; now is the default mode
<code>-no-wchar</code>	Removed
<code>-strict</code>	Removed; use <code>-pedantic-errors</code> instead
<code>-strictwarn</code>	Removed; use <code>-pedantic</code> instead
<code>-tpautooff</code>	Removed
<code>-trdforinit</code>	Removed
<code>-typename</code>	Removed
<code>-wchar</code>	Removed

Linker

For VisualDSP++ 3.5, no Linker Description File, command-line switches, LDF commands, and utilities were removed or considered obsolete. The only change involves the use of the `OVERLAY_GROUP{ }` legacy LDF command.

In VisualDSP++ 3.5, the `OVERLAY_GROUP{ }` is still used to group overlays, allowing each overlay to run from a different start address in run-time memory. The linker 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.

Tcl Scripting Engine

The existing Tcl scripting engine within the IDDE is removed in favor of the more generic Automation scripting approach, which is the preferred mechanism for VisualDSP++ scripting.

The new approach is backward-compatible in nearly all respects to the legacy Tcl functionality. Only four Tcl commands are no longer available in VisualDSP++ 3.5. If any of these commands are invoked, a message is issued to state that the command is no longer supported:

```
dspplotwin  
dspplotrotate  
dspmemorywin  
dspregisterwin
```

