VisualDSP++® 5.0 Product Release Bulletin

Revision 3.0, August 2007

Part Number
82-000420-06

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

ANALOG
DEVICES

CONTENTS

VISUALDSP++ 5.0 OBSOLETE OR REMOVED FEATURES

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++®, release 5.0, which supports the following Analog Devices, Inc. processor families:

- SHARC® (ADSP-21xxx) processors
- TigerSHARC® (ADSP-TSxxx) processors
- Blackfin® (ADSP-BFxxx) processors

It also describes the differences (obsolete features and functions) between VisualDSP++ 5.0 and previous VisualDSP++ releases.

For details, refer to the VisualDSP++ 5.0 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 in VisualDSP++ 5.0.

Contents

This Product Release Bulletin consists of:

- Chapter 1, "Introduction"
  Describes VisualDSP++ 5.0 and its benefits, provides the minimal system requirements for running the product, and lists supported processors.

- Chapter 2, "VisualDSP++ 5.0 New Features and Enhancements"
  Describes what is new in the VisualDSP++ 5.0 IDDE, assembler, compiler, linker, loader, and documentation. Also describes the new features in projects and the VisualDSP++ Kernel (VDK).

- Chapter 3, "VisualDSP++ 5.0 Major Changes"
  Describes major changes in VisualDSP++ 5.0 projects compared to VisualDSP++ 4.5 projects.

- Chapter 4, "VisualDSP++ 5.0 Obsolete or Removed Features"
  Describes removed/obsolete features in VisualDSP++ 5.0 (compared to the previous VisualDSP++ software release) as they pertain to the code generation tool chain: commands, switches, operators, directives, pragmas, keywords, macros, and library functions.

Technical or Customer Support

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

- Visit the Embedded Processing and DSP products Web site at
  `http://www.analog.com/processors/technicalSupport`
- E-mail tools questions to
  `processor.tools.support@analog.com`
- E-mail processor questions to
  `processor.support@analog.com (World wide support)`
  `processor.europe@analog.com (Europe support)`
  `processor.china@analog.com (China support)`
- Phone questions to **1-800-ANALOGD**
- Contact your Analog Devices, Inc. local sales office or authorized distributor
- Send questions by mail to:

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

Supported Processors

VisualDSP++ 5.0 is for Blackfin (ADSP-BFxxx), SHARC (ADSP-21xxx), and TigerSHARC (ADSP-TSxxx) processors. For the complete list of supported processors, see "Platform and Processor Support" on page 1-4.

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, including documentation errata against all manuals. You can also choose to receive weekly E-mail notifications containing updates to the Web pages that meet your interests. `MyAnalog.com` provides access to books, application notes, data sheets, code examples, and more.

Registration

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

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

Processor Product Information

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

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

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

Related Documents

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

- *VisualDSP++ 5.0 Getting Started Guide*
- *VisualDSP++ 5.0 User's Guide*
- *VisualDSP++ 5.0 Assembler and Preprocessor Manual*
- *VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processors*
- *VisualDSP++ 5.0 C/C++ Compiler and Library Manual for TigerSHARC Processors*
- *VisualDSP++ 5.0 C/C++ Compiler Manual for SHARC Processors*
- *VisualDSP++ 5.0 Run-Time Library Manual for SHARC Processors*

- *VisualDSP++ 5.0 Linker and Utilities Manual*
- *VisualDSP++ 5.0 Loader and Utilities Manual*
- *VisualDSP++ 5.0 Kernel (VDK) User's Guide*
- *VisualDSP++ 5.0 Device Drivers and System Services Manual for Blackfin Processors*
- *VisualDSP++ 5.0 Licensing Guide*
- *VisualDSP++ 5.0 Installation Quick 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/technicalSupport/technicalLibrary/`

Online Technical Documentation

Online documentation includes the VisualDSP++ Help system, software tools manuals, hardware tools manuals, processor manuals, Dinkum Abridged C++ library, and FLEXnet 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 FLEXnet network license manager software documentation. Viewing and printing the `.html` files requires a browser, such as Internet Explorer 6.0 (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 (toolbar buttons, menu commands, and windows).

Accessing Documentation From Windows

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

Help system files (`.chm`) are located in the `Help` folder of VisualDSP++ environment. The `.pdf` files are located in the `Docs` folder of your VisualDSP++ installation CD-ROM. The `Docs` folder also contains the Dinkum Abridged C++ library and the FLEXnet 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++ 5.0**, and **VisualDSP++ Documentation**.

Accessing Documentation From the Web

Download manuals at the following Web site:

`http://www.analog.com/processors/technicalSupport/technicalLibrary/`

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.

