

VISUALDSP++[®] 4.0

Product Release Bulletin

Revision 1.0, January 2005

Part Number
82-000420-06

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



Copyright Information

© 2005 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, Blackfin, SHARC, TigerSHARC, EZ-KIT Lite, and VisualDSP++ 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	xi
Supported Processors	xi
Product Information	xii
MyAnalog.com	xii
Processor Product Information	xii
Related Documents	xiii
Online Technical Documentation	xiv
Accessing Documentation From VisualDSP++	xv
Accessing Documentation From Windows	xv
Accessing Documentation From the Web	xvi
Printed Manuals	xvi
VisualDSP++ Documentation Set	xvi
Hardware Tools Manuals	xvi
Processor Manuals	xvi
Data Sheets	xvii

CONTENTS

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

INTRODUCTION

Product Release Description	1-2
VisualDSP++ 4.0 System Requirements	1-3
Platform and Processor Support	1-3

VISUALDSP++ 4.0 MAJOR CHANGES

New Processor Support	2-2
Installer Changes	2-2
License Changes	2-3
ICE Test	2-3
Changes Common to All Compilers	2-4
Multiline Strings Disabled by Default	2-4
Extended I/O Support	2-4
Support for Silicon Revisions in the Libraries	2-4
Header File time.h	2-6
New Section Used for C++ Virtual Function Tables	2-7
Order of Side-Effects Changed in "Undefined" Expressions	2-8
".o" No Longer Supported as a Suffix for Object Files	2-8
Compiler and Library for Blackfin Processors	2-8
Long Double Now a 64-Bit Floating-Point Data Type	2-8
Variable Size for Double Data Type	2-9
Smaller, Faster Stdio Library	2-10
Parameters to main() Initialized Differently	2-10

Overlay/DMA Managers Must Ensure Loop Counters Are Zero	2-10
System Register Read/Write Built-In Functions Have Different Prototypes	2-11
New noncache_code Section Needed	2-11
Instrumented Profiling Not Supported in a Multithreaded Environment	2-11
Single- and Dual-Core Processors Use Different Library Builds	2-12
Default CRTs Changed	2-12
Compiler and Library for TigerSHARC Processors	2-14
Stack Must Be in Internal Memory	2-14
Changes to the I/O Library	2-14
C/C++ Run-Time Library Naming Conventions	2-14
Compiler and Library for SHARC Processors	2-15
IDDE Changes	2-15
Loader Changes	2-16
Linker Changes	2-16

NEW FEATURES AND ENHANCEMENTS

VisualDSP++ IDDE	3-2
New Processor Support	3-2
New ICE Features	3-2
New Installation Options	3-3
New “Authorize Script” Option	3-3
Assembler	3-4

CONTENTS

Assembler Feature Macros	3-4
.SEPARATE_MEM_SEGMENTS Directive	3-5
Special Assembler Operators	3-5
New Features Common to All Compilers	3-6
New Switches	3-6
New Pragmas	3-7
New Built-In Functions to Control Branch Prediction	3-11
New -Og Switch for Optimizing and Debugging	3-12
Support for Gathering Cycle-Counts	3-12
Bank Type Qualifiers	3-13
Predefined Compiler Macros	3-13
Compiler and Library for Blackfin Processors	3-14
C/C++ Compiler Command-Line Switches	3-15
Workaround <workaroundid>[,<workaroundid> ...]	3-16
Endian-Swapping Intrinsics	3-16
System Built-In Functions	3-17
Misaligned Data Built-In Functions	3-18
Long Double Floating Point Type	3-19
Run-Time Libraries and Start-Up Files	3-21
New C Library Functions	3-21
New DSP Run-Time Library Functions	3-22
CRT Generation	3-23
Assembly Annotations	3-23
New Clipping Fractional Shift Functions	3-24

Compiler and Library for TigerSHARC Processors	3-24
Compiler Command-Line Switch	3-25
New Math Built-In Functions	3-25
New TigerSHARC-Specific Pragma	3-25
Miscellaneous Changes	3-26
Compiler and Library for SHARC Processors	3-26
Access to System Registers	3-26
Predefined Compiler Macros	3-27
Switch Table Placement Control	3-28
New Default Run-Time Headers	3-28
Linker and Utilities	3-30
Modified Link Page in Project Options Dialog Box	3-31
Support for Specifying Two Buffers in Different Memory Segments	3-32
Linker Command-Line Switches	3-32
Updated List of LDF Keywords	3-32
External Execution Packing in SHARC Processors	3-34
Expert Linker Features	3-35
Adding a Memory Segment	3-36
Managing Output Section Properties	3-36
Managing Packing Properties	3-39
Managing Overlay Properties	3-40
Managing Shared Memory Properties	3-41
Memory_INITIALIZER Utility	3-43
Migration of LDFs From Previous Versions of VisualDSP++ ...	3-44

CONTENTS

Data Section for C++ Virtual Tables	3-44
Blackfin-Specific Features	3-45
New Features Requiring .LDF Support	3-45
New Features in the VisualDSP++ 4.0 Default Template LDFs	3-46
TigerSHARC C/C++ Run-Time Library Naming Conventions	3-48
Loaders and Splitter	3-49
Blackfin Loader Features	3-49
SHARC Loader Features	3-50
VDK	3-50

OBSOLETE OR REMOVED FEATURES

Discontinued Processor Support	4-2
VisualDSP++ IDDE	4-2
Assembler and Preprocessor	4-2
Compiler and Library for SHARC Processors	4-2
Compiler and Library for TigerSHARC Processors	4-3
Linker	4-4
VCSE	4-4
VDK	4-4

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++ 4.0 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++ 4.0 and previous VisualDSP++ releases.

For details, refer to the VisualDSP++ 4.0 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++ 4.0.

Manual Contents

This manual consists of:

- Chapter 1, “[Introduction](#)”
Describes VisualDSP++ 4.0 and its benefits, provides the minimal system requirements for running the product, and lists the supported processors.
- Chapter 2, “[VisualDSP++ 4.0 Major Changes](#)”
Describes major changes in VisualDSP++ 4.0 compared to VisualDSP++ 3.5 release.
- Chapter 3, “[New Features and Enhancements](#)”
Describes what is new in the VisualDSP++ 4.0 IDDE, assembler, compiler, linker, loader, and documentation. Also describes the new features in the Expert Linker (EL) and the VisualDSP++ Kernel (VDK).
- Chapter 4, “[Obsolete or Removed Features](#)”
Describes the removed/obsolete features in VisualDSP++ 4.0 (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 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++ 4.0 release for Blackfin (ADSP-BFxxx), 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++ 4.0 Getting Started Guide*
- *VisualDSP++ 4.0 User's Guide*
- *VisualDSP++ 4.0 Assembler and Preprocessor Manual*
- *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors*
- *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for TigerSHARC Processors*
- *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for SHARC Processors*
- *VisualDSP++ 4.0 Linker and Utilities Manual*
- *VisualDSP++ 4.0 Loader Manual*

Product Information

- *VisualDSP++ 4.0 Kernel (VDK) User's Guide*
- *VisualDSP++ 4.0 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>

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 4.0 (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 (tool-bar 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**.

Product Information

Accessing Documentation From the Web

Download manuals in PDF format 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. 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 ellipsis; read the example as an optional comma-separated list of <i>this</i> .

Notation Conventions

Example	Description
.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.
	Note: For correct operation, ... A Note provides supplementary information on a related topic. In the online version of this book, the word Note appears instead of this symbol.
	Caution: Incorrect device operation may result if ... Caution: Device damage may result if ... 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.
	Warning: Injury to device users may result if ... A Warning identifies conditions or inappropriate usage of the product that could lead to conditions that are potentially hazardous for devices users. 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, 4.0. 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++ 4.0 System Requirements” on page 1-3
- “Platform and Processor Support” on page 1-3

Product Release Description

VisualDSP++ is Analog Devices project management and development environment for Digital Signal Processing (DSP) applications. VisualDSP++ 4.0 successfully integrates a graphical user interface and code generation and debugging tools, enabling programmers to move easily between editing, building, debugging, and deployment of final products.

The VisualDSP++ 4.0 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++ 3.5, this software release incorporates a number of new features and enhancements, as described in Chapter 3, [“New Features and Enhancements”](#).

VisualDSP++ 4.0 System Requirements

To install and run VisualDSP++ 4.0, 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

Note: Windows NT, 98, and ME are not supported.

- At least 750 MB of available hard drive space
- At least 256 MB of RAM
- CD-ROM drive
- Internet Explorer 4.01 or later

Platform and Processor Support

The following is the list of Analog Devices, Inc. processors supported in VisualDSP++ 4.0 release.

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
------------	------------	------------	------------

Platform and Processor Support

SHARC (ADSP-21xxx) Processors

The name “SHARC” refers to a family of high-performance, 32-bit, floating-point processors. 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	ADSP-21366	ADSP-21367
ADSP-21368	ADSP-21369		

Blackfin (ADSP-BFxxx) Processors

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

ADSP-BF531	ADSP-BF532 (formerly ADSP-21532)
ADSP-BF533	ADSP-BF535 (formerly ADSP-21535)
ADSP-BF534	ADSP-BF537
ADSP-BF536	ADSP-BF539
ADSP-BF538	ADSP-BF566
ADSP-BF561	AD6532

2 VISUALDSP++ 4.0 MAJOR CHANGES

This chapter summarizes major changes in VisualDSP++ 4.0 compared with the VisualDSP++ 3.5 release.

The chapter details:

- “New Processor Support” on page 2-2
- “Installer Changes” on page 2-2
- “License Changes” on page 2-3
- “ICE Test” on page 2-3
- “Changes Common to All Compilers” on page 2-4
- “Compiler and Library for Blackfin Processors” on page 2-8
- “Compiler and Library for TigerSHARC Processors” on page 2-14
- “Compiler and Library for SHARC Processors” on page 2-15
- “IDDE Changes” on page 2-15
- “Loader Changes” on page 2-16
- “Linker Changes” on page 2-16

Please note that new features and enhancements are listed in Chapter 3, and all obsolete and removed features are listed in Chapter 4.

New Processor Support

The following is the list of new processors supported in VisualDSP++ 4.0.

SHARC (ADSP-21xxx) Processors:

ADSP-21366	ADSP-21367	ADSP-21368	ADSP-21369
------------	------------	------------	------------

Blackfin (ADSP-BFxxx) Processors:

ADSP-BF536	ADSP-BF537
ADSP-BF538	ADSP-BF539
ADSP-BF566	ADSP-BF534

Installer Changes

Emulator and EZ-KIT Lite device drivers are now copied to the VisualDSP++ installation directory and the installer makes Windows aware of this location. Thus, the original distribution media is no longer needed to install the emulator drivers when your hardware is connected to your computer. Upon uninstallation, the driver may be removed from your system, negatively impacting any remaining installation of VisualDSP++. Should this occur, use the **Maintain this installation's Touch this installation** feature (VisualDSP++ 4.0), or perform a repair installation (VisualDSP++ 3.5 and earlier).

