

VISUAL**DSP++**[®] 4.5 Product Release Bulletin

Revision 2.0, April 2006

Part Number
82-000420-06

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



Copyright Information

©2006 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, Blackfin, SHARC, TigerSHARC, and VisualDSP++, EZ-KIT Lite, and EZ-Extender 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	ix
Intended Audience	ix
Manual Contents	x
Technical or Customer Support	x
Supported Processors	xi
Product Information	xi
MyAnalog.com	xii
Processor Product Information	xii
Related Documents	xiii
Online Technical Documentation	xiv
Accessing Documentation From VisualDSP++	xiv
Accessing Documentation From Windows	xv
Accessing Documentation From the Web	xv
Printed Manuals	xvi
VisualDSP++ Documentation Set	xvi
Hardware Tools Manuals	xvi
Processor Manuals	xvi
Data Sheets	xvi

CONTENTS

Notation Conventions	xvii
----------------------------	------

INTRODUCTION

Product Release Description	1-2
VisualDSP++ 4.5 System Requirements	1-3
Platform and Processor Support	1-3

VISUALDSP++ 4.5 NEW FEATURES AND ENHANCEMENTS

VisualDSP++ IDDE	2-2
New Processor Support	2-3
Support Enhancements	2-3
Connectionless IDDE	2-4
Starting VisualDSP++ While Holding Down the Shift Key ..	2-4
Starting VisualDSP++ While Holding Down the Ctrl Key ...	2-4
Session Wizard	2-5
Project Wizard Enhancements	2-6
Modified Property Pages in Project Options Dialog Box	2-6
Load : Compression Page of Project Options Dialog Box	2-7
Editor Window Enhancements	2-7
Breakpoint Enhancements	2-8
Profiling Results in XML Format	2-9
Expressions Window and Locals Window Enhancements	2-9
Flash Programmer Window Redesign	2-9
Power Estimation Analysis	2-10

Assembler	2-11
Processor Support	2-11
Assembler Feature Macros	2-12
.FILE_ATTR Directive	2-12
.ASCII Directive	2-13
.MESSAGE Directive	2-13
User Control of Assembler Message Severity	2-14
Features Common to All Compilers and Libraries	2-15
New Switches	2-15
New Pragmas	2-15
File Attributes	2-19
Inlining Control	2-19
Unnamed Struct and Union Fields Within Struct and Union Definitions	2-20
Additional Library Routines	2-20
Compiler and Library for Blackfin Processors	2-21
LDF Generator	2-21
Long-Long Types in asm Statements	2-22
Compiler Builtins for Accessing Memory-Mapped Registers ...	2-22
Speculative Memory Access Pragma	2-23
New Processor Support	2-24
Core-B Enabling Function	2-24
Compiler and Library for SHARC Processors	2-24
External Memory Access Support	2-25
New Interrupt Pragmas	2-25

CONTENTS

Bank Type Qualifiers	2-25
New Processor Support	2-26
Additional DSP Library Functions	2-27
Compiler and Library for TigerSHARC Processors	2-28
Linker and Utilities	2-29
Updated List of LDF Keywords	2-29
Linking with Attributes	2-30
RESERVE LDF Command	2-30
DMAONLY Qualifier	2-31
Loader and Utilities for Blackfin and SHARC Processors	2-31
Loader for Blackfin Processors	2-31
Loader for SHARC Processors	2-33
File Conversion Programs	2-36
VDK	2-36
Example Programs Re-Organized	2-37

VISUALDSP++ 4.5 MAJOR CHANGES

New Processor Support	3-2
TigerSHARC Simulator Platform Names	3-2
License Server Tools Upgrade	3-3
Assembler Changes	3-3
Blackfin .GLOBAL Directive Syntax / Error ea5004	3-3
Compiler Changes	3-4
Std Namespace is Now Default in C++ Mode	3-5
Revised .LDF Files	3-5

Multiprocessor and Multi-Core Support	3-6
Multi-Core Linking	3-7
Integrated Section-Placement Mechanisms	3-7
Non-Optimizing Inter-Procedural Analysis	3-7
weak_entry Pagma Restriction	3-8
Optimization Control Pragmas and Inter-Procedural Analysis ...	3-8
Extended Optimizer Annotations	3-8
Updated C++ Support Libraries and Header Files	3-8
Compiler and Library for Blackfin Processors	3-9
“M3-Free” Libraries No Longer Required	3-9
CRT Header File Name Appended with <project_name> ...	3-10
Compiler and Library for SHARC Processors	3-10
Compiler and Library for TigerSHARC Processors	3-10
Linker Changes	3-11
Migration of .LDF Files from Previous Versions of VisualDSP++	3-12
Blackfin-Specific LDF Features	3-12
Loader Changes	3-13
VDK Changes	3-13
Changes to Existing Projects	3-15
SPI_MS Macro	3-15
SSL Libraries in New Location	3-15
MMR Definitions Include File Uses of “volatile void” is Replaced	3-15
VISUALDSP++ 4.5 OBSOLETE OR REMOVED FEATURES	
Discontinued Processor Support	4-2

CONTENTS

VisualDSP++ IDDE	4-2
Compilers and Libraries	4-2
Removed Command-Line Switches	4-2
Deprecated Pragmas	4-3
VDK	4-3

PREFACE

Thank you for purchasing Analog Devices, Inc. development software for digital signal processing (DSP) applications.

Purpose of This Document

This document briefly describes the new features and enhancements provided by VisualDSP++[®] 4.5 release that supports the following Analog Devices, Inc. processor families—SHARC[®] (ADSP-21xxx) processors, TigerSHARC[®] (ADSP-TSxxx) processors, and Blackfin[®] (ADSP-BFxxx) processors.

It also describes the differences (obsolete features and functions) between VisualDSP++ 4.5 and previous VisualDSP++ releases.

For details, refer to the VisualDSP++ 4.5 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++ 4.5.

Manual Contents

This manual consists of:

- Chapter 1, “[Introduction](#)”
Describes VisualDSP++ 4.5 and its benefits, provides the minimal system requirements for running the product, and lists supported processors.
- Chapter 2, “[VisualDSP++ 4.5 New Features and Enhancements](#)”
Describes what is new in the VisualDSP++ 4.5 IDDE, assembler, compiler, linker, loader, and documentation. Also describes the new features in the Expert Linker (EL) and the VisualDSP++ Kernel (VDK).
- Chapter 3, “[VisualDSP++ 4.5 Major Changes](#)”
Describes major changes in VisualDSP++ 4.5 projects compared to VisualDSP++ 4.0 projects.
- Chapter 4, “[VisualDSP++ 4.5 Obsolete or Removed Features](#)”
Describes the removed/obsolete features in VisualDSP++ 4.5 (compared to the previous VisualDSP++ software release) as they pertain to code generation tool chain: commands, switches, operators, directives, pragmas, keywords, macros, and library functions.

Technical or Customer Support

You can reach 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 processor.tools.support@analog.com

- E-mail processor questions to
processor.support@analog.com (World wide support)
processor.europe@analog.com (Europe support)
processor.china@analog.com (China support)
- 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++ 4.5 is for Blackfin (ADSP-BFxxx), SHARC (ADSP-21xxx), and TigerSHARC (ADSP-TSxxx) processors. For the complete list of supported processors, see “[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.

Product Information

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
processor.support@analog.com (World wide support)
processor.europe@analog.com (Europe support)
processor.china@analog.com (China support)
- Fax questions or requests for information to
1-781-461-3010 (North America)
+49-89-76903-157 (Europe)

Related Documents

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

- *VisualDSP++ 4.5 Getting Started Guide*
- *VisualDSP++ 4.5 User's Guide*
- *VisualDSP++ 4.5 Assembler and Preprocessor Manual*
- *VisualDSP++ 4.5 C/C++ Compiler and Library Manual for Blackfin Processors*
- *VisualDSP++ 4.5 C/C++ Compiler and Library Manual for TigerSHARC Processors*
- *VisualDSP++ 4.5 C/C++ Compiler and Library Manual for SHARC Processors*
- *VisualDSP++ 4.5 Linker and Utilities Manual*
- *VisualDSP++ 4.5 Loader and Utilities Manual*
- *VisualDSP++ 4.5 Kernel (VDK) User's Guide*
- *VisualDSP++ 4.5 Quick Installation Reference Card*

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

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

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

Product Information

Online Technical Documentation

Online documentation includes 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 using the Search function of VisualDSP++ Help system. For easy printing, supplementary .pdf files of most manuals are also provided.

Each documentation file type is described as follows.

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 requires a browser, such as Internet Explorer 5.01 (or higher).
.pdf	VisualDSP++ and processor manuals in Portable Documentation Format (PDF). Viewing and printing the .pdf files requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher).

Access the online documentation from the VisualDSP++ environment, Windows[®] Explorer, or the Analog Devices Web site.

Accessing Documentation From VisualDSP++

From the VisualDSP++ environment:

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

Accessing Documentation 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 of VisualDSP++ environment. The .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 the 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.
- Open your VisualDSP++ installation CD-ROM and 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**.

Accessing Documentation From the Web

Download manuals at the following Web site:

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

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

Product Information

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. Only production (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); they also can be downloaded from the Web site.

To have a data sheet faxed to you, call the Analog Devices Faxback System at 1-800-446-6212. Follow the prompts and a list of data sheet code numbers will be faxed to you. If the data sheet you want is not listed, check for it on the Web site.




Notation Conventions

Text conventions used in this manual are identified and described as follows.

 Additional conventions, which apply only to specific chapters, may appear throughout this document.

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 required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as <i>this</i> or <i>that</i> . 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.