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 `this` or `that`. 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 `this` or `that`.
`[this,…]`	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of `this`.
`.SECTION`	Commands, directives, keywords, and feature names are in text with `letter gothic` font.
`filename`	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 the 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, 5.0. 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++ 5.0 System Requirements" on page 1-3
- "Platform and Processor Support" on page 1-4

Product Release Description

VisualDSP++ is the Analog Devices project management and development environment for digital signal processing (DSP) applications. VisualDSP++ 5.0 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++ 5.0 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 the VisualDSP++ Kernel (VDK).

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

The successor to VisualDSP++ 4.0 and VisualDSP++ 4.5, VisualDSP++ 5.0 incorporates a number of new features and enhancements, as described in Chapter 2, "VisualDSP++ 5.0 New Features and Enhancements".

VisualDSP++ 5.0 System Requirements

To install and run VisualDSP++ 5.0, your computer must provide the following software, configuration, and system resources.

- Intel Pentium 32-bit processor (or x86 compatible), 1 GHz or faster
- Windows® 2000 SP4, Windows XP® Professional SP2 (or greater), Windows Vista™ Business edition, Windows Vista Enterprise edition, or Windows Vista Ultimate edition

ⓘ Windows NT®, Windows 98, and Windows ME are not supported.

ⓘ 64-bit host processors (x64) and 64-bit Windows editions are not supported.

- At least 2 GB of available hard drive space (approximately 500 MB of additional space is required temporarily when installing from the Web)
- At least 512 MB of RAM
- CD-ROM drive
- Internet Explorer 6.0 or later
- One open USB port for EZ-KIT Lite and USB-ICE connections
- One open PCI slot for HP-PCI ICE connections

Windows Vista Users

While every effort has been made to maintain a consistent user experience under all supported versions of Microsoft Windows, certain changes to the security model in Windows Vista may impact select operations in the

VisualDSP++ tools. *Windows Vista Compatibility in VisualDSP++ 5.0 Development Tools (EE-330)* describes these changes and suggests workarounds where applicable.

This document is available from VisualDSP++ Help and from the Analog Devices Web site at:
`http://www.analog.com/ee-notes`

Installation Warning Messages

When installing VisualDSP++ 5.0 onto a Windows Vista machine, warning message such as the following may be encountered.

While trying to open the downloaded installer:

```
The publisher could not be verified.
Are you sure you want to run this software?
```

-- or --

While the various drivers are being installed toward the end of the installation process:

```
Windows can't verify the publisher of this driver software
```

ⓘ These warnings are to be expected and can be ignored safely.

Platform and Processor Support

The following list of Analog Devices, Inc. processors is supported in VisualDSP++ 5.0.

SHARC (ADSP-21xxx) Processors

The name *SHARC* refers to a family of high-performance, 32-bit, floating-point processors that can be used in speech, sound, graphics, and imaging applications. VisualDSP++ currently supports the following SHARC processors.

ADSP-21020	ADSP-21060	ADSP-21061	ADSP-21062
ADSP-21065L	ADSP-21160	ADSP-21161	ADSP-21261
ADSP-21262	ADSP-21266	ADSP-21267	ADSP-21362
ADSP-21363	ADSP-21364	ADSP-21365	ADSP-21366
ADSP-21367	ADSP-21368	ADSP-21369	ADSP-21371
ADSP-21375			

Blackfin Processors

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

ADSP-BF522	ADSP-BF525
ADSP-BF527	ADSP-BF531
ADSP-BF532	ADSP-BF533
ADSP-BF534	ADSP-BF535
ADSP-BF536	ADSP-BF537
ADSP-BF538	ADSP-BF539
ADSP-BF542	ADSP-BF544
ADSP-BF548	ADSP-BF549
ADSP-BF561	

TigerSHARC (ADSP-TSxxx) Processors

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

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

2 VISUALDSP++ 5.0 NEW FEATURES AND ENHANCEMENTS

VisualDSP++ 5.0 has new features and enhancements designed to increase productivity and shorten application development cycles. This chapter describes the features and enhancements introduced in VisualDSP++ 5.0.

The information is presented as follows.

Licensing and Registration

VisualDSP++ 5.0 provides the following enhancements; further details of which can be found in the new *VisualDSP++ 5.0 Licensing Guide*.

Support of MAC Address for Use as Host ID

The MAC address is a unique number associated with the computer's network card.

From VisualDSP++ 5.0 onwards, if you do not have a hard drive on your machine (or when installing to a system that does not have a `C:\` drive), the MAC address is used by default as the host ID.

License Borrowing from the License Server

VisualDSP++ 5.0 now provides the ability to "borrow" floating licenses from the server machine for a predetermined length of time. This is particularly useful if you use a laptop and want to develop an application when traveling, or when you are at home.