License Changes

In VisualDSP++ 4.0, three types of licenses are available: TST (evaluation), KIT (evaluation), and ADI (permanent).

A TST (test drive) license provides unlimited (unrestricted) access to VisualDSP++ for emulation and simulation. You must register the software to receive a TST serial number, which expires 90 days after installation. After the 90 days, the software is inoperable.

A KIT license provides 10 days to register and validate the installation. Once validated, the KIT licence extends to 90 (ninety) days of a full evaluation in total. At the end of the evaluation period, unless you have added a permanent license, simulator and emulator connections become prohibited and the size of the user program is limited.

An ADI license provides 30 (thirty) days to register and validate the installation for a permanent use. If you fail to register and validate your installation during the 30-day evaluation period, your copy of VisualDSP++ 4.0 becomes inoperable.

ICE Test

ICETest is no longer a stand-alone utility. Comparable functionality is now available as part of the ICE configurator, which is no longer required to establish a debug target for Blackfin (ADSP-BF537, ADSP-BF561) and TigerSHARC (ADSP-TS101, ADSP-TS201) EZ-KIT Lites.

Changes Common to All Compilers

This section summarizes changes common to all compilers.

Multiline Strings Disabled by Default

In VisualDSP++3.5, multiline strings were accepted as part of gcc-compatibility. For VisualDSP++4.0, multiline strings are disabled by default, although the previous behavior may be restored with the new `-multiline` switch. The new `-no-multiline` switch explicitly specifies the default behavior.

Extended I/O Support

Three additional fields have been added to the `DevEntry` struct defined in the `device.h` header file. These facilitate a mechanism by which devices can claim the three standard streams, `stdin`, `stdout`, and `stderr` on registration. Projects that use a non-default I/O configuration which declare structs of this type may need to be modified to include the three additional fields. See the section "Extending I/O Support To New Devices" in the compiler manual for full details.

Support for Silicon Revisions in the Libraries

VisualDSP++ now includes libraries built for different silicon revisions of the supported processors. Located in `<dir>`, the libraries are built without any workarounds enabled. Within the `VisualDSP\<family>\lib` directory there are subdirectories named `<processor>_rev_<revision>` that contain libraries built for the specific revision, with the appropriate workarounds for that specific silicon revision enabled.

One single library directory may support more than one specific silicon revision. A set of libraries is included that supports all suitable workarounds valid for any of the revisions of a particular target. For Blackfin these libraries are located in the top-level `lib` directory (along with the non-workaround libraries) and have a “y” suffix to identify them.

Processor revisions are selected for a VisualDSP++ IDDE project from a menu available in the main project-options window. If you are using the tools from a command line, select a silicon revision target with the `-si-revision` switch.

The `-si-revision` switch (refer to compiler manuals) can be used to specify a silicon revision; VisualDSP++ will use the appropriately-built library when linking the application for the specified silicon revision.

The `-si-revision` switch supports two special revisions:

- “any” may be used to build applications that need to run on all revisions of the target processor; all known workarounds for the target processor will be enabled.
- “none” indicates that no silicon revisions should be enabled during building. No anomaly workarounds will be applied.

By default, when `-si-revision` is not specified on the command line, the libraries built with workarounds enabled for the most recently-supported version of the processor are linked against.

The meaning of the `-workaround all` switch has changed as a result, and normally `-si-revision any` should be used instead. This is because, depending on your platform, the `-workaround all` switch may include some workarounds that are not necessary for any silicon revision of the target, whereas `-si-revision any` will include only workarounds that are required for at least one silicon revision of the target.

Header File `time.h`

Earlier releases of VisualDSP++ did not support the `time.h` header file, which defines the C standard's data types, functions, and macros for manipulating date and time. The `time.h` header file is now fully supported in this release, apart from:

- Time zones
- The Daylight Saving flag (`tm_isdst`) in the structure `struct tm`
- The `time` function always returns -1, to indicate that the current calendar time is not available

The `time.h` header file sets the macro `CLOCKS_PER_SEC` to the number of processor cycles per second; usually, however, the processor speed is a property of a particular chip and it is therefore strongly recommended that the value of this macro is verified independently before it is used by an application. The macro is set by one of the following (in descending order of precedence):

- From the compile-time switch `-DCLOCKS_PER_SEC=<definition>`
- Via the System Services Library (Blackfin only)
- From the **Processor speed (MHz)** box in the VisualDSP++ **Project Options** dialog box, **Compile** tab, **Processor** category
- Via the header file `cycles.h`

Further details about this header file and also about additional facilities for benchmarking an application are available in the compiler manual.

New Section Used for C++ Virtual Function Tables

When virtual functions are used in C++ applications, the compiler generates read-only tables to implement method inheritance. Previous releases of VisualDSP++ have placed these tables into the default data section. The compiler now places these tables into their own section, so that they can be more easily mapped to read-only memory, or be positioned appropriately on a multicore or multiprocessor system. The section names used are:

Family	Section Name
SHARC	seg_vtbl
TigerSHARC	vtbl
Blackfin	vtbl

Existing C++ applications that use custom LDFs should be updated to map these sections accordingly. They can be mapped to read-only memory, and need not be contiguous.

A new compiler switch, `-section vtbl=<secname>`, will direct the compiler to place virtual-function tables into the specified section `<secname>`. This switch can be used to obtain the previous behavior, for example:

```
ccblkfn -proc ADSP-BF533 -c++ -section vtbl=data1 prog.cpp
```

It can also be used when building multiple applications in the same memory space (such as for dual-core systems) in order to keep the virtual-function tables for each application separate.

Order of Side-Effects Changed in "Undefined" Expressions

The ANSI C Standard defines the order of expression evaluation to be undefined in statements such as:

```
i = ++i + 1;
```

Use of such statements, where an object (i) is modified more than once between sequence points, is bad programming and can lead to obscure program failures.

In VisualDSP++ 4.0, the compiler evaluates such expressions differently, both in order to improve performance and to meet user expectations.

The compiler now issues discretionary warning cc1639 when such undefined sequences are detected.

".o" No Longer Supported as a Suffix for Object Files

Object files are now required to have ".obj" suffix.

Compiler and Library for Blackfin Processors

This section summarizes changes to the Blackfin compiler and library.

Long Double Now a 64-Bit Floating-Point Data Type

Previous releases of VisualDSP++ have only supported 32-bit floating-point data types, as `float` and `double`. `Long double` was supported as an undocumented alias for `double`.

For VisualDSP++ 4.0, the `long double` data type is a 64-bit double-precision IEEE-754 floating-point data type.

The new library `libf64ieee*.dlb` provides the emulation support for this new data type, and must be added to any customized LDFs.

Variable Size for Double Data Type

Previous releases of VisualDSP++ have supported the `double` data type as a 32-bit single-precision floating-point type. VisualDSP++ 4.0 retains this as a default, and also adds the option of supporting `double` as a 64-bit double-precision floating-point type, using the `-double-size-64` switch (refer to the compiler manual). The default behavior may be explicitly specified via the `-double-size-32` switch.

Object files are marked to indicate whether they have been built with `double` as a 32- or 64-bit data type, and the two kinds of object file may not be linked together in a single application. The `-double-size-any` switch may be used when building an object that has no reliance on floating-point types; such an object may be linked into 32-bit floating-point applications or into 64-bit floating-point applications.

Existing object files from earlier releases of VisualDSP++ may be linked with 32-bit floating-point objects created by VisualDSP++ 4.0. If objects from earlier releases of VisualDSP++ are linked with 64-bit floating-point objects created by VisualDSP++ 4.0, a warning will be produced, because there is a potential mismatch in floating-point sizes.

Smaller, Faster Stdio Library

The default I/O library used by VisualDSP++ 4.0 is a smaller, faster library than the one used in previous releases. The new I/O library is not as full-featured as the library used in previous releases. All commonly used functionality is present, but some features, such as wide-character I/O, are omitted. If such features are required, the `-full-io` switch can be used to specify the older library from previous releases. The functional differences between the two libraries are documented in Chapter 3 of the compiler manual.

Parameters to `main()` Initialized Differently

If support for `argc/argv` is enabled (refer to Chapter 1, “Support for `argc/argv`,” in the compiler manual), the `argc` and `argv` parameters are now initialized in line with traditional UNIX conventions: the first token in the argument string is stored in `argv[0]`, the second in `argv[1]`, and so on. In previous releases, `argv[0]` was initialized to the empty string.

Overlay/DMA Managers Must Ensure Loop Counters Are Zero

In previous releases, the compiler ensured that loop counters were always zeroed on exit from a hardware loop. For performance reasons, this is no longer the case. If your application uses overlays, or uses DMA to move different blocks of code into a common area for execution, you must ensure that loop counters are zeroed each time code is moved. This is necessary so that different code blocks do not encounter a live “end of loop” address that was set up by a previously-resident block of code.

System Register Read/Write Built-In Functions Have Different Prototypes

The built-in functions for reading and writing system registers have different prototypes from previous releases; they now read and write unsigned values instead of signed values. The modified prototypes for these functions are documented in Chapter 1 of the compiler manual under the section “System Built-In Functions.”

New `noncache_code` Section Needed

The new section `noncache_code` is for executable items that may not be placed into the SRAM/Cache part of L1 Instruction Memory. This change applies to the run-time library support for:

- C++ Exception-handling
- Memory Initialization on ADSP-BF535 and AD6532 processors
- Use of `_l1_memcpy` and `_memcpy_l1` (refer to the description of these functions in Chapter 3 of the compiler manual)

In the above cases, executing code may need to change the SRAM/Cache attributes temporarily. Code executing from within the SRAM/Cache part of L1 instruction memory cannot change these attributes, hence the requirement for the new section.

Instrumented Profiling Not Supported in a Multithreaded Environment

Instrumented profiling is not supported in a multithreaded environment, and, therefore, the `-threads` switch will give an error if any of the `-p`, `-p1` or `-p2` switches is also specified.

Single- and Dual-Core Processors Use Different Library Builds

In VisualDSP++ 3.5, the majority of libraries in the LDF for the ADSP-BF561 processor were the same libraries used by the single-core ADSP-BF532 processor. As a result of providing silicon revision-specific versions of libraries for each processor, this is no longer the case. Existing projects that use ADSP-BF561 processor and have a custom LDF should change the LDF to use libraries with a “561” suffix, rather than a “532” suffix.

Default CRTs Changed

The default C Run-Time Header (CRT) file is different from the CRT included in the VisualDSP++ 3.5 base release. All the changes listed below were introduced as part of updates to the base VisualDSP++ 3.5 release:

- In the base release, the processor speed was automatically raised to the fastest supported speed. Now, this is controlled by the `__clk_ctrl` global variable, and the default is to leave the processor speed unchanged. The `<sys/p11.h>` file defines this default state.
- The CRT now ensures that the `ITEST_COMMAND` and `DTEST_COMMAND` registers contain “Read” commands. If some event has to save these registers, manipulate the cache and restore the registers, this ensures that the restoration does not execute a `Write` command, writing garbage to a random address.