Notation Conventions

Example	Description
	<p>Note: For correct operation, ...</p> <p>A Note provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.</p>
	<p>Caution: Incorrect device operation may result if ...</p> <p>Caution: Device damage may result if ...</p> <p>A Caution identifies conditions or inappropriate usage of the product that could lead to undesirable results or product damage. In the online version of this book, the word Caution appears instead of this symbol.</p>
	<p>Warning: Injury to device users may result if ...</p> <p>A Warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for the devices users. In the online version of this book, the word Warning appears instead of this symbol.</p>

1 INTRODUCTION

This chapter describes the product, VisualDSP++, and the requirements for running its latest revision, 4.5. 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++ 4.5 System Requirements” on page 1-3](#)
- [“Platform and Processor Support” on page 1-3](#)

Product Release Description

VisualDSP++ is the Analog Devices project management and development environment for signal processing (DSP) applications. VisualDSP++ 4.5 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 VisualDSP++ 4.5 CD-ROM supplies the code generation tool chain comprised 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++ 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.

The successor to VisualDSP++ 4.0, VisualDSP++ 4.5 incorporates a number of new features and enhancements, as described in Chapter 2, [“VisualDSP++ 4.5 New Features and Enhancements”](#).

VisualDSP++ 4.5 System Requirements

To install and run VisualDSP++ 4.5, your computer must provide the following software, configuration, and system resources.

- Intel Pentium processor (or compatible), 500 MHz or better
- Windows® XP or 2000 only
-  Windows NT, 98, and ME are not supported.
- At least 1 GB of available hard drive space
- At least 512 MB of RAM
- CD-ROM drive
- Internet Explorer 5.01 or later

Platform and Processor Support

The following list of Analog Devices, Inc. processors is supported in VisualDSP++ 4.5.

Blackfin Processors

The name *Blackfin* refers to a family of 16-bit, embedded processors. VisualDSP++ currently supports the following Blackfin processors.

ADSP-BF531	ADSP-BF532
ADSP-BF533	ADSP-BF534
ADSP-BF535	ADSP-BF536
ADSP-BF537	ADSP-BF538
ADSP-BF539	ADSP-BF561

Platform and Processor Support

ADSP-BF566	AD6531 and AD6532
AD6900	AD6901
AD6902	AD6903

SHARC (ADSP-21xxx) Processors

The name *SHARC* refers to a family of high-performance, 32-bit, floating-point 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-21362
ADSP-21363	ADSP-21364	ADSP-21365	ADSP-21366
ADSP-21367	ADSP-21368	ADSP-21369	ADSP-21371
ADSP-21375			

TigerSHARC (ADSP-TSxxx) Processors

The name *TigerSHARC* refers to a family of floating-point and fixed-point (8-bit, 16-bit, and 32-bit) processors. VisualDSP++ currently supports the following TigerSHARC processors.

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

2 VISUALDSP++ 4.5 NEW FEATURES AND ENHANCEMENTS

VisualDSP++ 4.5 has new features and enhancements designed to increase productivity and shorten application development cycles. This chapter describes the features and enhancements introduced in VisualDSP++ 4.5.

The information is presented as follows.

- [“VisualDSP++ IDDE” on page 2-2](#)
- [“Assembler” on page 2-11](#)
- [“Features Common to All Compilers and Libraries” on page 2-15](#)
- [“Compiler and Library for Blackfin Processors” on page 2-21](#)
- [“Compiler and Library for SHARC Processors” on page 2-24](#)
- [“Compiler and Library for TigerSHARC Processors” on page 2-28](#)
- [“Linker and Utilities” on page 2-29](#)
- [“Loader and Utilities for Blackfin and SHARC Processors” on page 2-31](#)
- [“VDK” on page 2-36](#)

VisualDSP++ IDDE

VisualDSP++ 4.5 Integrated Development and Debugging Environment (IDDE) introduces:

- “New Processor Support” on page 2-3
- “Support Enhancements” on page 2-3
- “Connectionless IDDE” on page 2-4
- “Session Wizard” on page 2-5
- “Project Wizard Enhancements” on page 2-6
- “Modified Property Pages in Project Options Dialog Box” on page 2-6
- “Load : Compression Page of Project Options Dialog Box” on page 2-7
- “Editor Window Enhancements” on page 2-7
- “Breakpoint Enhancements” on page 2-8
- “Profiling Results in XML Format” on page 2-9
- “Expressions Window and Locals Window Enhancements” on page 2-9
- “Flash Programmer Window Redesign” on page 2-9
- “Power Estimation Analysis” on page 2-10
- “Flash Programmer Window Redesign” on page 2-9

For more information about the VisualDSP++ IDDE, refer to the *VisualDSP++ 4.5 User’s Guide* and online Help.

New Processor Support

The following new processors are supported in VisualDSP++ 4.5.

- Blackfin processors: AD6531, AD6900, AD6901, AD6902, and AD6903
- SHARC processors: ADSP-21371 and ADSP-21375

Refer to the processors' data sheets and hardware reference manuals for information on system configuration, peripherals, registers, and operating modes.

Support Enhancements

The **Help** menu offers two new facilities, **Product Info** and **E-mail Support**.

Product Info generates a complete list of component information and saves it as an XHTML file at

...\My Documents\VisualDSP Projects\ProductInfo.html. The file then displays automatically in the user's default browser.

E-mail Support composes an e-mail that includes the component information. [For more information, see “Technical or Customer Support” on page ii-x.](#)

The **About VisualDSP++** dialog box has been redesigned and now includes a **Versions** tab. You can view a list of your system's binary files (.exe, .dll, and .ocx files) located in the ...\\VisualDSP 4.5, ...\\VisualDSP 4.5\\System, and ...\\etc subdirectories. The binary file versions are displayed together with the version number and creation date of each file. If the version or timestamp information is not available, the corresponding field under the Version heading is left blank.

Connectionless IDDE

In previous versions of VisualDSP++, the IDDE requires you to connect to a debug target by creating a debug session before you could use the IDDE. This connection is required even for tasks where it is not necessary, such as editing a source file. In VisualDSP++ 4.5, this requirement has been eliminated, allowing you to edit your source files and build projects without connecting to a debug target.

You can disconnect from a debug target and connect to another debug target without exiting VisualDSP++. This is useful when you want to change the debug target (an EZ-KIT Lite board or custom board) to a similar model board or to an entirely different board.

Starting VisualDSP++ While Holding Down the Shift Key

Invoking VisualDSP++ while holding down the keyboard's **Shift** key takes you to connectionless IDDE instead of the last session.

Starting VisualDSP++ While Holding Down the Ctrl Key

Another invocation method, available in this version and older software versions, forces you to select a new debug session instead of connecting to the last session. This method involves holding down the keyboard's **Ctrl** key while invoking VisualDSP++ and waiting until the **Session List** dialog box appears. Then select/activate a new debug session. You might do this to overcome a problem such as a corrupted workspace.

Session Wizard

The new **Session Wizard** provides a step-by-step guide for creating a new debug session. Start the wizard by selecting one of the following.

- From the **Session** menu, **New Session**.
- From the **Session** menu, **Session List**. Then click **New Session** from the **Session List** dialog box.
- From the **Session** menu, **Connect to Target**. Then click **New Session** from the **Session List** dialog box.

The first page of the wizard, **Select Processor**, prompts you to specify a target processor. Once you have selected a processor, move to the next page, **Select Connection Type**, and choose a connection from a list of available connection types. The available connections include an EZ-KIT Lite, a simulator, an emulator, or a legacy target. Legacy targets are new to VisualDSP++ 4.5 and denote targets created in VisualDSP++ 4.0 or earlier.

On the next page, **Select Platform**, specify a licensed platform. Both multiprocessor and single platforms are listed. Optionally, select **Show all platforms** to display an unlicensed platform list in grayed-out font. If you select an unlicensed platform, click **Licenses** to open the **Licenses** page of the **About VisualDSP++** dialog box, from which to add a new license.

The **Configurator** button is available when the selected connection type is an EZ-KIT Lite or emulator. If the platform you want is not listed, clicking **Configurator** allows you to define the new platform. After configuring a new platform, the new platform appears in the **Select your platform** list.

Finally, specify a name for the debug session and move to the next page, **Final**, of the wizard. Click **Finish** to close the wizard.

Project Wizard Enhancements

The Project Wizard for a Blackfin-based project has been enhanced, allowing you to generate a Linker Description File in addition to startup code. The wizard displays options related to the user heap, system stack, system heap, external memory, and so on.

A step-by-step procedure, *Generating an .LDF File*, can be found in VisualDSP++ online help.

Modified Property Pages in Project Options Dialog Box

The **Compile**, **Link**, and **Load** property pages (see [Figure 2-1](#) through [Figure 2-3](#)) of the **Project Options** dialog box have been modified to extend flexibility of the tool settings for project builds. For more information, refer to the online Help.