ⓘ This feature requires that the license server is running the latest release of the license server software.

Refer to "Licensing Changes" on page 3-2 for information on obtaining this software.

Support for Remote Desktop

Users can now access VisualDSP++ through Remote Desktop (a Microsoft application that allows them to access one computer remotely from another computer) using both node-locked and floating licenses. Provided that the users can connect to their office's network remotely (using VPN, or a similar application), Remote Desktop allows them to use VisualDSP++ on their computers at work when logging in remotely from home.

ⓘ This feature is available only with license files created by VisualDSP++ 5.0 or greater.

Use of license files in VisualDSP++ 5.0 created with earlier releases do not support Remote Desktop. To upgrade an older license file to the version 5.0 format, contact Analog Devices registration support team at:

`processor.tools.registration@analog.com`

Modified License Strings

These changes have lead to a modification to the format of VisualDSP++ license strings, and therefore validation codes. This means that when you register your serial number and host ID, **two** validation codes will be e-mailed to you. One is valid for VisualDSP++ 4.5 and earlier versions; the other is valid for VisualDSP++ 5.0 and later versions.

ⓘ Ensure that you install the appropriate validation code for the version of VisualDSP++ you are using.

Refer to the “Changes in VisualDSP++ 5.0” section in the *VisualDSP++ 5.0 Licensing Guide* for details.

VisualDSP++ IDDE

The VisualDSP++ 5.0 integrated development and debugging environment (IDDE) introduces:

- "New Blackfin Processor Support in VisualDSP++ 5.0" on page 2-4
- "New Project Types" on page 2-5
- "Support for MISRA-C" on page 2-5
- "Call Stack Window Enhancements" on page 2-5
- "Binary File Support for Filling and Dumping" on page 2-5
- "Core File Support" on page 2-6
- "Stand-Alone Flash Programmer" on page 2-6
- "Custom Board Support" on page 2-7
- "Silicon Anomaly Support" on page 2-7
- "Emulator Troubleshooting Support" on page 2-8
- "Help Categories" on page 2-8

For more information about the VisualDSP++ IDDE, refer to the *VisualDSP++ 5.0 User's Guide* and online Help.

New Blackfin Processor Support in VisualDSP++ 5.0

The following new Blackfin processors are supported in VisualDSP++ 5.0.

ADSP-BF522	ADSP-BF525	ADSP-BF527
ADSP-BF542	ADSP-BF544	ADSP-BF548
ADSP-BF549		

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

New Project Types

The **Project Wizard** has been changed to simplify the process of creating a new project. Available options depend on the selected processor family and processor.

Support for MISRA-C

VisualDSP++ 5.0 adds support for the MISRA-C Programming Guidelines for Safety-Critical Systems. You can enable and configure this facility for Blackfin and SHARC projects through the new **Compile: MISRA-C** page of the **Project Options** dialog box. See "MISRA-C" on page 2-13 for details.

Call Stack Window Enhancements

When debugging an application that contains no debug information and no symbol information, addresses and offsets display in the **Call Stack** window.

Refer to "Call Stack Window" in VisualDSP++ online Help for details.

Binary File Support for Filling and Dumping

You can dump memory to (and fill memory from) a binary file. You can also append data to an existing file when dumping memory.

Core File Support

You can dump the entire state (registers and memory content) of a stopped application to a core file. Later, the core file can be loaded by the VisualDSP++ IDDE to restore the saved state so the application can be examined offline.

Use this capability to:

- Save the state of a misbehaving application for later analysis or sharing with a colleague. A snapshot of an application can be taken and loaded onto a different system.
- Understand all the effects of a program sequence, interrupt handler, and so on. For example, you can generate a core file, step over a subroutine, and then generate a second core file. Both core files can be converted to text file format, and an external `diff` utility can be used to show all the effects of the subroutine.

ⓘ Use the `elfdump` utility to dump the contents of a core file into a text format, allowing you to more easily view the contents. Refer to the *VisualDSP++ 5.0 Linker and Utilities Manual* for information about `elfdump`.

Stand-Alone Flash Programmer

This new utility provides flash programming support between the development/prototype stage and early pre-production runs. The Stand-Alone Flash Programmer enables the development engineer to script or automate this process with a license-free tool, allowing the manufacturing technician to repeatedly program any number of boards prior to major production.

Custom Board Support

Custom board support enables a user to specify register reset values on his/her custom board, view the content of any register on this custom board (not just Analog Devices processor memory-mapped registers [MMRs]), and display his/her custom register windows via the VisualDSP++ IDDE's **Register** menu. (You can configure menu items that open the customized register windows.) For details, refer to online Help.

Silicon Anomaly Support

From online Help, use the hyperlink to access the Analog Devices Web site to stay current on processor anomalies.