This problem could potentially occur with C++ exception handling, memory initialization on ADSP-BF535 and AD6532 processors, and if using the `_ll_memcpy` or `_memcpy_ll` routines.

- The unused entries in the Event Vector Table are initialized with a pointer to the “unknown exception occurred” handler, which raises an Emulation event. This makes it easier to see when unexpected events have occurred, which are usually a result of programming errors.
- The control variable `___cplb_ctrl` is not read until after `_mi_initialize()` has completed, in case the control variable's value needs to be initialized from read-only memory first.
- Library routine `_mc_data_initialize()` is called for all dual-core processors, to initialize the per-core data-storage records.
- When the processor's priority is lowered to the minimum supervisor level (IVG15), some higher-priority events may still be flagged as pending. This situation can occur if the application is restarted while servicing an interrupt. To avoid this occurrence, instead of waiting in user mode for the IVG15 event to take effect, the CRT keeps returning from events until no more higher-priority events are pending. At this point, the IVG15 event can occur, and the CRT will continue from the “`supervisor_mode`” label as before.
- The DAG port preferences are set in the opposite manner. This action has no effect on functionality or performance; the change is merely to bring the library's behavior in line with existing documentation from Analog Devices.
- The environment variable array is now defined separately in the library.

Compiler and Library for TigerSHARC Processors

This section summarizes changes to the TigerSHARC compiler and library.

Stack Must Be in Internal Memory

The stack is placed in internal memory by the LDFs, and must not be moved to external memory.

Changes to the I/O Library

Changes have been made internally to the `stdio` run-time library (`libio.dlb`) to make it both smaller and faster with respect to the formatting of floating-point values. As a consequence, you may notice a minor difference in your output, particularly for applications that are built with the `-double-size-32` switch and that print floating-point values with a precision of more than six digits.

C/C++ Run-Time Library Naming Conventions

The C/C++ run-time support libraries for TigerSHARC no longer have either the `_NP` or `_w` suffixes. These suffixes should be removed from any custom LDFs as the `-si-revision` switch has been extended to select libraries appropriate for specific silicon revisions of the hardware. The use of `-si-revision any` will select a set of libraries that are suitable for use on any silicon revision.

Compiler and Library for SHARC Processors

Changes have been made internally to the `stdio` run-time library (`libio.dlb`) to make it both smaller and faster with respect to the formatting of floating-point values. As a consequence, you may notice a minor difference in your output, particularly for applications that are built with the switch `-double-size-32` and that print floating-point values with a precision of more than six digits.

IDDE Changes

By default, the IDDE now uses terse output when building project and displays only the file name as the build progresses.

To revert to the historical behavior (full command-line output), go to the **Settings->Preferences->Project** dialog box and select **Verbose build output**.

When creating a new project, you are no longer prompted for VDK support. To create a project with VDK support, select **multi-threaded application using VDK** when initially creating the project.

Loader Changes

The following loader changes from 3.5 to 4.0 apply to Blackfin processors.

- Support for silicon revision 0.3 for the ADSP-BF531, ADSP-BF532, ADSP-BF533, and ADSP-BF561 processors by default from silicon revision 0.2.
- The use of fixed values to fill the address field of the executable byte count block header. The values are: 0xFF800000 for silicon revise 0.2 for ADSP-BF531/2/3 and 0xFF800040 for 8-bit output and 0xFF800060 for 16-bit output for silicon revise 0.3 for ADSP-BF531/2/3. Those values used to be all zeros in the 3.5 release.
- 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.

Linker Changes

The default .LDF files for Blackfin processors provided with VisualDSP++ 4.0 contain several changes to support new features of the tools.

- VisualDSP++ 3.5 projects that contain customized .LDF files can be safely used within VisualDSP++ 4.0 without the need to update their .LDF file.
- Projects with customized .LDF files from earlier versions of VisualDSP++ should be updated. The modifications applied to the previous .LDF file should be migrated into a copy of the default .LDF file provided with VisualDSP++ 4.0.

For more information, see [“Migration of LDFs From Previous Versions of VisualDSP++”](#) on page 3-44.

3 NEW FEATURES AND ENHANCEMENTS

VisualDSP++ 4.0 has a number of new features and enhancements designed to increase productivity and shorten application development cycles. This chapter describes the new features and enhancements introduced in VisualDSP++ 4.0.

The information is presented as follows.

- [“VisualDSP++ IDDE” on page 3-2](#)
- [“Assembler” on page 3-4](#)
- [“New Features Common to All Compilers” on page 3-6](#)
- [“Compiler and Library for Blackfin Processors” on page 3-14](#)
- [“Compiler and Library for TigerSHARC Processors” on page 3-24](#)
- [“Compiler and Library for SHARC Processors” on page 3-26](#)
- [“Linker and Utilities” on page 3-30](#)
- [“Loaders and Splitter” on page 3-49](#)
- [“VDK” on page 3-50](#)

VisualDSP++ IDDE

The VisualDSP++ 4.0 Integrated Development and Debugging Environment (IDDE) introduces:

- “New Processor Support”
- “New ICE Features”
- “New Installation Options”
- “New “Authorize Script” Option”

For more information about VisualDSP++ IDDE, refer to the *VisualDSP++ 4.0 User's Guide* and online Help.

New Processor Support

The following new processors are supported in VisualDSP++ 4.0.

- Blackfin processors: ADSP-BF536, ADSP-BF537, ADSP-BF538, ADSP-BF539, and ADSP-BF566
- SHARC processors: ADSP-21366, ADSP-21367, ADSP-21368, and ADSP-21369

For more information on these processors, refer to a hardware reference and data sheet of an appropriate processor.

New ICE Features

The ICE Configurator is now available from the IDDE's **Create Session** dialog box. It continues to be available as a stand-alone tool.

The ability of the ICE to set registers to a specific value on reset and before a program load is now supported for Blackfin and TigerSHARC processors but not for SHARC processors.

A traditional application of this feature is to initialize SDRAM values immediately before loading an application. This feature is on by default and, by default, initializes SDRAM to the values required for EZ-KIT Lite operation. (Note that this feature was added in an Update to VisualDSP++ 3.5.)

Watchpoint support has been added for the Blackfin architecture. (Note: this feature was added in an update to VisualDSP++ 3.5.)

New Installation Options

VisualDSP++ 4.0 may be installed in multiple discrete locations on the same machine. Likewise, an existing installation of VisualDSP++ 4.0 can be cloned, creating a copy of that installation in a different directory. Because the “**Maintain this installation**” option of the **Start** menu, available on a per-installation basis, different instances of VisualDSP++ 4.0 may have different updates installed. To effect this functionality, Updates downloaded from Analog Devices' Web site cannot be executed directly. A new **Start** menu item, **Maintain this installation**, is now used to install Updates to VisualDSP++.

The installer also has the ability to install “in situ” on a set of files, perhaps retrieved from a version control system. This is an advanced installation option, detailed in a separate EE-Note from Analog Devices, Inc.

New “Authorize Script” Option

If Norton Antivirus' “Script Blocking” feature is enabled, when you add startup code to your project via the **Project Wizard** or **New System Component** dialog boxes, Norton Antivirus displays a **Malicious script detected** dialog box. The IDDE runs a script to generate the startup code to let you safely ignore this warning. In the dialog box, choose **Authorize this script** and click **OK**, and you will not see the warning again.

Assembler

VisualDSP++ 4.0 continues to support the following assembler drivers:

- For **SHARC processors** – `easm21k.exe` assembler driver
- For **TigerSHARC processors** – `easmts.exe` assembler driver
- For **Blackfin processors** – `easmbkfn.exe` assembler driver

The *VisualDSP++ 4.0 Assembler and Preprocessor Manual* and online Help now combine all assembler and preprocessor feature descriptions under appropriate main topics.

Assembler Feature Macros

New assembler feature macros for VisualDSP++ 4.0 are:

Blackfin Processors

-D__ADSPBF536__=1 -D__ADSP21536__=1	Present when running <code>easmbkfn -proc ADSP-BF536</code> with ADSP-BF536 processor.
-D__ADSPBF537__=1 -D__ADSP21537__=1	Present when running <code>easmbkfn -proc ADSP-BF537</code> with ADSP-BF537 processor.
-D__ADSPBF538__=1 -D__ADSP21538__=1	Present when running <code>easmbkfn -proc ADSP-BF538</code> with ADSP-BF538 processor.
-D__ADSPBF539__=1 -D__ADSP21539__=1	Present when running <code>easmbkfn -proc ADSP-BF539</code> with ADSP-BF539 processor.
-D__ADSPBF566__=1	Present when running <code>easmbkfn -proc ADSP-BF566</code> with ADSP-BF566 processor.

SHARC Processors

-D__ADSP21366__=1 -D__2136x__=1	Present when running <code>easm21k -proc ADSP-21366</code> with ADSP-21366 processors
-D__ADSP21367__=1 -D__2136x__=1	Present when running <code>easm21k -proc ADSP-21367</code> with ADSP-21367 processors

-D__ADSP21368__=1 -D__2136x__=1	Present when running easm21k -proc ADSP-21368 with ADSP-21368 processors
-D__ADSP21369__=1 -D__2136x__=1	Present when running easm21k -proc ADSP-21369 with ADSP-21369 processors

.SEPARATE_MEM_SEGMENTS Directive



Used with TigerSHARC processors ONLY.

The `.SEPARATE_MEM_SEGMENTS` directive allows you to specify two buffers which the linker places into different memory segments. For example,

```
.SECTION data1;
.VAR buf1;
.VAR buf2;
.EXTERN buf3;
.SEPARATE_MEM_SEGMENTS(buf1, buf2)
.SEPARATE_MEM_SEGMENTS(buf1, buf3)
```

For more information, refer to the *VisualDSP++ 4.0 Linker and Utilities Manual*.

Special Assembler Operators

The assembler supports these new special assembler operators:

HI(expression) LO(expression)	<p>Extracts the most significant 16 bits of the expression. Extracts the least significant 16 bits of the expression.</p> <p>Note: These operators are used with the Blackfin assembler ONLY, where HI/LO replaces the ADDRESS() operator. The expression in the HI and LO operators can be either symbolic or constant.</p>
----------------------------------	---

New Features Common to All Compilers

New features common to all compilers are:

- [“New Switches” on page 3-6](#)
- [“New Pragmas” on page 3-7](#)
- [“New Built-In Functions to Control Branch Prediction” on page 3-11](#)
- [“New -Og Switch for Optimizing and Debugging” on page 3-12](#)
- [“Support for Gathering Cycle-Counts” on page 3-12](#)
- [“Bank Type Qualifiers” on page 3-13](#)
- [“Predefined Compiler Macros” on page 3-13](#)

This section summarizes new pragmas and other new features common to all compilers.

New Switches

[Table 3-1](#) describes the new switches that are common to all compilers.

Table 3-1. New C/C++ Switches Common to All Compilers

Switch Name	Description
-no-implicit-inclusion	C++ mode only. Prevents implicit inclusion of source file when processing corresponding header file.
-Og	Optimization mode with enhanced debugging information. See “New -Og Switch for Optimizing and Debugging” on page 3-12 .
-progress-rep-func	Issues a remark when starting to compile a new function.