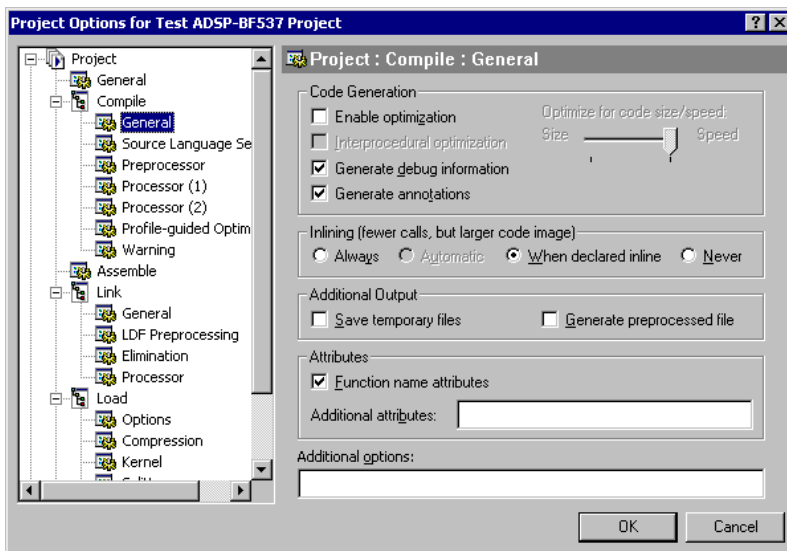


Figure 2-1. Example - Compile : General Page

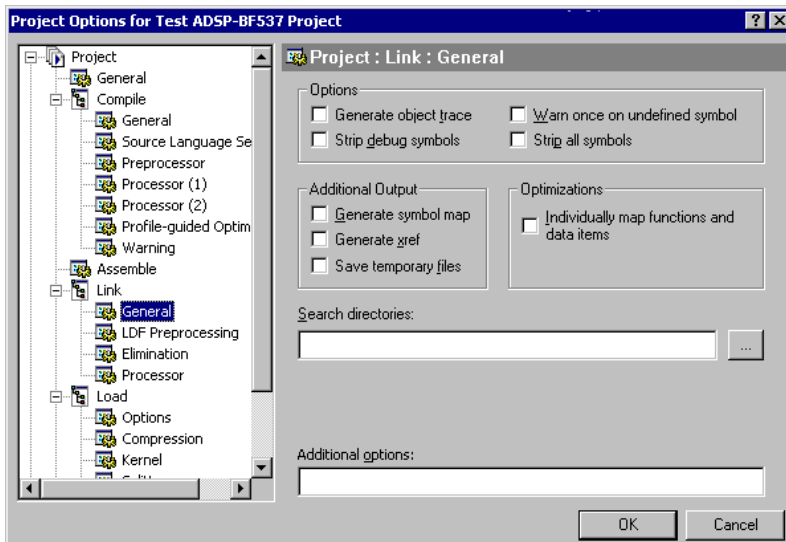


Figure 2-2. Example - Link : General Page

Load : Compression Page of Project Options Dialog Box

For ADSP-BF531/BF532/BF533/BF534, ADSP-BF536, or ADSP-BF537 based application and ADSP-2126x and ADSP-2136x based applications, you can specify zLib compression settings on the **Load : Compression** page of the **Project Options** dialog box (see [Figure 2-4](#)).

Editor Window Enhancements

The compiler can perform a large number of optimizations when generating assembly code. The feedback from the compiler optimizer is provided as annotations made to the assembly file generated by the compiler. Now the IDDE's editor window allows you to view assembly code annotations in C/C++ source files.

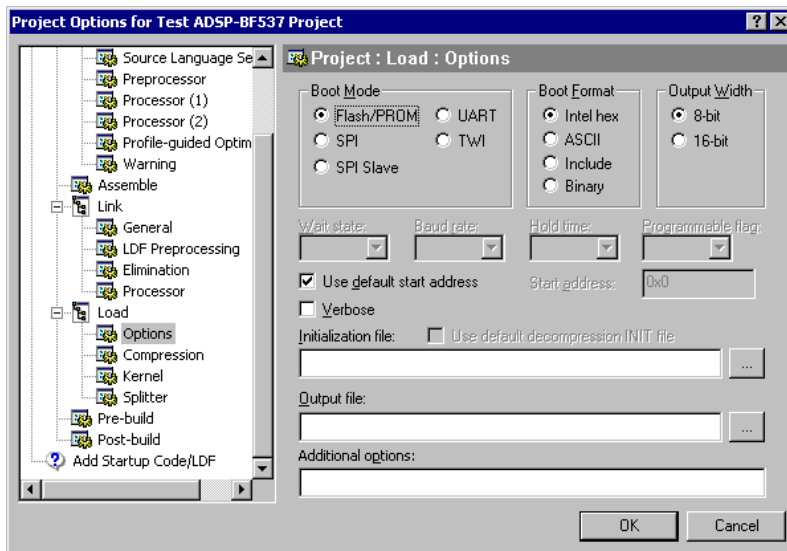


Figure 2-3. Example - Load : General Page

To enable assembly code annotations, choose **Preferences** from the **Settings** menu. In the **Preferences** dialog box, click **General** in the tree control. On the **General** page, select **Enable compiler annotations**, and click **OK**.

To view assembly code annotations, follow the *Viewing Assembly Code Annotations* procedure in the online help. [For more information, see “Extended Optimizer Annotations” on page 3-8.](#)

Breakpoint Enhancements

In VisualDSP++ 4.5, the IDDE can be configured so that automatic breakpoints can be set or not set after a program is loaded. In addition, you can specify the breakpoints to be set after the load and can specify whether the automatic breakpoints are software breakpoints or hardware breakpoints.

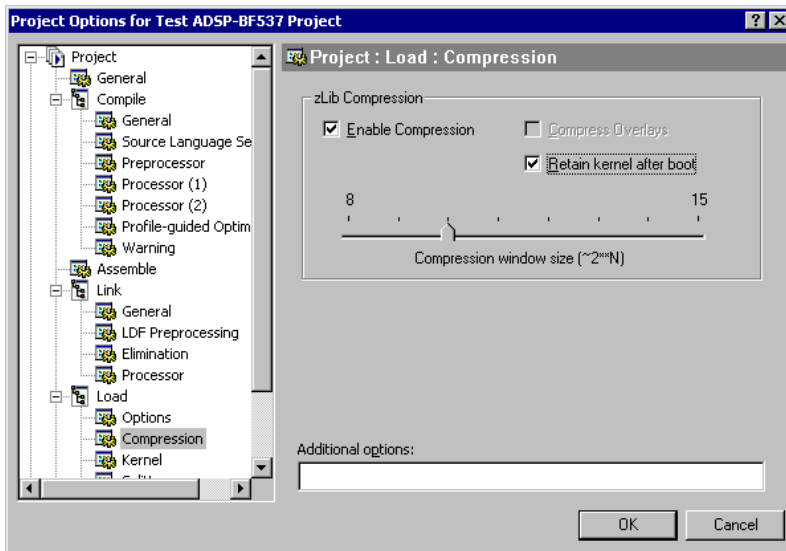


Figure 2-4. Example - Load : Compression Page

Profiling Results in XML Format

You can save the contents of the **Statistical Profiling** window or the **Linear Profiling** window as an `.xml` file.

Expressions Window and Locals Window Enhancements

The **Expressions** window and **Locals** window now allow you to set the display format on a per-expression basis. You can display additional columns showing each expression's type, address, size, and display format.

Flash Programmer Window Redesign

The **Flash Programmer** window has been redesigned to include tabbed pages. This not only reduces its size but also improves usability.

VisualDSP++ 4.5 includes additional Help topics to better describe Flash Programmer functionality, its interface, and how to debug your flash driver.

Power Estimation Analysis

For Blackfin processor applications, you can profile code to estimate the power consumption. Power estimation analysis (also called energy-aware programming) is the ability to use simulation to view the relative impact of instructions, source lines, functions, programs, frequency, and voltage on the application's estimated energy profile. This allows you to make trade-offs that minimize power usage. The technique used to estimate the energy of the application is a partial implementation of a process known as Instruction Level Energy Estimation (ILEE).

The new event logged in the **Linear Profiling** window (**Tools**→**Linear Profiling**→**New Profile**) is called energy units. The units are an accumulation of energy for every instruction that has been executed and is profiled to a PC (program counter). The numbers accumulated in the **Energy Units** column represent the “ranking” of each instruction executed with regard to a power change of the processor's core voltage. The “ranking” numbers were generated by measuring the core voltage while running test code for each instruction. The energy units are instruction-based only and give the energy savings at the core voltage, not the system voltage. Access to different memory sections or bus activity is not considered in any of these numbers. Utilizing these readings as absolute measurements would not be accurate enough considering factors (such as leakage current, temperature, and fabrication process of the chip) that play a part in the application's power. That is why these measurements are referred to as instruction “ranking”.

The gathered information enables you to profile functions to determine where the most energy is being taken up. For the most part, this follows the cycle count profiling. But it is possible to see where some functions require more energy even though the cycle counts can be close. This infor-

mation can help you to strategically place the low power sections of code, utilizing the Dynamic Power capabilities of the Blackfin processor. Then you can find the total power savings of your application at the core by using the numbers listed with some added calculations. The procedure to get this information can be found in the online Help.

Assembler

This section summarizes new macros, directives, switches, and other features common to Blackfin, SHARC and TigerSHARC assemblers.

New features include:

- [“Assembler Feature Macros” on page 2-12](#)
- [“.FILE_ATTR Directive” on page 2-12](#)
- [“.ASCII Directive” on page 2-13](#)
- [“.MESSAGE Directive” on page 2-13](#)
- [“User Control of Assembler Message Severity” on page 2-14](#)

For detailed information on the assembler and preprocessor features, refer to the *VisualDSP++ 4.5 Assembler and Preprocessor Manual* and online Help.

Processor Support

The assembler and preprocessor add support for the AD6531, AD6900, AD6901, AD6902, AD6903, ADSP-21371, and ADSP-21375 processors via the property page (**Project Options**→**Assemble**) and via new arguments to the `-proc` command-line switch. These arguments are:

```
-proc AD6531  
-proc AD6900  
-proc AD6901
```

Assembler

```
-proc AD6902
-proc AD6903
-proc ADSP-21371
-proc ADSP-21375
```

Assembler Feature Macros

VisualDSP++ 4.5 new assembler feature macros are:

Blackfin Processors	
-D__AD6531__=1	Present when running <code>easmb1kfn -proc AD6531.</code>
-D__AD6900__=1	Present when running <code>easmb1kfn -proc AD6900.</code>
-D__AD6901__=1	Present when running <code>easmb1kfn -proc AD6901.</code>
-D__AD6902__=1	Present when running <code>easmb1kfn -proc AD6902.</code>
-D__AD6903__=1	Present when running <code>easmb1kfn -proc AD6903.</code>
SHARC Processors	
-D__ADSP21371__=1 -D__2137x__=1	Present when running <code>easm21k -proc ADSP-21371.</code>
-D__ADSP21375__=1 -D__2137x__=1	Present when running <code>easm21k -proc ADSP-21375.</code>


.FILE_ATTR Directive

The `.FILE_ATTR` directive instructs the assembler to place an attribute in the object file, which can be referenced in the `.LDF` file when linking. The directive also sets the attribute to a value. If the value is omitted, “1” is assumed. The directive’s parameters must follow the rules for naming symbols.