To view processor anomalies from Help, search for and open the "**Silicon Anomaly Tools Support**" topic. Simply click any of the listed processors, which are hyperlinks that access the Analog Devices Web site. The Web site is updated regularly so you can stay current on known processor anomalies.

A stylesheet improves the readability of silicon anomaly support files, which reside in the …`\System\ArchDef` subdirectory of your VisualDSP++ installation directory.

The silicon anomaly support files have been expanded with additional information on what the tools do to work around the anomalies. Also, the compiler workaround descriptions have more detailed information, and the support files now include assembler, `.ldf` file, and run-time library workaround information.

Emulator Troubleshooting Support

New documentation is included to help you troubleshoot emulator problems. Topics include JTAG scans for ICE Test, target passes ICE Test but fails to connect with VisualDSP++, memory verification errors, common causes of JTAG connection failures, and common causes of EZ-KIT Lite connection failures. Refer to online Help.

Help Categories

VisualDSP++ Help can be filtered by specifying a preference or by launching Help via the Windows **Start** menu. Either way, you select a Help category.

VisualDSP++ provides three processor-specific Help categories, one for each processor family. Each Help category (for example, Blackfin processor family Help) displays information pertinent to that specific family of processors. By selecting a Help category, in effect, you remove information about other families of processors from Help; this improves your ability to quickly locate information in Help, especially when running a “search” or looking up an entry in the Help Index.

Assembler

New features include:

- "New Processor Support" on page 2-9
- "Assembler Feature and Predefined Macros" on page 2-9
- "New Command-Line Switches" on page 2-10
- "New Directive" on page 2-11

Refer to "Assembler/Linker Changes" on page 3-4 for information about a change to the `.ALIGN` directive.

For detailed information on the assembler and preprocessor features, refer to the *VisualDSP++ 5.0 Assembler and Preprocessor Manual* and online Help.

New Processor Support

The assembler and preprocessor add support for several new Blackfin processors; the new processors are listed in "New Blackfin Processor Support in VisualDSP++ 5.0" on page 2-4.

Assembler Feature and Predefined Macros

Table 2-1 describes the new assembler feature macros.

Table 2-1. Assembler Feature Macros for Blackfin Processors

Blackfin Processor	**Description**
`-D__ADSPBF522__=1`	Present when running `easmblkfn -proc ADSP-BF522` with an ADSP-BF522 processor
`-D__ADSPBF525__=1`	Present when running `easmblkfn -proc ADSP-BF525` with an ADSP-BF525 processor

Table 2-1. Assembler Feature Macros for Blackfin Processors (Cont'd)

Blackfin Processor	Description
`-D__ADSPBF527__=1`	Present when running `easmblkfn -proc ADSP-BF57` with an ADSP-BF527 processor
`-D__ADSPBF542__=1`	Present when running `easmblkfn -proc ADSP-BF542` with an ADSP-BF542 processor
`-D__ADSPBF544__=1`	Present when running `easmblkfn -proc ADSP-BF544` with an ADSP-BF544 processor
`-D__ADSPBF548__=1`	Present when running `easmblkfn -proc ADSP-BF548` with an ADSP-BF548 processor
`-D__ADSPBF549__=1`	Present when running `easmblkfn -proc ADSP-BF549` with an ADSP-BF549 processor

-D__VISUALDSPVERSION__ Predefined Macro

The `-D__VISUALDSPVERSION__` predefined macro provides VisualDSP++ product version information. The macro allows a pre-processing check to be placed within code. It can be used to differentiate between VisualDSP++ releases and updates. This macro applies to all Analog Devices processors.

This predefined macro is available for use with the assembler drivers and the preprocessor program.

New Command-Line Switches

Table 2-2 describes the new assembler command-line switches.

Table 2-2. New Assembler Command-Line Switches

Switch Name	Description
`-anomaly-detect id1[,id2...]`	Issues a warning or an error for an anomaly id
`-anomaly-warn [id1[,id2]|all|none]`	Checks assembly instructions against hardware anomalies. **NOTE:** Blackfin processors ONLY

Table 2-2. New Assembler Command-Line Switches (Cont'd)

Switch Name	Description
`-anomaly-workaround id1[,id2...]`	Implements a workaround for an anomaly id
`-expand-symbolic-links`	Enables support for Cygwin-style paths
`-expand-windows-shortcuts`	Enables support for Windows shortcuts
`-no-anomaly-detect id1[,id2...]`	Do not issue a warning or an error for an anomaly ID
`-no-anomaly-workaround id1[,id2...]`	Do not implement a workaround for an anomaly ID
`-no-expand-symbolic-links`	Disables support for Cygwin-style paths
`-no-expand-windows-shortcuts`	Disables support for Windows shortcuts
`-no-temp-data-file`	Suppresses writing temporary data to a memory (disk). **Note:** Blackfin processors only.
`-no-source-dependency`	Suppresses output of the source filename in the dependency output produced when `-M` or `-MM` has been specified

New Directive

Table 2-3 describes the new assembler directive.