Table 3-1. New C/C++ Switches Common to All Compilers (Cont'd)

Switch Name	Description
<code>-progress-rep-gen-opt</code>	Issues a remark at the start of each generic optimizer stage.
<code>-progress-rep-mc-opt</code>	Issues a remark at the start of each machine-level optimizer stage.
<code>-section <id></code>	Tells the compiler to use <code>sectname</code> as the section to store generated information in. See “New Section Used for C++ Virtual Function Tables” on page 2-7, and “Switch Table Placement Control” on page 3-28.
<code>-structs-do-not-overlap</code>	Tells the compiler that it is safe to use the simpler, faster semantics of <code>memcpy</code> rather than the more strict <code>memmove</code> .

New Pragmas

- `#pragma align` accepts keywords

In addition to specifying the alignment of a following variable or field by using a numeric value, `#pragma align` accepts a keyword to indicate the required alignment:

Keyword	Alignment Boundary Specified
<code>_WORD</code>	32 bit
<code>_LONG</code>	64 bit
<code>_QUAD</code>	128 bit

New Features Common to All Compilers

- New pragma `alignment_region`, for aligning multiple variables

`#pragma align` applies to the immediately following variable only. A single alignment can be specified for a group of variables with the new `alignment_region` pragma. This pragma specifies an alignment, as does the `align` pragma, but unlike the `align` pragma, `alignment_region` is persistent until the terminating `alignment_region_end` pragma. Refer to `alignment_region` in compiler manual for details.

- New pragmas `section` and `default_section`

The `section()` keyword (refer to `section()` in the compiler manual, formerly “Placement Support Keyword” in the 3.5 manual) is now also available as a pragma. The following are equivalent:

```
section("mysec") int foo;  
  
#pragma section("mysec")  
int foo;
```

The pragma form has an advantage over the keyword, as several pragmas can be specified in sequence.

The pragma form also accepts one or more qualifiers to specify the attributes of the section, for example:

```
#pragma section("mybsz", ZERO_INIT)  
int bar;
```

The `section` pragma applies to the immediately-following declaration. In contrast, the `default_section` pragma is persistent, and allows you to specify the sections used by the compiler when no `section` pragma or keyword is present.

The following example places both `imin()` and `imax()` into section `fastmem`, but does not affect `baz` (which goes into a data section, not a code section):

```
#pragma default_section(CODE, "fastmem")

int baz;
int imin(int a, int b) { return a<b? a : b; }
int imax(int a, int b) { return a>b? a : b; }
```

The `default_section` pragma also accepts qualifiers to indicate the attributes of the specified section. Refer to the `#pragma` section in compiler manual for more details.

- New pragmas for specifying memory attributes (Blackfin and TigerSHARC processors only)

The new memory bank pragmas allow you to specify attributes of memory banks, such as whether they are internal or external, and how many cycles a read or write is expected to require. The compiler can then use these attributes during optimization. For example, the compiler is more likely to inline a call to a function in slow memory from a function in fast memory than the other way around.

Memory banks can be assigned to data items, to functions (that is, the memory where the function's code resides) and to the stack. Attributes can then be assigned to memory banks. Refer to “Memory Bank Pragmas” in the compiler manual for more details.

New Features Common to All Compilers

- New `pragma diag`, to control compiler error and warning levels

The new `diag` `pragma` allows you to change the compiler's warning settings during compile-time. For example, you can turn off some warnings temporarily, or promote their severity to errors. This can be useful if you have a particular header file that triggers a lot of warnings that you know can be safely ignored, but where you do not want to turn off those warnings for the compilation of the whole module. In such cases you could do:

```
#pragma diag(push)

#pragma diag(suppress: <number of warning to suppress>)

#include <trusted_header_file.h>

#pragma diag(pop)
```

See the section about `#pragma diag` for more details.

- New `pragma no_implicit_inclusion`, to control C++ template instantiation

When the compiler includes a `.h` header file while compiling a C++ module, the compiler normally automatically includes the header file's corresponding `.c` or `.cpp` file too, if it exists. This feature is called *implicit inclusion*, and is used to assist in template instantiation.

Sometimes, implicit inclusion is not desirable. In such cases, it can be disabled by adding the following to the header file:

```
#pragma no_implicit_inclusion
```


- New `pragma noreturn`, to specify control flow

`#pragma noreturn` can be specified on a function's declaration or definition to inform the compiler that the function will never return. The compiler can use this to omit any code that follows a call to the function, as that code is unreachable. This pragma can be used to reduce the overall size of the application, as it allows the compiler to delete code and data that might otherwise appear to be in use.

New Built-In Functions to Control Branch Prediction

The compiler supports two new built-in functions:

- `int expected_true(int);`
- `int expected_false(int);`

These can be used in conditions to indicate to the compiler which way you expect the condition to be evaluated. As a result, the compiler can ensure that the most commonly executed path is the most efficient. For example:

```
if (expected_true(buffer_valid(buffer)))  
    if (expected_false(send_msg(buffer)==FAILED))  
        abort_msg();
```

Refer to the “Compiler Performance Built-in Functions” section in the compiler manual.

New -Og Switch for Optimizing and Debugging

The new -Og switch provides a mid-way point between optimization and debugging:

Switches	Code Produced
-O	Heavily optimized. No debug support.
-g	Full debug support. No optimization.
-O -g	Preference given to optimization.
-Og	Preference given to debugging.

Support for Gathering Cycle-Counts

VisualDSP++ now supports a uniform means of gathering cycle counts and benchmarking algorithms and applications. Two alternative header files, `cycle_counts.h` and `cycles.h`, provide macro definitions that can be used to determine the processor cycles used by a code-sequence.

The macros defined by the `cycle_counts.h` header file represent a basic and relatively unobtrusive method for counting processor cycles. The set of macros defined by the `cycles.h` header file on the other hand are more expensive (in terms of both memory and cycles) but they also have the capability of accumulating statistics that are suited to recording the performance of a section of code that is executed repeatedly.

Refer to “Measuring Cycle Counts” in the compiler manual for a full description of the two header files and for an alternative method of benchmarking using the `time.h` header file.

Bank Type Qualifiers

The `bank("string")` keyword can be used in data declarations to indicate that the data resides in a particular memory bank. For any given function, three banks are automatically defined. These default banks are:

- “`__data`” for global “static” or “extern” variables
- “`__stack`” for local variables of “auto” storage class
- “`__code`” for the function’s instructions

Each memory bank can have different performance characteristics. A different bank can be selected with pragmas. For more information, see [“New Pragmas” on page 3-7](#).

Predefined Compiler Macros

The new predefined compiler macro is `__VERSIONNUM__`. It defines `__VERSIONNUM__` as a numeric variant of `__VERSION__` constructed from the version number of the compiler. Eight bits are used for each component in the version number and the most significant byte of the value represents the most significant version component. As an example, a compiler with version 7.0.1.5 defines `__VERSIONNUM__` as `0x07000105`.

Compiler and Library for Blackfin Processors

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

- [“C/C++ Compiler Command-Line Switches” on page 3-15](#)
- [“Workaround <workaroundid>\[,<workaroundid> ...\]” on page 3-16](#)
- [“Endian-Swapping Intrinsics” on page 3-16](#)
- [“System Built-In Functions” on page 3-17](#)
- [“Misaligned Data Built-In Functions” on page 3-18](#)
- [“Long Double Floating Point Type” on page 3-19](#)
- [“Run-Time Libraries and Start-Up Files” on page 3-21](#)
- [“New C Library Functions” on page 3-21](#)
- [“New DSP Run-Time Library Functions” on page 3-22](#)
- [“CRT Generation” on page 3-23](#)
- [“Assembly Annotations” on page 3-23](#)
- [“New Clipping Fractional Shift Functions” on page 3-24](#)

For detailed information on these features, refer to the *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors* and online Help.

C/C++ Compiler Command-Line Switches

This section summarizes C/C++ compiler command-line switches introduced or enhanced in VisualDSP++ 4.0. [Table 3-2](#) lists and briefly describes each new or enhanced switch.

Table 3-2. C/C++ Compiler Command-Line Switches (Blackfin)

Switch Name	Description
<code>-cplbs</code>	Instructs the compiler to assume that CPLBs are active
<code>-decls-<weak strong></code>	Determines whether uninitialized global variables should be treated as definitions or declarations
<code>-double-size-any</code>	Indicate that the resulting object can link against any <code>double-size</code> object.
<code>-double-size-{32 64}</code>	Selects 32- or 64-bit IEEE format for <code>double</code> . The <code>-double-size-32</code> is the default mode.
<code>-extra-loop-loads</code>	Allows the compiler to read off the start or end of memory areas, within loops, to aid performance
<code>-full-io</code>	Links with the Dinkumware I/O library
<code>-guard-vol-loads</code>	Disables interrupts during volatile loads
<code>-jump-<constdata data code></code>	Determines which section the compiler uses to store jump-tables that are generated for C “switch” statements.
<code>-no-annotate</code>	Disables the annotation of assembly files
<code>-no-full-io</code>	Links with the Analog Devices I/O library. Enabled by default.

Table 3-2. C/C++ Compiler Command-Line Switches (Blackfin)

Switch Name	Description
-sdram	Instructs the compiler to assume that at least bank 0 of external SDRAM will be present and enabled
-workaround <workaround>	Enables code generator workaround for specific hardware errata; detailed in the following section.

Workaround <workaroundid>[,<workaroundid> ...]

The -workaround switch supports the following new workarounds.

WorkaroundID	Anomaly identifier
wb-dcache	05-00-0165
wt-dcache	05-00-0164
signbits	05-00-0127
killed-mmr-write	05-00-0157

Refer to the compiler manual for more details.

Endian-Swapping Intrinsics

The following two intrinsics are available for changing data from big-endian to little-endian, or vice versa.

```
#include <ccblkfn.h>
int byteswap4(int);
short byteswap2(int);
```

Since Blackfin processors use a little-endian architecture, these intrinsics are useful when communicating with big-endian devices, or when using a protocol that requires big-endian format. For example,

```
struct bige_buffer {
    int len;
    char data[MAXLEN];
} buf;
int i, len;
buf = get_next_buffer();
len = byteswap4(buf.len);
for (i = 0; i < len; i++)
    process_byte(buf.data[i]);
```

System Built-In Functions

The System Register Values built-in functions are enhanced to include 64-bit data types.

System Register Values

```
unsigned int sysreg_read(int reg);
void sysreg_write(int reg, unsigned int val);
unsigned long long sysreg_read64(int reg);
void sysreg_write64(int reg, unsigned long long val);
```

These functions get (read) or set (write) the value of a system register. In all cases, `reg` is a constant from the file `<sysreg.h>`.