Some examples of the `.FILE_ATTR` directive are:

```
.FILE_ATTR at1;  
.FILE_ATTR at10=a123;  
.FILE_ATTR at101=a123, at102,at103="999";
```

.ASCII Directive

 Blackfin processors only.

The `.ASCII` directive initializes a data location with one or more characters from a double-quoted ASCII string and is equivalent to the `.BYTE` directive. Note that the syntax differs from the `.BYTE` directive:

- There is no = (equal) character
- The string is enclosed in double quotes, not single quotes.

.MESSAGE Directive

The `.MESSAGE` directive alters the severity of an error, warning, or informational message generated by the assembler for all or part of an assembly source file.

The directive can have the following forms.

```
.MESSAGE/qualifier warnid1[,warnid2,...];  
.MESSAGE/qualifier warnid1[,warnid2,...]; UNTIL sym;  
.MESSAGE/qualifier warnid1[,warnid2,...]; FOR n LINES;  
.MESSAGE/DEFAULT/qualifier warnid1[,warnid2,...];
```

where *warnid1*[,*warnid2*,...] is a list of one or more message identification numbers.

A message *qualifier* can be:

- ERROR – change messages to errors
- WARN – change messages to warnings

Assembler

- INFO – change messages to informational
- SUPPRESS – do not output any messages
- RESTORE – change the severity of the messages back to the values they had at the beginning of the source file.
- POP – change the severity of the messages back to what they were prior to the previous .MESSAGE directive.

The simple form of the .MESSAGE directive changes the severity of messages until another .MESSAGE directive is seen. The directive can be placed anywhere in a source file. Messages not associated with a source line can be reported with line number 0. These messages cannot be altered in severity by a .MESSAGE directive but can be altered by the -Werror, -Wwarn, -Winfo, or -Wsuppress assembler switches.

User Control of Assembler Message Severity

The severity of many assembler error messages can be altered. To allow more flexible control over the error message reporting, the assemblers offer a set of command-line switches as listed in [Table 2-1](#).

Table 2-1. New Assembler Command-line Switches

-Winfo <i>number</i> [, <i>number</i> ...]	Selectively turns assembler messages into informational messages.
-Wno-info	Does not display informational assembler messages.
-Wsuppress <i>number</i> [, <i>number</i> ...]	Selectively turns off assembler messages.
-Wwarn <i>number</i> [, <i>number</i> ...]	Selectively turns assembler messages into warnings.
-Wwarn-error	Display all assembler warning messages as errors.

Features Common to All Compilers and Libraries

This section summarizes new switches, pragmas, and other features common to Blackfin, SHARC, and TigerSHARC compilers and libraries.

New features common to all compilers are:

- [“New Switches” on page 2-15](#)
- [“New Pragmas” on page 2-15](#)
- [“File Attributes” on page 2-19](#)
- [“Inlining Control” on page 2-19](#)
- [“Unnamed Struct and Union Fields Within Struct and Union Definitions” on page 2-20](#)

New features common to all run-time libraries are:

- [“Additional Library Routines” on page 2-20](#)

New Switches

[Table 2-2](#) lists the new command-line switches that are common to all compilers.

New Pragmas

- `#pragma loop_unroll(N)`

This pragma instructs the compiler to explicitly unroll the following loop N times before optimizing. This can improve opti-

Features Common to All Compilers and Libraries

Table 2-2. New C/C++ Command-Line Switches Common to All Compilers

Switch Name	Description
-add-debug-libpaths	Some of the libraries included in VisualDSP++ now are supplied with additional variants that include diagnostic or debugging information. These variants can be found in the “debug” subdirectory of the installation directory. The -add-debug-libpaths switch instructs the compiler to link against those debug variants in preference to the usual versions
-enum-is-int	Changes the compiler’s behavior when an enum type is defined, such that one or more of the enumeration’s values exceeds the limits of the int type. Normal behavior is to promote the type of the enumeration to a larger integral type, such that the value can be expressed by the type. This switch prevents such promotion, so that the type is forced to be int.
-implicit-pointers	Forces the compiler to allow assignments between pointer types that are incompatible, according to the ANSI C Standard. Technically, such assignments are illegal and typically cause the compiler to issue an error. The switch causes the compiler to raise a discretionary warning instead.
-check-init-order	C++ mode switch. The switch is used only during application development—not for product releases. It is possible to build applications where the initialization order of objects is undefined due to inter-module dependencies. This switch causes the compiler to plant diagnostic code to detect such cases and issue warnings.
-file-attr name=value	Inserts a file attribute into the generated output file. For more information, see “File Attributes” on page 2-19.
-always-inline	Causes the compiler always to inline any calls to functions declared with the inline qualifier in the same module. For more information, see “Inlining Control” on page 2-19.
-never-inline	Causes the compiler to ignore the inline qualifier, so that no calls to functions defined in the same module get inlined. For more information, see “Inlining Control” on page 2-19.
-overlay	Prevents the compiler from propagating register-clobber information between functions in the same module when a function is called under the auspices of an overlay manager.

VisualDSP++ 4.5 New Features and Enhancements

Table 2-2. New C/C++ Command-Line Switches Common to All Compilers (Cont'd)

Switch Name	Description
<code>-ignore-std</code>	Allows backwards compatibility to earlier versions of VisualDSP C++, which did not use namespace <code>std</code> to guard and encode C++ Standard Library names. By default, the header files and Libraries now use namespace <code>std</code> .
<code>-glite</code>	Generates lightweight DWARF-2 debug information
<code>-pgo-session</code>	Supports PGO in a multi-core or multiprocessor environment. It allows the same source file or the same global symbols to be profiled and optimized independently, without their profiles being merged into a single average behavior.

mization because the compiler recognizes more opportunities for parallelization across the different iterations within the unrolled loop.

- **`#pragma regs_clobbered_call "string"`**

This pragma is the counterpart to `#pragma regs_clobbered`. The pragma is applied to a function call rather than a function definition or declaration and directs the compiler to use a specific clobber set for the call. This allows the use of reduced (or expanded) register sets when calling functions through pointers or when using class methods.

- **`#pragma overlay`**

The compiler normally propagates register-usage information between caller and callee, within the same module, and automatically takes advantage of the information sharing to reduce or expand the clobber set for the functions where such benefit can be obtained. When a called function is mapped into an overlay, the calls can be redirected through an overlay manager, a mechanism

Features Common to All Compilers and Libraries

opaque to the compiler. Therefore, this pragma prevents the compiler from propagating such information, in case an overlay manager is using a normal clobber set invisibly to the compiler.



There is a corresponding `-overlay` switch, which is equivalent to applying the `#pragma overlay` pragma to all functions defined in the module.

- **`#pragma param_never_null`**
`#pragma suppress_null_check`

These pragmas are for use with derived classes, when assigning between pointers to base classes and pointers to derived classes, and when passing such pointers as parameters. Typically, the compiler must verify that the pointer is not a null pointer during the conversion. These pragmas assert that the pointers never are null, which allows the compiler to generate more efficient code.

- **`#pragma always_inline`**
`#pragma never_inline`

These pragmas provide additional control over the compiler inline function calls.

The `always_inline` pragma can be applied to functions defined with the `inline` qualifier; the pragma instructs the compiler to inline function calls regardless that the inlining increases the calling function beyond normally-acceptable limits.

The `never_inline` pragma can be applied to functions that are defined without the `inline` qualifier; the pragma instructs the compiler that calls to this function always must be generated

out-of-line, even if auto-inlining is enabled (the `-Oa` switch) or if the compiler believes that inlining the function call is beneficial. [For more information, see “Inlining Control” on page 2-19.](#)

- `#pragma file_attr(“name=value”)`

This pragma instructs the compiler to place the given file attribute *name/value* pair into the generated output file. [For more information, see “File Attributes”.](#)

File Attributes

The compiler now can place attributes into generated output files. The attributes can be used by the linker to provide additional filtering capabilities for mapping input sections to memory areas. The default .LDF files are enhanced to support this facility. File attributes can be added explicitly, using the `file_attr` pragma or the `-file-attr` switch. The compiler also generates some attributes automatically, for each file compiled. The run-time library is enhanced to exploit attributes as well.

Inlining Control

The compiler supports additional pragmas and switches to provide further control over function inlining:

- `#pragma always_inline`

The pragma indicates that calls to the following function always is inlined.

- `#pragma never_inline`

The pragma indicates that calls to the following function are never

Features Common to All Compilers and Libraries

inlined. This pragma is relevant when the `-Oa` switch is in use, since the `-Oa` switch normally allows the compiler to inline calls to functions without the `inline` qualifier if there is likely to be a benefit.

- **-always-inline**

The switch is equivalent to specifying `#pragma always_inline` on all functions declared in the module that have the `inline` qualifier.

- **-never-inline**

The switch causes the compiler to ignore the `inline` qualifier and to disable inlining.

Unnamed Struct and Union Fields Within Struct and Union Definitions

The compiler supports a GNU C extension, where a `struct` or `union` definition can contain another `struct` or `union` as a member, and the member has no name. In such cases, the compiler promotes the names of the inner `struct` or `union`, making them appear as if they are members of the outer `struct` or `union`.

Additional Library Routines

The run-time libraries support the ISO C99 standard functions `snprintf` and `vsnprintf`.