Table 2-3. New Assembler Directive

Directive	Description
`.RETAIN_NAME`	Stops the linker from eliminating a symbol

Compiler and Run-Time Library for Blackfin Processors

This section lists and briefly describes the most notable new features and enhancements of the C/C++ compiler and library for Blackfin processors.

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

- "New Processor Support" on page 2-13
- "MISRA-C" on page 2-13
- "New Compiler Switches" on page 2-13
- "New Pragmas" on page 2-15
- "New Section Identifiers" on page 2-16
- "New Predefined Macros" on page 2-16
- "Additional Path Support" on page 2-17
- "New Built-in Functions" on page 2-17
- "Fractional Values in C" on page 2-18

For detailed information on these features, refer to the *VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processors* and online Help.

New Processor Support

The new Blackfin processors supported in the VisualDSP++ 5.0 release are listed on page 2-4.

MISRA-C

The compiler supports checking for *MISRA-C:2004 Guidelines for the use of the C language in critical systems*.

To allow the compiler to verify MISRA-C conformance completely, it is sometimes necessary to generate a number of output files with the `.misra` suffix. The output files are used by the prelinker when validating MISRA-C conformance across multiple source files. Refer to the *VisualDSP ++ 5.0 Blackfin Compiler and Library Manual* for more information on MISRA-C feature.

New Compiler Switches

New compiler switches supported in the VisualDSP++ 5.0 release are:

Switch	Description
`-annotate`	Enables assembly annotations
`-expand-symbolic-links`	Provides support for Cygwin path extensions within command line switches and #include pre-processor directives
`-expand-windows-shortcuts`	Provides support for Windows shortcuts within command-line switches and `#include` pre-processor directives
`-list-workarounds`	Lists all supported workarounds for the targeted silicon revision
`-no-auto-attrs`	Directs the compiler not to emit automatic attributes based on the files it compiles
`-no-annotate-loop-instr`	Disables the production of additional loop annotation information by the compiler (default mode)

Switch	Description
`-no-assume-vols-are-mmrs`	Directs the compiler not to apply workarounds for MMR-related silicon errata to arbitrary `volatile`-qualified memory accesses
`-progress-rep-opt`	Issues a diagnostic message each time the compiler starts a new optimization pass on the current function. Equivalent to `-Wwarn=cc1473`. Replaces previous switches `-progress-rep-gen-opt` and `-progress-rep-mc-opt`
`-no-progress-rep-timeout`	Prevents the compiler from issuing a diagnostic during excessively long compilations
`-no-sat-associative`	Saturating addition is not associative
`-no-std-templates`	Disables the lookup of names used in templates
`-no-workaround` *`workaround_id`*	Disables specific hardware anomaly workarounds within the compiler
`-overlay-clobbers` *`regs`*	Specifies the registers assumed to be clobbered by an overlay manager
`-sat-associative`	Saturating addition is associative
`-std-templates`	(C++ mode): Enables the lookup of names used in templates
`-misra`	Enables checking for MISRA-C:2004 Guidelines, some rules relaxed
`-misra-linkdir`	Specifies directory in which `.misra` files are to be generated for checking rules across modules
`-misra-no-cross-module`	Enables checking for MISRA-C:2004 Guidelines. Prevents generation of `.misra` files
`-misra-no-runtime`	Enables checking for MISRA-C:2004 Guidelines. Inhibits generation of additional code to check rules requiring run-time checking
`-misra-strict`	Enables checking for MISRA-C:2004 Guidelines
`-misra-suppress-advisory`	Enables checking for MISRA-C:2004 Guidelines, but inhibits checking for advisory rules
`-misra-testing`	Enables checking for MISRA-C:2004 Guidelines, but inhibits checking for rules 20.4, 20.7, 20.8, 20.9, 20.10, 20.11, and 20.12 advisory

Switch	Description
`-Wmis_suppress` *`rule_number`*`[,`*`rule_number`*`...]`	Suppresses checks for listed *`rule_number`*s
`-Wmis_warn` *`rule_number`*`[,`*`rule_number`*`...]`	Produces warnings for listed *`rule_number`*s

New Pragmas

VisualDSP++ 5.0 introduces the following new pragmas.

#pragma generate_exceptions_tables

This pragma may be applied to a C function definition. It instructs the compiler to generate tables which enable C++ exceptions to be thrown through executions of this function. An alternative to using `#pragma generate_exceptions_tables` is to compile C files with the `-eh` (enable exception handling) switch which, for C files, is equivalent to using the pragma before every function definition.