Misaligned Data Built-In Functions

The following intrinsic functions exist to allow the user to explicitly perform loads from misaligned memory locations and stores to misaligned memory locations. These functions always generate expanded code to read and write from such memory locations, regardless of whether the access is aligned or not.

```
#include <ccblkfn.h>

short misaligned_load16(void *);
short misaligned_load16_vol(volatile void *);
void misaligned_store16(void *, short);
void misaligned_store16_vol(volatile void *, short);

int misaligned_load32(void *);
int misaligned_load32_vol(volatile void *);
void misaligned_store32(void *, int);
void misaligned_store32_vol(volatile void *, int);

long long misaligned_load64(void *);
long long misaligned_load64_vol(volatile void *);
void misaligned_store64(void *, long long);
void misaligned_store64_vol(volatile void *, long long);
```

Note that there are also volatile variants of these functions. Because of the operations required to read from and write to such misaligned memory locations, no assumptions should be made regarding the atomicity of these operations.

Long Double Floating Point Type

The Blackfin compiler now supports 64-bit double-precision floating-point data types, such as the `long double` data type. The `long double` data type, being a floating-point type, has several attributes in common with the 32-bit `float` data type:

- Being an emulated type, it is not as efficient as the integer and fractional types that are supported natively by the Blackfin architecture.
- There are two flavors of emulation library. By default, VisualDSP++ links with the “fast” version which provides IEEE-compliant behavior, apart from some relaxed checks for NaN values as inputs. There is also a fully-compliant library flavor, which supports the full IEEE behavior at the expense of some performance.
- The two library flavors are selectable with the `-fast-fp` and `-ieee-fp` switches.

The size of the `double` data type is now selectable, through the following switches:

Switch	Double Data Type
<code>-double-size-32</code>	32-bit single-precision
<code>-double-size-64</code>	64-bit double-precision
<code>-double-size-any</code>	(double type not used)

Compiler and Library for Blackfin Processors

By default, the `double` data type is a 32-bit single-precision type, as it was for VisualDSP++ 3.5. That is, `double` is the same size as the `float` data type. The `-double-size-32` switch can be used to select this default case explicitly.

The `-double-size-64` switch makes the `double` data type a 64-bit double-precision type. That is, `double` is the same size as the `long double` data type.

Because the `-double-size` switches change the size and representation of the `double` data type, it is not safe to mix modules that have been compiled with different `double` data types, and the linker will issue a linkage error if this is attempted. For library routines that make no use of the `double` data type, the `-double-size-any` switch may be used to indicate that the module can be linked with `double-size-32` or `double-size-64` modules without conflict.

Note that the `double` data type is the only type that is affected by the `-double-size` switches: the `float` data type is always a 32-bit single-precision data type, and the `long double` data type is always a 64-bit double-precision data type.

The run-time libraries have been upgraded to support the dual sizes of the `double` type and also the new size and properties of the `long double` data type. Thus, for every library function that supports the `float` data type, there is a library function that also supports the `double` and `long double` data types.

In the run-time libraries, floating-point functions whose name end with `f` are generally functions that operate on the `float` data type; functions whose name end with `d` operate on the `long double` data type, and names of floating-point functions that do not end with either `f` or `d` generally operate on the `double` data type. All these new library functions are documented in the compiler manual.

Run-Time Libraries and Start-Up Files

Several additional variants of the run-time libraries and binary files are supplied with VisualDSP++ that have been specifically built for the new Blackfin processors that are now supported. These new files are installed in the VisualDSP++ subdirectory `Blackfin\lib`, and are identified by one of the following suffixes in their filename:

Filename Suffix	Description
534	Compiled only for ADSP-BF534 processor
536	Compiled only for ADSP-BF536 processor
537	Compiled only for ADSP-BF537 processor
538	Compiled only for ADSP-BF538 processor
539	Compiled only for ADSP-BF539 processor
566	Compiled only for ADSP-BF566 processor

New C Library Functions

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

Library Function	Description
<code>heap_space_unused</code>	Returns the total free space in bytes for the heap with index <code>idx</code> .
<code>l1_memcpy</code> , <code>memcpy_l1</code>	Copies instructions from L1 Instruction Memory to data memory, and from data memory to L1 Instruction Memory.
<code>space_unused</code>	Returns the total free space in bytes for the heap with index 0.



The new functions are fully documented in the *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for Blackfin Processors*.

Compiler and Library for Blackfin Processors

For VisualDSP++ 4.0, the C run-time library now supports the following `long long int` library functions.

Library Function	Description
<code>atoll</code>	Converts a string to a long long int
<code>llabs</code>	Absolute value of a long long int
<code>lldiv</code>	Long long division returning quotient and remainder
<code>llmax</code>	Maximum of two long long ints
<code>llmin</code>	Minimum of two long long ints
<code>llclip</code>	Clipped value of two long long ints
<code>llcountones</code>	Count bits set in a long long int
<code>strtoll</code>	String to long long int conversion
<code>strtoull</code>	String to unsigned long long int conversion

New DSP Run-Time Library Functions

Two new functions have been added to the DSP run-time library; these new functions are identified in the following table.

Library Function	Description
<code>iirdfl_fr16</code>	Direct form I infinite response filter
<code>coeff_iirdfl_fr16</code>	Convert coefficients for DF1 IIR

The DSP run-time library contains the function `iir_fr16`, which represents a direct form II implementation of an infinite impulse response (IIR) filter. The function however makes the assumption that the A0 coefficient is greater than both A1 and A2 for all stages, and this assumption can be too restrictive for some applications.

The `iirdfl_fr16` library function is an alternative function that implements a direct form I IIR. It does not have the same restriction as the `iir_fr16` function but it does require that the filter coefficients are first transformed by the library function `coeff_iirdfl_fr16`.

The two new library functions are fully documented in Chapter 4 of the Blackfin compiler manual.

CRT Generation

VisualDSP++4.0 includes support for automatically configuring the C Run-Time header (CRT) through the **Project Creation Wizard**. This wizard allows you to have a customized CRT for your project that avoids the need for run-time checks to see whether certain support functions, such as cache configuration, are necessary.

Refer to the **Project Creation Wizard** in the VisualDSP++ 4.0 User's Guide for more details.

Assembly Annotations

VisualDSP++4.0 introduces assembly annotation to the Blackfin compiler. The compiler augments the generated assembly listing by including:

- A header, giving the amount of stack space the function requires, and listing the registers used by the function.
- Loop tracing, showing where each source-level loop appears in the assembly output, how much processor capacity the loop exercises (in terms of multi-issue slots), and how the loop has been optimized.

Compiler and Library for TigerSHARC Processors

The loop annotations are particularly useful because the compiler optimizer applies transformations such as vectorization, unrolling, and unroll-and-jam. Loops may be duplicated to provide vectorized and non-vectorized versions (selected at runtime), and loops may be entirely unrolled into linear code, or deleted as unreachable. All these transformations are recorded in the assembly annotations, allowing you to see how the optimizer has transformed your source code.

Refer to the section about assembly annotations in the Blackfin compiler manual for more details.

New Clipping Fractional Shift Functions

VisualDSP++ 4.0 introduces new fractional shift built-in functions that behave as expected when the shift magnitude is outside of a normal shift range. The ETSI shift functions call these by default. Refer to the section about `fract` built-ins in the Blackfin compiler manual for more details.

Compiler and Library for TigerSHARC Processors

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

- [“Compiler Command-Line Switch” on page 3-25](#)
- [“New Math Built-In Functions” on page 3-25](#)
- [“New TigerSHARC-Specific Pragma” on page 3-25](#)
- [“Miscellaneous Changes” on page 3-26](#)

For more information about these features, refer to the *VisualDSP++ 4.0 C/C++ Compiler Manual for TigerSHARC Processors* and online Help.

Compiler Command-Line Switch

VisualDSP++ 4.0 includes the following new TigerSHARC-specific switch:

```
-fp-div-lib
```

This switch specifies that a call to the run-time library support routine be made for floating-point divides rather than for planting in-line code.

New Math Built-In Functions

The new `RECIPS` and `RSQRTS` built-in functions are added to the set of built-in functions.

`RECIPS`

Instruction:

```
Rs = RECIPS Rm
```

Builtins:

```
float __builtin_recip (float);
```

Example:

```
float r, a;
r = __builtin_recip (a);
```

Description:

```
r corresponds to Rsd
a corresponds to Rmd
```

`RSQRTS`

Instruction:

```
Rs = RSQRTS Rm
```

Builtins:

```
float __builtin_rsqrt (float);
```

Example:

```
float r, a;
r = __builtin_rsqrt (a);
```

Description:

```
r corresponds to Rsd
a corresponds to Rmd
```

New TigerSHARC-Specific Pragma

The VisualDSP++ 4.0 C/C++ compiler supports a number of new pragmas. Pragmas are implementation-specific directives that modify the compiler's behavior. The new TigerSHARC-specific pragma is described briefly in [Table 3-3](#).

Table 3-3. New ADSP-TSxxx Compiler Pragma

Pragma	New or Enhanced Functions
<code>#pragma separate_mem_segments (var1, var2)</code>	This pragma specifies that the two variables <code>var1</code> and <code>var2</code> should be placed into different memory segments.

Miscellaneous Changes

The TigerSHARC stack can only be in internal memory, to allow interrupts to context-save safely.

Compiler and Library for SHARC Processors

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

- [“Access to System Registers” on page 3-26](#)
- [“Predefined Compiler Macros” on page 3-27](#)
- [“Switch Table Placement Control” on page 3-28](#)
- [“New Default Run-Time Headers” on page 3-28](#)

For more information about these features, refer to the *VisualDSP++ 4.0 C/C++ Compiler Manual for SHARC Processors* and online Help.

Access to System Registers

The `sysreg.h` header file defines a set of functions that provide efficient system access to registers, modes, and addresses not normally accessible from C source. These functions are specific to individual architectures.

For ADSP-2136x processors, in addition to the header files `def21363.h`, `def21364.h` and `def21365.h`, the `def21366.h`, `def21367.h`, `def21368.h` and `def21369.h` header files provide symbolic names for the individual bits in the system registers.

Predefined Compiler Macros

The enhanced and new macros are:

Macro	Description
<code>__2136x__</code>	cc21k defines <code>__2136x__</code> as 1 when compiling for the ADSP-21363, ADSP-21364, ADSP-21365 processors as well as new ADSP-21366, ADSP-21367, ADSP-21368, or ADSP-21369 processors.
<code>__ADSP21366__</code>	cc21k defines <code>__ADSP21366__</code> as 1 when you compile with the <code>-proc ADSP-21366</code> command-line switch.
<code>__ADSP21367__</code>	cc21k defines <code>__ADSP21367__</code> as 1 when you compile with the <code>-proc ADSP-21367</code> command-line switch.
<code>__ADSP21368__</code>	cc21k defines <code>__ADSP21368__</code> as 1 when you compile with the <code>-proc ADSP-21368</code> command-line switch.
<code>__ADSP21369__</code>	cc21k defines <code>__ADSP21369__</code> as 1 when you compile with the <code>-proc ADSP-21369</code> command-line switch.
<code>__SIMDSHARC__</code>	When compiling for ADSP-2116x, ADSP-2126x and ADSP-2136x processors, cc21k defines <code>__SIMDSHARC__</code> as 1. The <code>__SIMDSHARC__</code> define is used to identify processors that are capable of executing SIMD code.
<code>__VERSIONNUM__</code>	The preprocessor defines <code>__VERSIONNUM__</code> as a numeric variant of <code>__VERSION__</code> constructed from the version number of the compiler. Eight bits are used for each component in the version number and the most significant byte of the value represents the most significant version component. As an example, a compiler with version 7.0.1.5 would define <code>__VERSIONNUM__</code> to be the value <code>0x07000105</code> .