Compiler and Library for Blackfin Processors

The most notable new features and enhancements of the C/C++ compiler and library for Blackfin processors are:

- “LDF Generator” on page 2-21
- “Long-Long Types in asm Statements” on page 2-22
- “Compiler Builtins for Accessing Memory-Mapped Registers” on page 2-22
- “Speculative Memory Access Pragma” on page 2-23
- “New Processor Support” on page 2-24
- “Core-B Enabling Function” on page 2-24

For detailed information on these features, refer to the *VisualDSP++ 4.5 C/C++ Compiler and Library Manual for Blackfin Processors* and online Help.

LDF Generator



This applies to Blackfin projects. SHARC and TigerSHARC projects should use Expert Linker.

VisualDSP++ contains support for auto-generating a customized .LDF file for your Blackfin project, in addition to a customized CRT (C run-time) startup routine. This considerably reduces the complexity of both the .LDF file and the CRT startup routine because only requested functionality is included. Furthermore, regions of the generated .LDF file are reserved as user-modifiable areas; modifications made in these regions

Compiler and Library for Blackfin Processors

will be preserved by VisualDSP++ during future upgrades, which prevents the .LDF file from becoming obsolete as new features are added to VisualDSP++.

Long-Long Types in asm Statements

The compiler now supports operand specifiers for `asm` statements, which allow you to pass 64-bit types to an `asm`. The “I” constraint and a pair of allocated registers support the feature; the template string can reference a high or low register, using “%nH” or “%nL” as required.

```
long long int res;

int main(void) {
    long long result_ll, x_ll = 123;
    asm(
        "%0H = %1H; %0L = %1L;" :
        "=I" (result_ll) :
        "I" (x_ll)
    );
    res = result_ll;
}
```

Compiler Builtins for Accessing Memory-Mapped Registers

Memory-mapped registers (MMRs) now can be accessed through the following compiler builtins:

- `unsigned short mmr_read16(volatile void *);`
- `unsigned int mmr_read32(volatile void *);`
- `void mmr_write16(volatile void *, unsigned short);`
- `void mmr_write32(volatile void *, unsigned int);`

These builtins ensure that the compiler can distinguish between accesses to MMRs and accesses to arbitrary memory locations. This allows the compiler to generate more efficient code for the former because there are silicon anomalies that must be avoided for arbitrary memory that are not a concern when accessing MMRs. An example follows.

```
#include <cdef_LPblackfin.h>
#include <ccb1kfn.h>

unsigned long inline get_base_addr(void) {
    return (unsigned long)mmr_read32(pSRAM_BASE_ADDRESS);
}
```

Speculative Memory Access Pragma

The `#pragma extra_loop_loads` extends the compiler's support for speculative memory accesses within loops. The pragma allows you to be specific about the loops to which this characteristic applies. In addition, VisualDSP++ 4.5 compilers continue to support the `-extra-loop-load` switch, which applies to all functions in the module.

Typically, the compiler must be careful of not issuing accesses to locations that are not dictated by the source code because the locations may not correspond to valid memory. When speculative accesses are enabled by the pragma or switch, the compiler is permitted to speculatively access memory if this can help the compiler to gain higher-performing code, as if a further extra iteration of the loop is occur.

```
void foo(const int *src, int *dst, int val, int num) {
    #pragma extra_loop_loads
    while (num--)
        *dst++ = *src++ + val;
}
```

New Processor Support

The Blackfin compiler adds support for the AD6531, AD6900, AD6901, AD6902, and AD6903 processors via the **Project : General** property page of the **Project Options** dialog box (**Project**→**Project Options**) and via new arguments to the `-proc` command-line switch. These arguments are:

```
-proc AD6531
-proc AD6900
-proc AD6901
-proc AD6902
-proc AD6903
```

Core-B Enabling Function

The run-time library now supports the `adi_core_b_enable()` function for enabling the second core on dual-core Blackfin processors, such as ADSP-BF561 and ADSP-BF566.

```
#include <ccblkfn.h>
void main(void) {           /* Core A's main() */
    /* Core B is disabled – do general set-up */
    adi_core_b_enable();
    /* Core B now running */
}
```

Compiler and Library for SHARC Processors

For SHARC processors, the most notable new compiler features and enhancements are in the following areas:

- [“External Memory Access Support” on page 2-25](#)
- [“New Interrupt Pragmas” on page 2-25](#)

VisualDSP++ 4.5 New Features and Enhancements

- “Bank Type Qualifiers” on page 2-25
- “New Processor Support” on page 2-26
- “Additional DSP Library Functions” on page 2-27

For more information about these features, refer to the *VisualDSP++ 4.5 C/C++ Compiler and Library Manual for SHARC Processors* and online Help.

External Memory Access Support

On the ADSP-2126x and some ADSP-2136x processors, external memory is not directly accessible. The compiler now supports a `DMAONLY` section qualifier, allowing the linker to verify that data intended for external-memory placement is not mapped to an internal memory region, or vice versa. The compiler provides two functions, `read_extmem()` and `write_extmem()`, for accessing external memory data.

New Interrupt Pragmas

Two new interrupt pragmas, `#pragma interrupt_complete_nesting` and `#pragma interrupt_complete`, have been added to the SHARC compiler. The new pragmas create an interrupt handler function, which can be called directly from the interrupt vector table.

Bank Type Qualifiers

The SHARC compiler now supports bank type qualifiers, which can be used to inform the compiler that data items reside in different memory areas. The bank qualifiers are identifiers associated with pointer data

Compiler and Library for SHARC Processors

types. The compiler compares the identifiers and considers two pointers to refer to different memory (cannot be aliases) if they have different identifiers. For example:

```
void foo(bank("blue") int *src,  
         bank("green") int *dst, int val, int num)  
{  
    while (num--)  
        *dst++ = *src++ + val;  
}
```

The blue and green identifiers have no semantic significance; they are merely strings. The key point is that the compiler can see that `src` and `dst` have different identifiers for their respective pointer types, and it is safe to consider the identifiers are pointing to different memory regions.

New Processor Support

The SHARC compiler adds support for the ADSP-21371 and ADSP-21375 processors via the property page (**Project Options** → **Compile**) and via new arguments to the `-proc` command-line switch. These arguments are `-proc ADSP-21371` and `-proc ADSP-21375`.

Additional DSP Library Functions

The SHARC DSP library has been extended with the following functions.

- New multi-rate filter functions:

Function	Description
<code>fir_decima</code>	FIR-based decimation filter
<code>fir_interp</code>	FIR-based interpolation filter

- Additional fast Fourier transform (FFT) functions, which support a size argument for the FFT and a user-provided twiddle table:

Function	Description
<code>cfft</code>	Complex radix-2 FFT
<code>ifft</code>	Inverse radix-2 FFT
<code>rfft</code>	Real radix-2 FFT
<code>twidfft</code>	Generate FFT twiddle factors

- New FFT magnitude function `fft_magnitude`

For ADSP-2106x processors, there are new vector-based functions complementing the existing scalar functions:

Function	Description
<code>a_compress_vec</code>	Vector A-law compression
<code>a_expand_vec</code>	Vector A-law expansion
<code>biquad_vec</code>	Biquad filter
<code>fir_vec</code>	Vector finite impulse response (FIR) filter
<code>iir_vec</code>	Vector infinite impulse response (IIR) filter
<code>mu_compress_vec</code>	Vector μ -law compression
<code>mu_expand_vec</code>	Vector μ -law expansion

Compiler and Library for TigerSHARC Processors

For ADSP-21xxx SIMD architectures, the following new functions are included in the DSP library:

- SIMD variants of the above FFT functions and FFT magnitude function.
- Additional optimized FFT functions that complement the fast complex FFT function `cfftfc`:

Function	Description
<code>rfftfc_2</code>	Fast real radix-2 fast Fourier transform, which computes two FFTs in parallel
<code>ifftfc</code>	Fast inverse radix-2 fast Fourier transform

- A new magnitude function `fftfc_magnitude` for the fast FFT functions.

Compiler and Library for TigerSHARC Processors

[Table 2-3](#) describes new TigerSHARC compiler switches.

Table 2-3. New TigerSHARC Command-Line Switches

Switch Name	Description
<code>-no-fp-minmax</code>	The MAX and MIN instructions on TigerSHARC processors will return 0xFFFFFFFF if either input value is a NaN. This can result in behavior not anticipated or intended by the original user code. This switch prevents the compiler from using these instructions in floating-point comparisons.
<code>-allow-macs-to-extend-saturation</code>	Instructs the compiler to try to generate multiply-accumulate instructions using saturating add and subtract operations.

Linker and Utilities

The VisualDSP++ 4.5 linker, Linker Description Files, and utility programs are upgraded to operate more efficiently on Blackfin, TigerSHARC, and SHARC processors.

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

- [“Updated List of LDF Keywords” on page 2-29](#)
- [“Linking with Attributes” on page 2-30](#)
- [“RESERVE LDF Command” on page 2-30](#)
- [“DMAONLY Qualifier” on page 2-31](#)

For more information, refer to the *VisualDSP++ 4.5 Linker and Utilities Manual*.

Updated List of LDF Keywords

[Table 2-4](#) lists .ldf file keywords that apply to all supported processors.

Table 2-4. LDF File Keywords Summary

ABSOLUTE	ADDR	ALGORITHM
ALIGN	ALL_FIT	ARCHITECTURE
BEST_FIT	BOOT	DEFINED
DYNAMIC	ELIMINATE	ELIMINATE_SECTIONS
ENTRY	END	
FALSE	FILL	FIRST_FIT
INCLUDE	INPUT_SECTION_ALIGN	INPUT_SECTIONS
INPUT_SECTIONS_PIN	INPUT_SECTIONS_PIN_EXCLUSIVE	KEEP

Linker and Utilities

Table 2-4. LDF File Keywords Summary (Cont'd)

KEEP_SECTIONS	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	PROCESSOR
RAM	RESERVE	RESOLVE
RESERVE_EXPAND	ROM	
SEARCH_DIR	SECTIONS	SHARED_MEMORY
SHT_NOBITS	SIZE	SIZEOF
START	TYPE	
VERBOSE	WIDTH	XREF

Linking with Attributes

An object file can be marked with one or more attributes. The LDF syntax has been extended to support filter expressions to allow selective mapping based on object file attributes.