#pragma inline

This pragma instructs the compiler to inline the function if it is considered desirable. The pragma is equivalent to specifying the `inline` keyword, but may be applied when the `inline` keyword is not allowed (such as when compiling in MISRA-C mode).

#pragma misra_func(*arg*)

This pragma is placed before a function prototype. It is used to support MISRA rules 20.4, 20.7, 20.8, 20.9, 20.10, 20.11, and 20.12. The *arg* indicates the type of function with respect to the MISRA Rule. Functions following Rule 20.4 would take *arg* `heap`, 20.7 *arg* `jmp`, 20.8 *arg* `handler`, 20.9 *arg* `io`, 20.10 *arg* `string_conv`, 20.11 *arg* `system` and 20.12 *arg* `time`.

#pragma pgo_ignore

This pragma tells the compiler that no profile should be generated for this function, when using Profile-Guided Optimization. This is useful when the function is concerned with error checking or diagnostics.

New Section Identifiers

The compiler supports these new section identifiers:

Section Identifier	Description
`strings`	Controls the placement of string literals
`autoinit`	Controls placement of data used to initialize aggregate autos
`alldata`	Controls placement of `data`, `constdata`, `bsz`, `strings` and `autoinit` all at once

New Predefined Macros

New predefined macros are:

Macro	Function
`_MISRA_RULES`	Defines `_MISRA_RULES` as 1 when compiling in MISRA-C mode.
`__VISUALDSPVERSION__`	The preprocessor defines this macro to be an eight-digit hexadecimal representation of the VisualDSP++ release, in the form `0xMMmmuurr`, where: – `MM` is the major release number – `mm` is the minor release number – `uu` is the update number – `rr` is “00”, and reserved for future use For example, VisualDSP++5.0 Update 1 would be `0x05000100`.

Additional Path Support

The compiler driver and compiler provide support for extensions to standard Windows path names. Windows shortcuts and Cygwin paths are both supported. The extensions are controlled independently by compiler switches. Both features are disabled by default. Four new compiler switches have been added for this feature:

- **-expand-symbolic-links**
  Instructs the compiler to recognize Cygwin path extensions within command line switches and #include pre-processor directives.
- **-no-expand-symbolic-links**
  Disables compiler support for Cygwin path extensions.
- **-expand-windows-shortcuts**
  Instructs the compiler to recognize Windows shortcuts within command line switches and #include pre-processor directives.
- **-no-expand-windows-shortcuts**
  Disables compiler support for Windows shortcuts.

New Built-in Functions

The following are new built-in compiler functions for the Blackfin compiler.

Built-in Function	**Description**
`cmac_fr16_s40`	Complex multiply accumulate using internal 40-bit operations
`cmsu_fr16_s40`	Complex multiply subtract using internal 40-bit operations
`multr_fr1x32x32`	Same as `mult_fr1x32x32` but with additional rounding precision

For more information, see "Saturation and Optimization" on page 3-10.

Fractional Values in C

The compiler supports the use of the suffixes “`r16`” and “`r32`” to indicate literal values that should be interpreted as fractional values, and converted to the hexadecimal bit-pattern that represents that value in a signed fixed-point representation. For example,

```
#include <fract.h>
fract16 x = 0.5r16;   // x is set to 0x4000
fract32 y = 0.5r32;   // y is set to 0x40000000
```

These suffixes are a syntactic convenience; the literal values are still signed-integer types, and are subject to the usual integer arithmetic and type conversion and promotions.

Compiler and Run-Time Library for SHARC Processors

This section lists and briefly describes the most notable new features and enhancements of the C/C++ compiler and library for SHARC processors.

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

- “MISRA-C” on page 2-19
- “New Compiler Switches” on page 2-19
- “New Pragmas” on page 2-21
- “New Section Identifiers” on page 2-22
- “New Predefined Macros” on page 2-22

For more information about these features, refer to the *VisualDSP++ 5.0 C/C++ Compiler Manual for SHARC Processors*, *VisualDSP++ 5.0 Run-Time Library Manual for SHARC Processors*, and online Help.