Switch Table Placement Control

The new compiler switch `-section switch=<secname>` directs the compiler to place switch tables (used to implement C/C++ 'switch' expressions) in the named section. By default, switch tables are placed in `seg_dmda`.

See also [“New Section Used for C++ Virtual Function Tables” on page 2-7](#) and [“New Switches” on page 3-6](#).

New Default Run-Time Headers

New default run-time headers from the appropriate `...lib` directory are:

ADSP-21366	213xx\lib\366_hdr.doj
ADSP-21367	213xx\lib\367_hdr.doj
ADSP-21368	213xx\lib\368_hdr.doj
ADSP-21369	213xx\lib\369_hdr.doj

A VisualDSP++ installation for ADSP-213xx processors includes additional default C and C++ start-up files for the new ADSP-21366, ADSP-21367, ADSP-21368, and ADSP-21369 processors. These new binaries are installed in the subdirectory `...213xx\lib` and are described in [Table 3-4](#).

Table 3-4. C and C++ Start-Up Files for ADSP-21366/7/8/9 Processors

Description	Library Name	Comments
C run-time library	libc36x.dlb libc36xmt.dlb	
C++ run-time library	libcpp.dlb libcppmt.dlb	
C++ run-time support library	libcppprt.dlb libcppprmt.dlb	

Table 3-4. C and C++ Start-Up Files for ADSP-21366/7/8/9 Processors

Description	Library Name	Comments
C++ run-time library with exception handling	libcppeh.dlb libcppehmt.dlb	
C++ run-time support library with exception handling	libcpprteh.dlb libcpprteht.dlb	
C++ exception handling support library	libeh.dlb libehmt.dlb	
DSP run-time library	libdsp36x.dlb	
I/O run-time library	libio.dlb libiomt.dlb	
I/O run-time library with no sn support for alternative device drivers or printf(“%a”)	libio_lite.dlb libio_litemt.dlb	
C start-up file – calls set-up routines and main()	366_hdr.doj 367_hdr.doj 368_hdr.doj 369_hdr.doj	ADSP-21366 processor only ADSP-21367 processor only ADSP-21368 processor only- ADSP-21369 processor only
C++ start-up file – calls set-up routines and main()	366_cpp_hdr.doj 367_cpp_hdr.doj 368_cpp_hdr.doj 369_cpp_hdr.doj 366_cpp_hdr_mt.doj 367_cpp_hdr_mt.doj 368_cpp_hdr_mt.doj 369_cpp_hdr_mt.doj	ADSP-21366 processor only ADSP-21367 processor only ADSP-21368 processor only ADSP-21369 processor only ADSP-21366 processor only ADSP-21367 processor only ADSP-21368 processor only ADSP-21369 processor only

Linker and Utilities

The VisualDSP++ 4.0 linker and utility programs are upgraded to operate more efficiently on Blackfin, 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 3-31
- “Support for Specifying Two Buffers in Different Memory Segments” on page 3-32
- “Linker Command-Line Switches” on page 3-32
- “Updated List of LDF Keywords” on page 3-32
- “External Execution Packing in SHARC Processors” on page 3-34
- “Expert Linker Features” on page 3-35
- “Memory Initializer Utility” on page 3-43
- “Migration of LDFs From Previous Versions of VisualDSP++” on page 3-44

For more information, refer to the *VisualDSP++ 4.0 Linker and Utilities Manual* and online Help.

Modified Link Page in Project Options Dialog Box

Within VisualDSP++, the **Link** page of the **Project Options** dialog box is modified to have more flexibility in specifying tool settings for project builds. The **Link** item in the options tree presents several different pages of link options (see [Figure 3-1](#)).

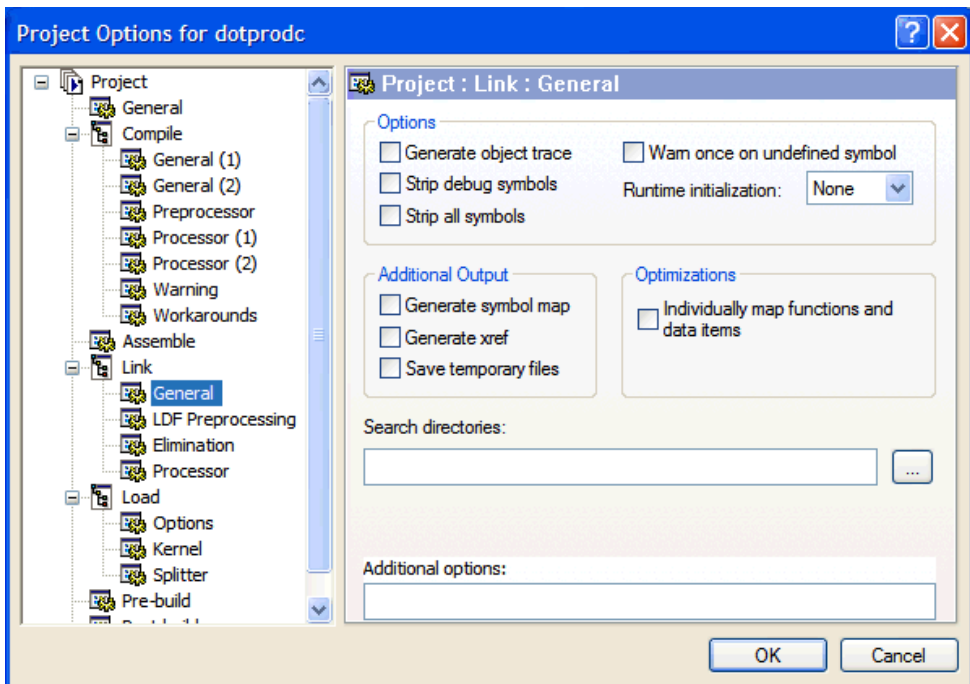


Figure 3-1. Main Link Tab with General Settings

You can access four sub-pages—**General**, **LDF Preprocessing**, **Elimination**, and **Processor**. Almost every setting option has a corresponding compiler command-line switch. For more information, refer to the *VisualDSP++ 4.0 Linker and Utilities Manual* and online Help.

Support for Specifying Two Buffers in Different Memory Segments

On TigerSHARC processors, the linker is enhanced to support efficient programming using the `.SEPARATE_MEM_SEGMENTS` assembler directive. This directive (or the compiler's `#pragma separate_mem_segments`) specifies two buffers directing the linker to place the buffers into different memory segments.

For example:

```
.SECTION data1;  
.VAR buf1;  
.VAR buf2;  
.EXTERN buf3;  
.SEPARATE_MEM_SEGMENTS(buf1, buf2)  
.SEPARATE_MEM_SEGMENTS(buf1, buf3)
```

The set of available memory segments for each buffer is defined by using the linker's "one-to-many" feature—mapping the input section(s) that contain the buffer into multiple memory segments.

For more information, refer to Chapter 2 of the *VisualDSP++ 4.0 Linker and Utilities Manual*. See also "Pragmas" in Chapter 1 of the *VisualDSP++ 4.0 C/C++ Compiler and Library Manual for TigerSHARC Processors* for more information.

Linker Command-Line Switches

Table 3-5 lists linker command-line switches that are unique for either Blackfin, TigerSHARC or SHARC processors.

Updated List of LDF Keywords

Table 3-6 lists `.LDF` file keywords that apply to all supported processors (in Blackfin, SHARC and TigerSHARC processor families).

Table 3-5. Linker Command-Line Switches

Switch	Description
-ek secName	Specifies a section name in which elimination should not take place
-ip	Fills fragmented memory with individual data objects that fit. Note: Not used with Blackfin processors.
-jcs21 and -jcs21+	Converts out-of-range short calls and jumps to the longer form. Note: Blackfin processors only.

Table 3-6. LDF File Keywords Summary

ABSOLUTE	ADDR	ALGORITHM
ALIGN	ALL_FIT	ARCHITECTURE
BEST_FIT	BM ¹	BOOT
DEFINED	DM1	ELIMINATE
ELIMINATE_SECTIONS	END	FALSE
FILL	FIRST_FIT	INCLUDE
INPUT_SECTION_ALIGN	INPUT_SECTIONS	KEEP
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	PM1
PROCESSOR	RAM	RESOLVE
RESOLVE_LOCALLY	ROM	SEARCH_DIR
SECTIONS	SHARED_MEMORY	SHT_NOBITS
SIZE	SIZEOF	SROM1

Table 3-6. LDF File Keywords Summary (Cont'd)

START	TYPE	VERBOSE
WIDTH	XREF	

1 Supported on ADSP-21xxx processors only.

External Execution Packing in SHARC Processors

The `.PACKING()` LDF command supports better data packing and memory management in VisualDSP++ 4.0. The following is the improved description of external execution packing functionality.

The only two processors that require packed memory for external execution are the ADSP-21161N and the ADSP-21065L chips. The ADSP-21161N processor supports 48-, 32-, 16-, and 8-bit-wide external memory. The ADSP-21065L processor supports 32-bit external memory only.

Previous to VisualDSP++ 3.5, it was required to use “packing” commands in the `.LDF` file to cause the code to be placed properly. In VisualDSP++ 3.5 and latter releases, the VisualDSP++ tools are enhanced to perform packing automatically.

In order for the VisualDSP++ to execute packing directly from external memory on ADSP-21065L and ADSP-21161N processors, VisualDSP++ tools “pack” the code into the external memory providing the following conditions are met:

- Ensure the “type” of the external memory is PM (Program Memory)
- Ensure the data width matches the “real/actual” memory width: ADSP-21065L processor – 32 bits; ADSP-21161N processor – 48, 32, 16 and 8 bits
- If the .LDF file has the `PACKING()` command for the particular section, remove the command

When defining memory segments (required for external memory), the “type” of a memory section is recommended to be:

- PM – code or 40-bit data (data requires PX register to access)
- DM – all other sections

Width should be the “actual/physical” width of the external memory.

Expert Linker Features

The Expert Linker provides a GUI-based means of supplying the link commands currently supplied to the linker in a Linker Description File (.LDF). For more information, refer to the corresponding chapter of the *VisualDSP++ 4.0 Linker and Utilities Manual* and online Help.

Adding a Memory Segment

Using Expert Linker, you can add memory segments to the memory map. This procedure assumes that the **Expert Linker** window (**Memory Map** pane) is open. To add a memory segment:

1. Right-click in the **Memory Map** pane.
2. Choose **New** and then choose **Memory Segment**. The **Memory Segment Properties** dialog box appears. In **Name**, type a name for the memory segment.
3. Specify the following attributes:
 - Start address
 - End Address
 - Size (hexadecimal)
 - Width
 - ROM/RAM
 - Internal/External (memory location)
 - Memory Space – Not available for Blackfin and TigerSHARC processors because these processors employ a unified memory space.
4. Click **OK**.