Refer to Chapter 2, “Linker” of the *VisualDSP++ 4.5 Linker and Utilities Manual* and online Help for more information.

RESERVE LDF Command

This new command can be used to reserve memory regions at specified address and/or of specified size.

Refer to Chapter 2, “Linker” of the *VisualDSP++ 4.5 Linker and Utilities Manual* and online Help for more information.

DMAONLY Qualifier

(ADSP-2126x and ADSP-2136x only) This qualifier is used with the `TYPE` command to specify a memory segment. The linker will validate that external memory on parts without direct memory access.

Refer to Chapter 3, “Linker Description File” of the *VisualDSP++ 4.5 Linker and Utilities Manual* and online Help for more information.

Loader and Utilities for Blackfin and SHARC Processors

The loader utility (`elfloader.exe`) has been enhanced and modified to support new features and new Blackfin and SHARC processors. For details, see the *VisualDSP++ 4.5 Loader and Utilities Manual*.

The loader utility’s modifications and enhancements are described in the following sections.

- [“Loader for Blackfin Processors” on page 2-31](#)
- [“Loader for SHARC Processors” on page 2-33](#)
- [“File Conversion Programs” on page 2-36](#)

Loader for Blackfin Processors

The new features for Blackfin processors include:

- **Support for new processors**

The loader utility supports the AD6531, AD6900, AD6901, AD6902, and AD6903 processors via the **Project** page of the **Project Options** dialog box (**Project** → **Project Options**) and via new arguments to the `-proc` command-line switch. These

Loader and Utilities for Blackfin and SHARC Processors

arguments are:

```
-proc AD6531  
-proc AD6900  
-proc AD6901  
-proc AD6902  
-proc AD6903
```

- **Support for loader file compression**

The loader utility for the ADSP-BF531/BF532/BF533/BF534, ADSP-BF536, and ADSP-BF537 processors supports a loader file (boot stream) compression mechanism known as zLib. VisualDSP++ 4.5 includes a copy of the zLib dynamic link library `zlib1.dll`.

The loader `-compression` switch directs the loader utility to perform the compression from the command line. VisualDSP++ 4.5 also offers a new property page (**Load : Compression**) on the **Project Options** dialog box (**Project** → **Project Options**) to manage compression via the IDDE ([Figure 2-4 on page 2-9](#)).

- **Specification of the compression window width**

The compression window width specifies the maximum compression window size to the compression engine. The value range is 8 to 15 bits, and the default is 9 bits. The compression engine uses the maximum byte count of the compression window size of 2 raised to the power of the specified bit value to perform the compression operation. For example, when the window width is specified as 9 bits, the compression window size is 512 bytes (two raised to the ninth power).

The compression window size can be changed via the

VisualDSP++ 4.5 New Features and Enhancements

Load : Compression page of the **Project Options** dialog box (see [Figure 2-4 on page 2-9](#)) or via the `-compressWS #` command-line switch.

- **Specification of the F, G, and H ports**

The `-pflag PF#`, `-pflag PG#`, and `-pflag PH#` command-line switches specify a 4-bit hex value for ports PF, PG, and PH (respectively). The switches are applicable to the ADSP-BF531/BF532/BF533 (silicon revision 0.2 and newer) and BF534/BF536/BF537 (silicon revision 0.1 and newer) processors.

- **Specification of the block size for zero blocks**

The new `-MaxZeroFillBlockSize #` command-line switch specifies the maximum block byte count for zero-filled blocks. The valid values are from `0x0` to `0xFFFFFFFF0`; the default value matches `-MaxBlockSize #`.

- **Prefix of 0x for data in ASCII output format**

The loader utility distinguishes between output files in ASCII and other formats by prefixing data in ASCII-formatted files with `0x`.

Loader for SHARC Processors

The new loader features for SHARC processors include:

- **Support for new processors**

The loader utility supports the ADSP-21371 and ADSP-21375

Loader and Utilities for Blackfin and SHARC Processors

processors via the property page (**Project Options**→**Load**) and via two new arguments to the `-proc` command-line switch. These switches are `-proc ADSP-21371` and `-proc ADP-21375`.

- **Support for loader file compression**

The loader utility for the ADSP-21261/21266/21267, ADSP-21363/21364/21365/21366/21367/21368/21369, and ADSP-21371/21375 processors supports a loader file (boot stream) compression mechanism known as zLib. VisualDSP++ 4.5 includes a copy of the zLib dynamic link library, `zlib1.dll`.

The loader `-compression` switch directs the loader utility to perform the compression from the command line. VisualDSP++ 4.5 also offers a new dedicated loader property page (**Project Options**→**Load**→**Compression**) to manage the compression from the IDDE.

- **Support for overlay file compression**

The loader utility compresses the data and code from executable (`.dx`) and associated shared memory (`.sm`) files when the `-compression` switch (or **Enable Compression** option on the loader property page) is specified. The `-compressionOverlay` switch must be used (or selected from the loader property page) as the data and code from the associated overlay files are also intended to be compressed. The `-compressionOverlay` switch takes effect when used with the `-compression` switch.

- **Specification of the compression window width**

The compression window width specifies the maximum compression window size to the compression engine. The value range is 8 to 15 bits, and the default is 9 bits. The compression engine uses the maximum byte count of the compression window

VisualDSP++ 4.5 New Features and Enhancements

size of 2 raised to the power of the specified bit value to perform the compression operation. For example, a window width of 9 bits specifies a compression window of 512 bytes (two raised to the ninth power) .

The compression window size can be changed through the loader property page (**Project Options**→**Load**→**Compression**) or through the `-compressWS #` command-line switch.

- **Retaining the kernel with the decompression engine**

The `-retainSecondStageKernel` command-line switch (ADSP-2126x and ADSP-2136x processors only) directs the loader utility to retain the kernel's code and data with the decompression engine. The switch takes effect when used with the `-compression` switch.

- **Support for multiple inputs**

Up to eight ADSP-21367/21368/21369 and ADSP-21371/21375 processors can be clustered together and supported by the VisualDSP++ loader utility. Each input executable (.dxe) can be specified with a processor ID through the `-id#exe=filename` command-line switch, where # is a processor ID for the particular input file, filename. Another command line switch, `-id#ref=N`, can be used to direct the loader to share the same executable with two or more processor IDs in the loader file, where the N is the processor ID which has already had an input executable assigned and # is the other processor ID to share the same executable with the processor ID of N. Using the `-id#ref` switch effectively reduces the size of the loader file. Valid processor IDs are 0 to 7.

File Conversion Programs

VisualDSP++ 4.5 offers two new utility programs for SHARC and Blackfin processors to support a uClinux binary flat format (BFLT). For details, see Appendix B, “Utilities” of the *VisualDSP++ 4.5 Loader and Utilities Manual*. The utilities run from the command line only:

- **elf2flt.exe**

The ELF-to-BFLT file converter program converts a (.dxe) file in Executable and Linkable Format (ELF) to Binary Flat Format (BFLT).

- **fltdump.exe**

The BFLT file dumper program extracts data from a BFLT executable (.bflt) file and yields text showing the BFLT file’s contents.

VDK

The following new features and enhancements have been added to VDK for VisualDSP++ 4.5.

- The new VDK API `GetBlockSize()`
- A new History Logging mechanism
- 40-bit floating-point arithmetic support for SHARC processors
- VDK has been built using the new file attribute `ADI_OS`. This provides greater flexibility in the mapping of VDK code. [For more information, see “File Attributes” on page 2-19.](#)
- Rationalization of the levels which APIs can be called from thread, kernel, or interrupt. The level at which each VDK API can be called is explicitly documented.

See the *VisualDSP++ 4.5 Kernel (VDK) User's Guide* for details.

Example Programs Re-Organized

The example programs that are bundled with VisualDSP++ have been reorganized to make it easier to find examples relevant to your target processor and/or EZ-KIT Lite target board. The basic organization is:

...\

For example, if you are a Blackfin user with an ADSP-BF537 EZ-KIT Lite, "...\Blackfin\Examples\ADSP-BF537 EZ-KIT Lite" is a good starting place when looking for examples. A SHARC user of an ADSP-21369 EZ-KIT Lite would want to start at "...\213xx\Examples\ADSP-21369 EZ-KIT Lite". EZ-Extender cards like the A-V EZ-Extender have their own "target board" subdirectories as well.

There is also a special "target board" called "No Hardware Required". Here one can find more generic examples that can be demonstrated with simulator targets and do not rely on the capability of any particular target board.

Example Programs Re-Organized

3 VISUALDSP++ 4.5 MAJOR CHANGES

This chapter summarizes major changes in VisualDSP++ 4.5 compared with the VisualDSP++ 4.0 release.

The chapter details:

- “New Processor Support” on page 3-2
- “License Server Tools Upgrade” on page 3-3
- “Assembler Changes” on page 3-3
- “Compiler Changes” on page 3-4
 - ✓ “Compiler and Library for Blackfin Processors” on page 3-9
 - ✓ “Compiler and Library for SHARC Processors” on page 3-10
 - ✓ “Compiler and Library for TigerSHARC Processors” on page 3-10
- “Linker Changes” on page 3-11
- “Loader Changes” on page 3-13
- “VDK Changes” on page 3-13
- “Changes to Existing Projects” on page 3-15

Please note that new features and enhancements are listed in Chapter 2, and all obsolete and removed features are listed in Chapter 4.

New Processor Support

New Blackfin processors include the AD6531, AD6900, AD6901, AD6902, and AD6903.

New SHARC processors include the ADSP-21371 and ADSP-21375.