MISRA-C

The compiler supports checking for *MISRA-C:2004 Guidelines for the use of the C language in critical systems.*

To allow the compiler to verify MISRA-C conformance completely, it is sometimes necessary to generate a number of output files with the `.misra` suffix. The output files are used by the prelinker when validating MISRA-C conformance across multiple source files. Refer to the *VisualDSP ++ 5.0 SHARC Compiler Manual* for more information on MISRA-C feature

New Compiler Switches

New compiler switches supported in the VisualDSP++ 5.0 release are:

Switch	Description
`-annotate`	Enables assembly annotations
`-list-workarounds`	Lists all supported workarounds for the targeted silicon revision
`-misra`	Enables checking for MISRA-C:2004 Guidelines, some rules relaxed
`-misra-linkdir`	Specifies directory in which `.misra` files are to be generated for checking rules across modules
`-misra-no-cross-module`	Enables checking for MISRA-C:2004 Guidelines. Prevents generation of `.misra` files
`-misra-no-runtime`	Enables checking for MISRA-C:2004 Guidelines. Inhibits generation of additional code to check rules requiring run-time checking
`-misra-strict`	Enables checking for MISRA-C:2004 Guidelines

Switch	Description
`-misra-suppress-advisory`	Enables checking for MISRA-C:2004 Guidelines, but inhibits checking for advisory rules
`-misra-testing`	Enables checking for MISRA-C:2004 Guidelines, but inhibits checking for rules 20.4, 20.7, 20.8, 20.9, 20.10, 20.11, and 20.12 advisory
`-Wmis_suppress` *`rule_number`*`[,`*`rule_number`*`...]`	Suppresses checks for listed *`rule_number`*s
`-Wmis_warn` *`rule_number`*`[,`*`rule_number`*`...]`	Produces warnings for listed *`rule_number`*s
`-no-workaround` *`workaround_id`*	Disables specific hardware anomaly workarounds within the compiler
`-no-progress-rep-timeout`	Prevents the compiler from issuing a diagnostic during excessively long compilations
`-no-sat-associative`	Saturating addition is not associative
`-overlay`	Disables the propagation of register information between functions and forces the compiler to assume that all functions clobber all scratch registers
`-overlay-clobbers` *`regs`*	Specifies the registers assumed to be clobbered by an overlay manager
`-progress-rep-opt`	Issues a diagnostic message each time the compiler starts a new optimization pass on the current function. Equivalent to `-Wwarn=cc1473`. Replaces previous switches `-progress-rep-gen-opt` and `-progress-rep-mc-opt`
`-progress-rep-timeout`	Issues a diagnostic message if the compiler exceeds a time limit during compilation
`-progress-rep-timeout-secs`	Specifies how many seconds must elapse during a compilation before the compiler issues a diagnostic on the length of compilation
`-sat-associative`	Saturating addition is associative
`-no-std-templates`	Disables the lookup of names used in templates
`-std-templates`	Enables the lookup of names used in templates

New Pragmas

VisualDSP++ 5.0 introduces the following new pragmas.

#pragma inline

This pragma instructs the compiler to inline the function if it is considered desirable. The pragma is equivalent to specifying the `inline` keyword, but may be applied when the `inline` keyword is not allowed (such as when compiling in MISRA-C mode).

#pragma generate_exceptions_tables

This pragma may be applied to a C function definition. It instructs the compiler to generate tables which enable C++ exceptions to be thrown through executions of this function. An alternative to using `#pragma generate_exceptions_tables` is to compile C files with the `-eh` (enable exception handling) switch which, for C files, is equivalent to using the pragma before every function definition.

#pragma misra_func(*arg*)

This pragma is placed before a function prototype. The pragma is used to support MISRA rules 20.4, 20.7, 20.8, 20.9, 20.10, 20.11, and 20.12. The *arg* indicates the type of function with respect to the MISRA Rule. Functions following Rule 20.4 would take *arg* `heap`, 20.7 *arg* `jmp`, 20.8 *arg* `handler`, 20.9 *arg* `io`, 20.10 *arg* `string_conv`, 20.11 *arg* `system` and 20.12 *arg* `time`.

#pragma pgo_ignore

This pragma tells the compiler that no profile should be generated for this function, when using Profile-Guided Optimization. This is useful when the function is concerned with error checking or diagnostics.

New Section Identifiers

The compiler supports these new section identifiers:

Section Identifier	Description
`pm_data`	Controls placement of initialized data declared with the `_pm` keyword
`pm_constdata`	Controls placement of constant data declared with the `_pm` keyword
`strings`	Controls placement of string literals
`autoinit`	Controls placement of data used to initialize aggregate autos

New Predefined Macros

New predefined macros are:

Macro	Function
`_MISRA_RULES`	Defines `_MISRA_RULES` as 1 when compiling in MISRA-C mode.
`__VISUALDSPVERSION__`	The preprocessor defines this macro to be an eight-digit hexadecimal representation of the VisualDSP++ release, in the form `0xMMmmuurr`, where: – `MM` is the major release number – `mm` is the minor release number – `uu` is the update number – `rr` is “00”, and reserved for future use For example, VisualDSP++5.0 Update 1 would be `0x05000100`.