Managing Output Section Properties

Use the **Output Section** tab ([Figure 3-2](#)) to change the output section's name or to set the overflow. The enhanced dialog box introduces new options—**Initialization** (for selecting the initialization qualifier for an output section) and **Contiguity of Input Sections**.

To specify output section properties:

1. Right-click an output section (for example, PROGRAM_DXE or CODE_DXE) in the **Memory Map** pane.
2. Choose **Properties** to open the **Output Section Properties** dialog box (Figure 3-2).

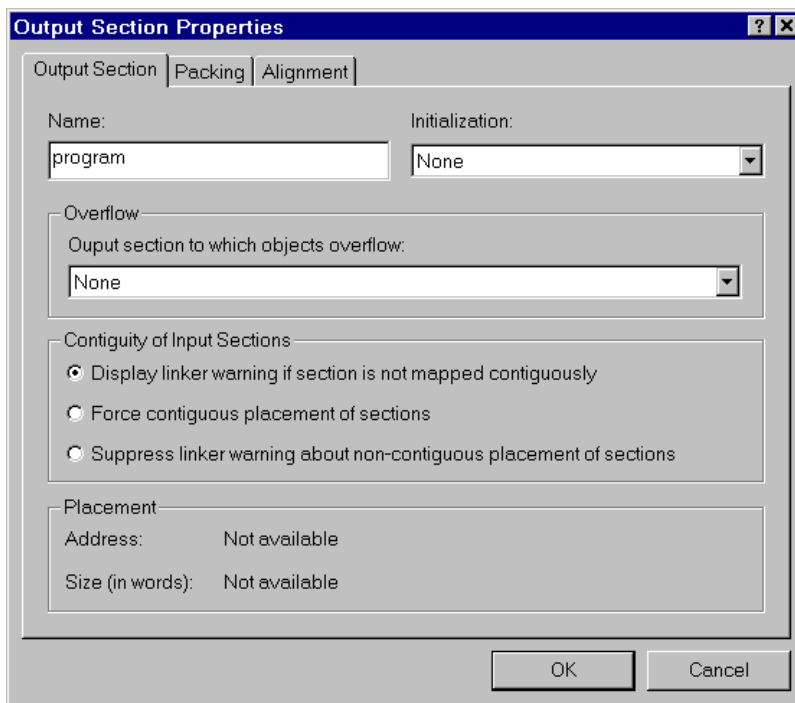


Figure 3-2. Output Section Properties Dialog Box – Output Section Tab

To pin objects to an output section, right-click the object in the **Memory Map** pane and choose **Pin to Output Section**.

3. Complete the dialog box as follows.

Name – Type a name for the output section.

Overflow – Select an output section into which the selected output section will overflow. All output sections are listed. Select **None** for no overflow. This setting appears in the **Placement** box.

Before linking the project, the **Placement** box indicates the output section's address and size as "Not available". After linking is done, the box displays the output section's actual address and size.

Initialization – Choose the initialization qualifier for an output section. The choices are:

- **None:** No initialization qualifier is specified for this output section.
- **No initialization:** No data initialization will happen for this output section.
- **Initialize to zero:** The memory space for this section will be initialized to zero at either "load" or "runtime", if invoked with the linker's `-meminit` switch. If the `-meminit` switch is not used, the memory is initialized at "load" time when the `.DxE` file is loaded via VisualDSP++ IDDE, or boot-loaded by the boot kernel. If the memory initializer is invoked, the C/C++ run-time library (CRTL) will process embedded information to initialize the memory space during the CRTL initialization process.
- **Initialize at runtime:** If the linker is invoked with the `-meminit` switch, this section will be filled at runtime. If the `-meminit` switch is not specified, the section is filled at "load" time.

Contiguity of Input Sections – Choose whether code or data in an output section should be mapped contiguously. The choices are:

- **Display linker warning if section is not mapped contiguously**
 - **Force contiguous placement of sections**
 - **Suppress linker warning about non-contiguous placement**
4. Specify the **Packing and Alignment** (with **Fill value**) properties as needed.

Managing Packing Properties

Use the **Packing** tab to specify the packing format that the linker employs to place bytes into memory. The enhanced packing properties' selection procedure is:

1. Right-click a memory segment in the **Memory Map** pane.
2. Choose **Properties** and click the **Packing** tab.
3. In **Packing method**, select a method.

No packing	Specifies no packing. Number of bytes and Packing order are grayed out.
Custom	Permits the selection of number of bytes and packing order.
other choices-	Specifies the number of bytes and packing order of the selected method. The list of packing methods is derived from the included packing .h file. Packing method information (number of bytes and packing order) appears, but you cannot change it.

4. In **Number of bytes** (if **Custom** is selected), specify the number of bytes to be reordered at one time. This value does not include the number of null bytes inserted into memory.

5. In **Packing order**, specify byte packing by selecting a byte and performing one of these actions:
 - Click the keyboard's Up arrow or Down arrow key.
 - Drag and drop it to a new location.
 - Insert a null byte by clicking on **Insert**.
 - Delete a null byte by selecting the null byte and clicking **Delete**.
6. Click **OK**.

Managing Overlay Properties

Use the **Overlay** tab to add/choose the output file for the overlay, its “live” memory, and its linking algorithm. The enhanced description of the overlay properties’ selection is:

1. Right-click an overlay object in the **Memory Map** pane.
2. Choose **Properties** and click the **Overlay** tab.
3. Use the **Output file name** box to specify the name of the overlay file (.OVL).
4. The **Live Memory** drop-down list contains all output sections or memory segments within one output section. The “live” memory is where the overlay is stored before it is swapped into memory.
5. The **Overlay linking algorithm** box permits one overlay algorithm—`ALL_FIT`. Expert Linker does not currently allow changes to this setting. When `ALL_FIT` is used, the linker tries to fit all of the mapped objects into one overlay.

6. The **Placement** box provides the following information:
 - **Live Address** – The starting address of the overlay
 - **Run Address** – The starting address where the overlay is swapped into memory at runtime
 - **Size** – The overlay's size
7. Click the **Packing** tab to specify byte packing order.

The **Browse** button is only available after the overlay build and when the symbols are available. Clicking **Browse** opens the **Browse Symbols** dialog box. You can choose the address for the symbol group or let the linker choose the address.

Managing Shared Memory Properties

You can specify the path and name of the file used by shared memory. This procedure assumes the **Expert Linker** window is open.

To specify shared memory properties:

1. In the **Memory Map** pane, click the **Shared Memory** tab (located at the bottom of dialog box).
2. Right-click anywhere on the **Memory Map** pane.
Note: Do not right-click on a memory segment, output section, input section, or overlay.
3. Choose **Properties**. The **Shared Memory** page of the **Shared Memory Properties** dialog box appears (see [Figure 3-3](#)).
4. In **Output file name**, specify the name of the output file for the shared memory.
5. In **Processors sharing this memory**, select the processors that share the file whose name appears in **Output file name**. Selecting a processor links its executable file against this shared memory file.

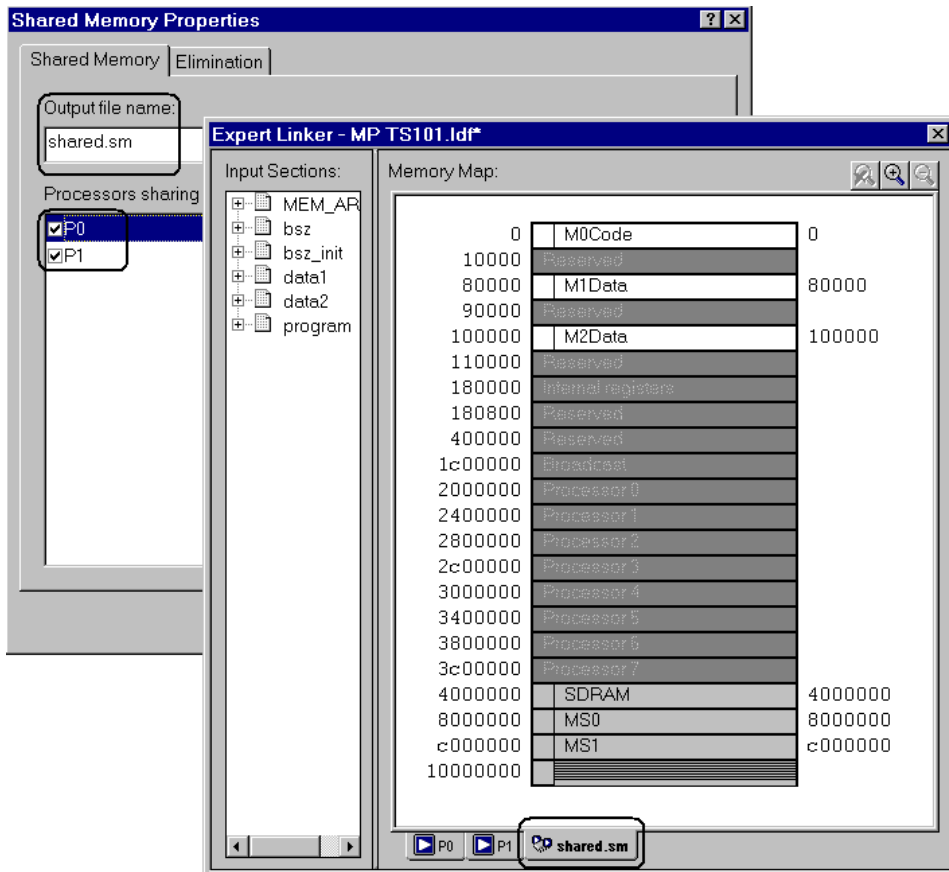


Figure 3-3. Shared Memory Tab Sample

6. Optionally, click the **Elimination** tab and specify options.
7. Click **OK**.

Memory Initializer Utility

The Memory Initializer (`MemInit.exe`) has been added as a separate utility. The Memory Initializer's main function is to generate a single initialization stream and save it in a section in the output executable file (`.DxE`). You can invoke this utility from the VisualDSP++ IDDE as well as from its own, linker's or compiler's command line.

The Memory Initializer may be used with processor systems where the RAM memory needs to be initialized with the code and data resided in the ROM memory before the execution of the application code begins. This is generally true for a processor system running in NO-BOOT mode.

The initialization stream generated by the Memory Initializer is consumed by a dedicated Run-Time Library (RTL) routine. Following a system reset, the RTL routine searches the initialization stream and initializes the processor's RAM memory with the data in the initialization stream before the call to `main()`, the starting point of the application code.

In creating initialization stream, the Memory Initializer can, in most cases, effectively reduce the overall size of an executable file by combining contiguous, identical initialization into a single block. For example, a large zero-initialized array in an executable file can be compressed to a single small data block by the Memory Initializer.