TigerSHARC Simulator Platform Names

The platform names for ADSP-TS201, ADSP-TS202, ADSP-TS203 simulator targets are changed as follows:

For Silicon Rev. 0.x

“ADSP-TS201 Rev. 0.0 Single Processor Simulator” becomes
“ADSP-TS201 Rev. 0.x Single Processor Simulator”.

“ADSP-TS202 Rev. 0.0 Single Processor Simulator” becomes
“ADSP-TS202 Rev. 0.x Single Processor Simulator”.

“ADSP-TS203 Rev. 0.0 Single Processor Simulator” becomes
“ADSP-TS203 Rev. 0.x Single Processor Simulator”.

For Silicon Revisions 1.x and 2.x

“ADSP-TS201 Rev. 1.0 Single Processor Simulator” becomes
“ADSP-TS201 Rev. 1.x/2.x Single Processor Simulator”.

“ADSP-TS202 Rev. 1.0 Single Processor Simulator” becomes
“ADSP-TS202 Rev. 1.x/2.x Single Processor Simulator”.

“ADSP-TS203 Rev. 1.0 Single Processor Simulator” becomes
“ADSP-TS203 Rev. 1.x/2.x Single Processor Simulator”.

License Server Tools Upgrade

The license server tools for VisualDSP++ 4.5 have been upgraded to FlexLM v10.8. It is highly recommended that you upgrade the server tools when installing the VisualDSP++ 4.5 release.

The latest license server tools can be found at:

<http://www.analog.com/processors/resources/crosscore/toolsUpgrades.html>

Assembler Changes

Blackfin .GLOBAL Directive Syntax / Error ea5004

(Blackfin processors only) As described in the *VisualDSP++ 4.5 Assembler and Preprocessor Manual*, the .GLOBAL directive expects the GLOBAL keyword followed by a list of one or more comma-separated symbols.

```
.GLOBAL symbolName1[, symbolName2, ...];
```

The VisualDSP++ 4.5 Blackfin assembler no longer accepts the following undocumented syntax for the .GLOBAL directive.

```
.GLOBAL symbolName[length] ...;
```

This code fragment does assemble in VisualDSP++ 4.0, where the length declarator is ignored by the Blackfin assembler:

```
.BYTE2 _LeftOne[12000];  
.GLOBAL _LeftOne[12000];  
  
.VAR _RightOne[12];  
.GLOBAL _RightOne[12];
```

Compiler Changes

```
.BYTE _WrongOne[5];  
.GLOBAL _WrongOne[50000];
```

In VisualDSP++ 4.5, a length declarator of the `.GLOBAL` directive causes an error. The new error (ea5004) raised by the assembler is a generic “syntax error”.

To revise, do not include the *length* in `.GLOBAL` symbols:

```
.BYTE2 _LeftOne[12000];  
.GLOBAL _LeftOne;  
  
.VAR _RightOne[12];  
.GLOBAL _RightOne;  
  
.BYTE _WrongOne[5];  
.GLOBAL _WrongOne;
```

Compiler Changes

The following sections summarize changes common to Blackfin, SHARC, and TigerSHARC compilers.

- [“Std Namespace is Now Default in C++ Mode” on page 3-5](#)
- [“Revised .LDF Files” on page 3-5](#)
- [“Multiprocessor and Multi-Core Support” on page 3-6](#)
- [“Integrated Section-Placement Mechanisms” on page 3-7](#)
- [“Non-Optimizing Inter-Procedural Analysis” on page 3-7](#)
- [“Optimization Control Pragmas and Inter-Procedural Analysis” on page 3-8](#)

- [“Extended Optimizer Annotations” on page 3-8](#)
- [“Updated C++ Support Libraries and Header Files” on page 3-8](#)

Std Namespace is Now Default in C++ Mode

When compiling C++ code, the compiler now uses the `std` namespace. This means that C++ programs using `std` must have:

```
using namespace std;
```

in scope, otherwise the compiler raises build-time errors.

The old behavior, where the `std` namespace is not used, can be obtained through the new `-ignore-std` switch.

The compiler also issues discretionary warning `cc1756` if a preprocessor macro is defined with the name "std" when compiling C++ sources, as this can cause unexpected warnings when used with C++ Standard Library header files. For example:

```
"file.c", line 1: cc1756: {D} warning: this macro should be
renamed as it can cause spurious errors conflicting
with the C++ Standard Library namespace std
#define std
      ^
```

Revised .LDF Files

The Linker Description Files have received a number of modifications to address certain enhancements. These modification include:

- **File attributes**

The .LDF files now use the `prefersMem` attribute if the attribute is

Compiler Changes

set within any of the objects or libraries involved in the link. The attribute allows you to influence the relative priority of different functions when there is contention for internal memory.

- **C++ exception support**

The implementation of the compiler's C++ exception support has been revised to address some issues, resulting in an additional input section being generated. The section is called `.rtti`; the section must be mapped to data memory.

- **More flexible stack/heap memory arrangement**

The heap and stack layouts have changed to be more flexible but, in some cases, leading to less overall memory available for combined heap and stack.

Multiprocessor and Multi-Core Support

The compiler and run-time libraries include additional support for multiprocessor and multi-core environments. The following features now are available in such environments.

- Inter-procedural analysis (IPA)
- Profile-guided optimization (PGO)
- C++ exceptions

The `-pgo-session` switch has been added to support PGO in a multi-core or multiprocessor environment. It allows the same source file or the same global symbols to be profiled and optimized independently, without their profiles being merged into a single average behavior.

Multi-Core Linking

For dual-core Blackfin processors, you can develop a single application using one build process. This approach uses advanced linker facilities to resolve cross-references between the cores and shared memories.

For details, refer to Appendix A of the *VisualDSP++ 4.5 C/C++ Compiler and Library Manual for Blackfin Processors*.

Integrated Section-Placement Mechanisms

The following interfaces to the compiler's section-placement mechanisms now are integrated to provide a common set of facilities:

- `#pragma section` and `#pragma default_section`
- The `-section` switch
- Platform-specific jump table switches (where available)

For example, the section kinds previously only available using `#pragma default_section` are now available to be used as section identifiers for the compiler switch `-section`.


Target sections for compiler-generated constructs, such as jump tables for C switch statements, now can be specified using the pragmas or the `-section` switch. Existing functionality remains unchanged.

Non-Optimizing Inter-Procedural Analysis

The inter-procedural analysis (IPA) framework now can be invoked without optimizing. The compiler still can analyze and gather information about a module, and the information can be used by other modules in the application. This combination can be obtained by using the switch combination:

Compiler Changes

`-ipa -00`

 The switch order is significant: `-ipa` turns optimization on implicitly; `-00` (capital 0, zero) explicitly disables optimization.

weak_entry Pragma Restriction

Objects that define weak symbols with the `weak_entry` pragma will contain a special symbol that disables interprocedural analysis (IPA) when it is seen by the prelinker. This prevents IPA from being confused, as it does not know which symbol will be used to satisfy the reference at link time. It is anticipated that this restriction will be removed in a future release.

Optimization Control Pragmas and Inter-Procedural Analysis

The IPA framework no longer forces optimization, so `#pragma optimize_off` is honored when using IPA.

Extended Optimizer Annotations

The compiler now generates additional annotations in assembly files to deepen your understanding of the generated code. These annotations can be viewed by looking at the assembly file directly (see the `-S` and `-save-temps` descriptions in the *VisualDSP++ 4.5 C/C++ Compiler and Library Manual* for your target processor) or by viewing the file in the IDDE's editor window. The compiler also emits the annotations in binary form in the ELF file to make the annotations visible in the debugger.

Updated C++ Support Libraries and Header Files

The implementation of the C++ support libraries and associated header files has been updated. One consequence of these changes is that some of the header files do not include the same files as previous releases. These

changes could possibly lead to a difference in existing code behavior if all of the correct header files were not specified for inclusion within the existing code. One example of this is that the `memory` header file no longer includes `iterator` where it did in the previous releases.

Compiler and Library for Blackfin Processors

This section summarizes changes to the Blackfin compiler and library.

“M3-Free” Libraries No Longer Required

VisualDSP++ 4.0 and earlier versions support additional variants of some libraries that ensure the M3 register is reserved for emulator use. These libraries usually are used to link against an application that is built with the `-reserve M3` switch. The VisualDSP++ 4.5 libraries no longer use the M3 register; therefore, these additional variants are no longer required.

The libraries in question are:

- `libdspm3resXXX.dlb` (replaced by `libdspXXX.dlb`)
- `libm3resXXX.dlb` (merged into `libcXXX.dlb`)
- `libm3freeXXX.dlb` (merged into `libcXXX.dlb`)

VisualDSP++ 4.5 includes a set of `libm3freeXXX.dlb` libraries for compatibility with current `.LDF` files, but the library files are empty files and have no effect on linking. However, any tailored `.LDF` file that links against either of the libraries `libdspm3resXXX.dlb` or `libm3resXXX.dlb` must be modified.



Although the run-time libraries no longer use the M3 register, they continue to preserve and restore the register when switching contexts, such as in interrupt dispatchers and in the library functions `setjmp` and `longjmp`.

Compiler Changes

CRT Header File Name Appended with <project_name>

The names of CRT header files produced by the Project wizard have changed from `basiccrt.s` (prior to VisualDSP++ 4.5) to `<project_name>_basiccrt.s`. If you use a source code control (SCC) system to maintain source files, adjust your file lists accordingly.

Compiler and Library for SHARC Processors

There are no changes that are specific to the SHARC compiler and library.

Compiler and Library for TigerSHARC Processors

This section summarizes changes to the TigerSHARC compiler and library.

The behavior of the optimizer for TigerSHARC processors with respect to transforming sequences of multiplies and adds or subtracts into multiply-and-accumulate instructions has changed since the previous release.

The optimizer no longer tries to transform code that uses a saturating add or subtract into a multiply-accumulate; thus, the precision to which the intermediate values saturate is no longer affected. For example, when using a builtin to saturate to 32 bits, it may be undesirable for the compiler to use a multiply-accumulate, which on a TigerSHARC processor saturates to 40 bits for intermediate results and then saturates to 32 bits on extraction from the accumulator.

When enabling remarks with `-wremarks`, the optimizer produces messages similar to the following for operations that previously have used a multiply-accumulate.

```
"test.c", line 16 (col. 5): cc1754: {D} remark: the compiler is  
no longer attempting to use a multiply-accumulate here  
because doing so would cause the saturation behavior  
to be transformed: use -allow-macs-to-extend-saturation  
to give the old behavior.
```

To change the behavior of the optimizer to that of the previous product, use the `-allow-macs-to-extend-saturation` compiler switch.

Linker Changes

This section summarizes changes to the linker, Linker Description Files, and utility programs.

- [“Migration of .LDF Files from Previous Versions of VisualDSP++” on page 3-12](#)
- [“Blackfin-Specific LDF Features” on page 3-12](#)

Refer also to [“Revised .LDF Files” on page 3-5](#).

Migration of .LDF Files from Previous Versions of VisualDSP++

The default Linker Description Files provided with VisualDSP++ 4.5 contain several changes to support new features and enhancements and to provide better default templates for customized .LDF files. VisualDSP++ 4.0 projects containing non-default .LDF files can be safely used within VisualDSP++ 4.5 without the need to update. However, many new features provided with VisualDSP++ 4.5 will not be supported as they require changes to existing .ldf files. These features are listed in the following sections.

Update projects containing customized .ldf files from VisualDSP++ 4.0 and earlier versions. The modifications applied to the previous .ldf file should be migrated into a copy of the default .ldf file provided with VisualDSP++ 4.5.

Blackfin-Specific LDF Features

The following new features apply to Blackfin processors.

Project Wizard, CRT Generation

The modified **Project Wizard** provides support for the generation of a Linker Description File for Blackfin-based projects, in addition to startup code. The wizard displays options related to the user heap, system stack, system heap, external memory, and so on. This feature is not supported for projects with customized .ldf files derived from VisualDSP++ 4.0 and earlier.

Loader Changes

These VisualDSP++ 4.5 loader changes apply to Blackfin and SHARC processors:

- No zero padding for 16-bit output for silicon revision 0.3 for the ADSP-BF531, ADSP-BF532, and ADSP-BF533 processors.
- Support for multiple initialization blocks.

VDK Changes

When porting existing VDK projects from VisualDSP++ 4.0 to VisualDSP++ 4.5, the following changes need to be taken into account:

- Message queues are no longer included in thread structures if the total number of messages in the system is specified to be zero, even in cases where “messaging allowed” has been specified for particular thread types.
- Support for VDK C/C++ ISRs was introduced in VisualDSP++ 4.0. However, on SHARC processors the compiler pragmas did not provide all of the functionality required to declare the functions as ISRs. An update to VisualDSP++ 4.0 introduced new pragmas to address this issue, and these pragmas should also be used in VisualDSP++ 4.5. Creating a new ISR in VisualDSP++ 4.5 produces a skeleton ISR that contains the correct pragmas.
- The run-time libraries now use the standard namespace. Commensurate changes are required in source code for VDK projects. [For more information, see “Std Namespace is Now Default in C++ Mode” on page 3-5.](#)

VDK Changes

- The Blackfin VDK exception handler no longer uses `JUMP.X` to switch execution to the `UserExceptionHandler`. This is because if expansion to a long jump is required, there is no way of preventing the trashing of the `P1` register. Both the VDK exception handler and the `UserExceptionHandler` must now be placed in the same memory area, otherwise a linker error will be produced.
- The project's `.LDF` file must be updated to be compatible with VisualDSP++ 4.5. The most reliable way to upgrade the `.LDF` file is to start with the default VisualDSP++ 4.5 `.LDF` file for the target processor, then customize the `.LDF` file as required. Some notable changes to the `.LDF` files are:
 - ✓ The new microkernel libraries must be linked in, named `TMK-<processor_name>.dlb`.
 - ✓ Variables that must be placed into internal memory on SHARC processors are now mapped to a new section, `internal_memory_data`, which must be mapped to a suitable output section in the `.LDF` file.
 - ✓ `RESERVE` now is used to allocate space for heaps in default `.LDF` files for Blackfin processors. This is not a required change but is a more efficient way to allocate memory for heaps.
 - ✓ The run-time libraries no longer include `M3` free/reserved libraries, and any usage has been removed from the default VDK `.LDF` files. [For more information, see ““M3-Free” Libraries No Longer Required” on page 3-9.](#)
 - ✓ An additional input section called `.rtti` is generated for the compiler's C++ exception support, which must be mapped to data memory in the `.LDF` file.

Changes to Existing Projects

This section lists changes to projects started prior to VisualDSP++ 4.5.

SPI_MS Macro

For ADSP-21161 processors, reference the “Master/Slave Mode Bit” on the SPI control register using the macro `SPI_MS` instead of `MS`. The macro definitions for the SPI control registers are contained in `def21161.h`. This macro was changed because `MS` is also recognized as a keyword by the assembler (for the multiplier signed bit).

SSL Libraries in New Location

The location of the debug SSL libraries has moved from the base `lib` directory to the `debug` subdirectory. Adjust the existing `.LDF` files or add the debug directory to the search path.


The compiler’s `-add-debug-libpaths` command-line switch is added in order to do this.

MMR Definitions Include File Uses of “volatile void” is Replaced

In prior versions of VisualDSP++, various macros for MMRs that hold addresses were declared using “(volatile void **)” casts in `cdefBF5*.h` and associated standard include files (for example, the pointers to CPLB addresses `pDCPLB_ADDR0` through `pDCPLB_ADDR15` and Event Vector Table addresses `pEVT0` through `pEVT16`).

Changes to Existing Projects

In VisualDSP++ 4.5, these have been modified to use casts to “(void *volatile *)”. That is, they use a pointer to a volatile pointer, rather than a volatile pointer to pointer which was incorrect.

 Some code that compiled previously without error may trigger compiler error cc0513 when built with VisualDSP++ 4.5.

For example, the following would compile without error in VisualDSP++ 4.0.

```
#include <cdefBF532.h>
void foo (volatile void ** fielder) {
    *pEVT0 = *fielder;
}
```

However, when built with VisualDSP++ 4.5 it will result in an error of the following type:

```
"t.c", line 5: cc0513: error: a value of type "volatile void *"
    assigned to an entity of type "void *" cannot be
    *pEVT0 = *fielder;
    ^
```

To resolve these problems, modify the source to make it compatible with the changed macro type or enable the legacy definitions. The preferred resolution is to modify the source because the incorrect use of the volatile qualifier in the prior definitions (still available for legacy purposes) may give rise to run-time problems, especially when an application is built as a Release version.

To modify the source, change affected code from using “volatile void **” types to “void *volatile *” types. In our small example, this means changing the type of parameter fielder, as follows:

```
#include <cdefBF532.h>
void foo (void *volatile * fielder) {
```

```
*pEVT0 = *fielder;  
}
```

If required, to enable the legacy behavior, define the macro “_USE_LEGACY_CDEF_BEHAVIOUR” before including the `cdef*.h` files. This can be done, for example, using the compiler’s `-D` switch by adding `-D_USE_LEGACY_CDEF_BEHAVIOUR` to the project compiler additional options. Alternatively, the legacy behavior macro can be defined in the source by adding “`#define _USE_LEGACY_CDEF_BEHAVIOUR`” before `#include` statements.

Changes to Existing Projects

4 VISUALDSP++ 4.5 OBSOLETE OR REMOVED FEATURES

This chapter describes the features that have been deprecated or removed in VisualDSP++ 4.5. 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++ 4.0. Similarly, new projects created with VisualDSP++ 4.5 cannot be used by earlier versions of the tools.

This chapter contains lists of obsolete or removed features:

- “Discontinued Processor Support” on page 4-2
- “VisualDSP++ IDDE” on page 4-2
- “Compilers and Libraries” on page 4-2
- “VDK” on page 4-3

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

You may also want to visit the Software Tools Upgrades Web site (<http://www.analog.com/processors/resources/crosscore/toolsUpgrades.html>) to check if an update is available for your VisualDSP++ 4.5 installation. Installing an update ensures that your software contains the latest processor support, example code, and bug fixes.

Discontinued Processor Support

VisualDSP++ 4.5 does not provide support for ADSP-218x and ADSP-219x processors. Therefore, refer to VisualDSP++ 3.5 user documentation and the online Help if you need information on how to develop and run DSP projects on ADSP-21xx processors. VisualDSP++ 3.5 continues to be available for ADSP-218x and ADSP-219x developers.

VisualDSP++ IDDE

VisualDSP++ 4.5 does not support the Apex-ICE emulator.

Compilers and Libraries

This section contains information about all removed or deprecated features within the compilers and libraries:

- [“Removed Command-Line Switches” on page 4-2](#)
- [“Deprecated Pragmas” on page 4-3](#)

Refer to Chapter 3, [“VisualDSP++ 4.5 Major Changes”](#) for more information about the changes to the C/C++ compilers and run-time libraries.

Removed Command-Line Switches

The following compiler command-line switches have been removed from VisualDSP++ 4.5.

VisualDSP++ 4.5 Obsolete or Removed Features

Switch	Description
-default-linkage	Removed. In VisualDSP++ 4.5, C linkage is always the default.
-write-files	Removed
-write-opts	Removed

Deprecated Pragmas

The `#pragma retain_name` has been deprecated. The compiler now obtains the required information from the linker at link time, making the pragma redundant.

VDK

VDK no longer supports 0.x silicon for TS20x processors. In order to use VDK on these processors, you must use silicon revision 1.0 or higher.

See [“VDK Changes” on page 3-13](#) for information on the kernel changes.