Refer to Chapter 7, "Memory Initializer" of the *VisualDSP++ 4.0 Linker and Utilities Manual* and online Help for more information.

Migration of LDFs From Previous Versions of VisualDSP++

The default .LDF files for Blackfin provided with VisualDSP++ 4.0 contain several changes to support new features of the tools, and enhancements to provide a better default template. VisualDSP++ 3.5 projects containing customized .LDF files can be safely used within VisualDSP++ 4.0 without the need to update their .LDF file. However, some new features provided with VisualDSP++ 4.0 will not be supported as they require changes to existing .LDF files. These features are listed below.

Projects containing customized .LDF files from earlier versions of VisualDSP++ should be updated. The modifications applied to the previous .LDF file should be migrated into a copy of the default .LDF file provided with VisualDSP++ 4.0.

Data Section for C++ Virtual Tables

The virtual tables are now placed in a separate data section, `seg_vtbl` for SHARC processors and `vtbl` for Blackfin and TigerSHARC processors. The default LDFs supplied with VisualDSP++ 4.0 handle this task, but customized LDFs must be amended (see the default LDFs for details). Refer to [“New Section Used for C++ Virtual Function Tables” on page 2-7](#) for more information.

The [“Assembler Feature Macros” on page 3-4](#) discusses new assembler macros.

Blackfin-Specific Features

The following new features apply to Blackfin processors.

New Features Requiring .LDF Support

64-Bit Floating-Point Support – VisualDSP++ 4.0 provides software-based support for 64-bit `long double` and `double` types. The support routines for these types are provided in the new `libf64ieee*.dlb` libraries. These libraries must be included in the list of libraries linked into the application in order to use the new data types.

Fast I/O Support – VisualDSP++ 4.0 provides a reduced functionality but faster implementation of the C I/O run-time library. This implementation is not complete and does not have wide character support. All features implemented within the reduced library are standard-compliant.

When projects are loaded from earlier versions of VisualDSP++, the project options will be configured to use this new fast I/O library while the LDF will not contain the correct configuration for enabling this feature. Projects with imported .LDF files should enable the **Full I/O** option from the **Project Options>Compile>Processor (1)** page. Support for the new I/O mechanism can be imported into existing projects by ensuring that `libio` precedes `libc` in the definition of the `LIBS` macro within the project LDF.

Project Configuration Wizard, CRT Generation – The new **Project Configuration Wizard** provides support for the generation of customized startup code within a project. This option requires a .LDF file based on the default template LDF provided with VisualDSP++ 4.0. The feature is not supported for projects from earlier revisions of VisualDSP++ which contain a customized .LDF file.

Non-Cached Code Section – When using the memory initialization tool for BF535, or C++ Exceptions for any platform, or directly calling `__memcpy` or `memcpy__`, the section `noncached_code` is required in the LDF (see [“New noncache_code Section Needed” on page 2-11](#)).

New Features in the VisualDSP++ 4.0 Default Template LDFs

Improved Placement of Data – The default LDFs provided with VisualDSP++ 4.0 contain several improvements to the ordering and placement of data. Where valid, data is placed into external memory once the internal sections have been filled. This feature is not supported for dual core processors, in order to ensure that shared and private data remain separate.

`USE_SDRAM` macro – The default LDFs contain a memory mapping, guarded by the `USE_SDRAM` link-time macro, which makes all the theoretical SDRAM space available to the application.

This macro differs from the `USE_CACHE` macro by not allocating space in internal memory for caching activities. Note that this configuration makes available all the potential SDRAM memory range. Ensure that your hardware has suitable SDRAM installed.

`PARTITION_EZKIT_SDRAM` macro – The default LDFs contain a memory mapping, guarded by the `PARTITION_EZKIT_SDRAM` link-time macro, which sections the SDRAM space to provide an optimal configuration of the memory space:

The External Bus Interface Unit's (EBUI) SDRAM Controller (SDC) provides access to the four external banks of SDRAM that can be supported. As well as supporting up to four external banks, the SDC also provides support for accessing four internal banks within each of the external SDRAM banks. The SDC allows for multiple internal banks to remain open in parallel, which can offer improved performance—for example, executing instructions from one external bank to access data in another internal bank.

The ADSP-BF533 processor allows for four internal banks within a single external bank to remain open at the same time. The ADSP-BF561 processor allows for four internal banks across the four external banks to remain open at the same time. The default Linker Description Files (LDF) for the ADSP-BF533 and ADSP-BF561 processors now contain an optional enhanced setup that provides SDRAM partitioning in the manner above.

For the ADSP-BF533 processor, you enable the configuration by defining the link-time macros `USE_CACHE` and `PARTITION_EZKIT_SDRAM`. The configuration makes use of the first external SDRAM bank, which is the SDRAM configuration for the ADSP-BF533 EZ-KIT Lite.

The memory is partitioned as follows:

E0I0:	Heap (8MB)
E0I1:	Data (8MB)
E0I2:	Data/Bsz (8MB)
E0I3:	Program (8MB)

For ADSP-BF561 EZ-KIT Lite the configuration is enabled by defining the link-time macro `USE_CACHE` to the linker. The configuration makes use of the first and second external SDRAM banks, both of which are populated by 32MB SDRAM for the ADSP-BF561 EZ -Kit Lite. The memory is partitioned as follows:

E0I0:	Core A Heap
E0I1:	Core A Data
E0I2:	Core A Data/Bsz
E0I3:	Core A Program
E1I0:	Core B Heap
E1I1:	Core B Data
E1I2:	Core B Data/Bsz
E1I3:	Core B Program + Shared area for both cores



For the ADSP-BF561 processor, enabling the `USE_CACHE` link-time macro moves the heap from L2 memory to L3/SDRAM. The L2 space is then made available for additional program/data.

TigerSHARC C/C++ Run-Time Library Naming Conventions

The C/C++ run-time support libraries for TigerSHARC no longer have either the `_NP` or `_w` suffixes. These suffixes should be removed from any custom LDFs as the `-si-revision` switch has been extended to select libraries appropriate for specific silicon revisions of the hardware. Using `-si-revision` any selects a set of libraries that are suitable for use on any silicon revision.

Loaders and Splitter

The loader program (`elfloader.exe`) for Blackfin, TigerSHARC and SHARC processors has been enhanced and modified to support new features and new processors. For details, see the *VisualDSP++ 4.0 Loader Manual*.

Blackfin Loader Features

New Blackfin loader features are:

- Support for the ADSP-BF534, ADSP-BF536, ADSP-BF537, ADSP-BF538, ADSP-BF539, and ADSP-BF566 (silicon revision 0.0) processors.
- Support for boot modes of UART and TWI (Two Wire Interface) for the ADSP-BF534, ADSP-BF536, ADSP-BF537 processors.
- A new switch `-pFlag` from which the loader accepts a 4-bit strobe value from 0 to 15 for the ADSP-BF531, ADSP-BF532, ADSP-BF533, ADSP-BF534, BF536, ADSP-BF537, and ADSP-BF538 processors. The default value is 0.
- Support for the special last block for the ADSP-BF537 processor for the TWI boot made. The loader uses the data from `0xFF903F00` to `0xFF903FFF` to make a last block. That space is reserved for the boot kernel to use as a data buffer during the boot processing.

SHARC Loader Features

The SHARC loader has been updated to support these processors:

- **ADSP-2126x processors:** ADSP-21261, ADSP-21266, ADSP-21267
- **ADSP-2136x processors:** ADSP-21363, ADSP-21364, ADSP-21365, ADSP-21366, ADSP-21367, ADSP-21368, and ADSP-21369

VDK

The following new features and enhancements have been added to VDK for the VisualDSP++ 4.0 release:

- Support for C++ exception handling.
- Support for writing ISRs in C/C++. Previous releases restricted the writing of VDK ISRs to assembly code.
- The use of common (with the run-time libraries) user modifiable startup code. Previous releases had non-modifiable startup code included in the VDK libraries. Due to this change all existing VDK LDF files must be updated for use in VisualDSP++ 4.0, and also the definitions of multiple heaps must be updated (see the VDK MultipleHeaps examples in the installation).

4 OBSOLETE OR REMOVED FEATURES

This chapter describes the features that have been deprecated or removed in VisualDSP++ 4.0. 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++ 3.5. Similarly, new projects created with VisualDSP++ 4.0 cannot be used by earlier versions of the tools.

This chapter contains listings of obsolete or removed features:

- [“Discontinued Processor Support” on page 4-2](#)
- [“VisualDSP++ IDDE” on page 4-2](#)
- [“Assembler and Preprocessor” on page 4-2](#)
- [“Compiler and Library for SHARC Processors” on page 4-2](#)
- [“Compiler and Library for TigerSHARC Processors” on page 4-3](#)
- [“Linker” on page 4-4](#)
- [“VCSE” on page 4-4](#)
- [“VDK” on page 4-4](#)

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

Discontinued Processor Support

VisualDSP++ 4.0 does not provide support for ADSP-21xx (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 21xx developers.

VisualDSP++ IDDE

Support for the Summit-ICE, ADI-ICE PAC, and Mountain-ICE have been removed in VisualDSP++ 4.0.

Assembler and Preprocessor

In the VisualDSP++ 4.0 preprocessor for TigerSHARC and SHARC processors, the following command-line switch was deprecated:

Switch	Description
<code>-cpredef</code>	Deprecated; replaced with the <code>-cstring</code> switch

Compiler and Library for SHARC Processors

[Table 4-1](#) lists the C/C++ SHARC compiler command-line switches that have been deprecated or removed.

Table 4-1. Obsolete/Deprecated C/C++ Compiler Common Switches

Switch Name	Description
-21xxx	All -21xxx switches are deprecated, use the proc -ADSP-21xxx switch instead
-vcse-add <filename>	Removed
-vcse-cname <component name>	Removed
-vcse-enforce	Removed
-vcse-names <filename>	Removed
-no-dollar	Removed
-no-inline	Removed
-no-restrict	Removed; use -no-extra-keywords instead
-swf-screening	Removed
-restrict	Removed; now the default mode
-traditional	Removed
-xml	Removed

Refer to Chapter 3, “[New Features and Enhancements](#),” for more information about the changes to the C/C++ compiler and run-time libraries.

Compiler and Library for TigerSHARC Processors

[Table 4-2](#) lists the C/C++ TigerSHARC compiler switches that were either renamed or became obsolete in VisualDSP++ 4.0.

Linker

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

Switch Name	Description
-TS101 201	Removed; use <code>-proc</code> (processor) switch instead
-dollar	Removed
-inline	Removed
-vcse-add <filename>	Removed
-vcse-cname <component name>	Removed
-vcse-enforce	Removed
-vcse-names <filename>	Removed

Linker

In VisualDSP++ 4.0, this command-line switch has been removed:

Switch	Description
-0vcse	Enabled VCSE method call optimization

VCSE

In VisualDSP++ 4.0, VisualDSP++ Component Software (VCSE) is not supported. All VCSE-related features in the assembler, compiler, and linker have been removed.

VDK

In VisualDSP++ 4.0, the API `VDK::GetThreadType` has been removed.