Compiler and Library for TigerSHARC Processors

New Compiler Switches

New TigerSHARC compiler switches supported in VisualDSP++ 5.0 release are:

Table 2-4. New TigerSHARC Command-Line Switches

Switch	Description
`-annotate`	Enables assembly annotations
`-full-dependency-inclusion`	(C++ mode): Ensures re-inclusion of implicitly included files when generating dependency information
`-list-workarounds`	Lists silicon errata workarounds supported by the compiler
`-no-progress-rep-timeout`	Prevents the compiler from issuing a diagnostic during excessively long compilations
`-no-workaround workaround_id`	Disables a compiler workaround for a silicon errata
`-progress-rep-timeout`	Issues a diagnostic message if the compiler exceeds a time limit during compilation
`-progress-rep-timeout-secs`	Specifies how many seconds must elapse during a compilation before the compiler issues a diagnostic on the length of compilation
`-no-std-templates`	(C++ mode): Disables the lookup of names used in templates
`-std-templates`	(C++ mode): Enables the lookup of names used in templates

New Pragmas

#pragma pgo_ignore

This pragma directs the compiler to generate no profile for the function when using Profile-Guided Optimization. The pragma is useful when the function is concerned with error checking or diagnostics.

New Predefined Macros

New predefined macro is:

Macro	Function
`__VISUALDSPVERSION__`	The preprocessor defines this macro to be an eight-digit hexadecimal representation of the VisualDSP++ release, in the form `0xMMmmuurr`, where: – `MM` is the major release number – `mm` is the minor release number – `uu` is the update number – `rr` is "00", and reserved for future use For example, VisualDSP++5.0 Update 1 would be `0x05000100`.

Linker and Utilities

The VisualDSP++ 5.0 linker, Linker Description Files, and utility programs are upgraded to operate more efficiently on Blackfin, TigerSHARC, and SHARC processors.

Linker map file generation is improved to report correct sizes for regions allocated by the `RESERVE()` command and `meminit`.

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

- "Updated List of LDF Keywords" on page 2-25
- "Built-In Macro" on page 2-25
- "New Utility: elfpatch" on page 2-26
- "New Switch" on page 2-26

For more information, refer to the *VisualDSP++ 5.0 Linker and Utilities Manual*.

Updated List of LDF Keywords

The following new `.ldf` file keyword applies to all supported processors:

```
COMMON_MEMORY
```

Built-In Macro

The preprocessor provides the following built-in predefined macro:

```
__VISUALDSPVERSION__
```

The macro defines this to be an eight-digit hexadecimal representation of the VisualDSP++ , in the form `0xMMmmuurr`, where:

- – `MM` is the major release number
- – `mm` is the minor release number
- – `uu` is the update number
- – `rr` is “00”, and reserved for future use

For example, VisualDSP++5.0 Update 1 would be `0x05000100`.

New Utility: elfpatch

The executable and linking format (ELF) file patch (`elfpatch`) utility allows the bits of an ELF section to be extracted or replaced from a specified file.

New Switch

The following new linker command-line switch is now available:

Table 2-5. New Linker Command-Line Switch

Switch	Description
`-reserve-null`	Directs the linker to reserve the first 4 addressable units (words) in memory

Loader and Splitter for Blackfin Processors

The loader utility (`elfloader.exe`) has been enhanced and modified to support new features and new Blackfin processors. The loader utility's modifications and enhancements are summarized in the following sections.

- "Support for New Processors" on page 2-27
- "Support for Multiple Initialization Input Blocks" on page 2-28
- "Automatic Inclusion of ROM Sections in Boot Streams" on page 2-28
- "Support for Callback and Indirect Blocks" on page 2-28
- "Support for Save and Quickboot Blocks" on page 2-29
- "-pFlag Parameter Interface Management" on page 2-29

Support for New Processors

The loader and splitter utilities add support for ADSP-BF52x/BF54x Blackfin processors; the new processors are listed in "New Blackfin Processor Support in VisualDSP++ 5.0" on page 2-4.

The loader utility generates a boot stream in a new boot stream format (for these new processors), including the new format for block headers and flags. The ADSP-BF52x and ADSP-BF54x processor boot sequences, boot streams, block headers, and flags are documented in "Loader/Splitter for ADSP-BF52x/BF54x Blackfin Processors" chapter of the *VisualDSP++ 5.0 Loader and Utilities Manual*.

Support for Multiple Initialization Input Blocks

For ADSP-BF52x/BF54x processors, the loader utility removes the constraints of limiting initialization (`init`) blocks to a singular instance of an executable file at the start of a loader boot stream. VisualDSP++ 5.0 users are able to execute multiple in-application `init` block calls at any time during the boot process.

The loader utility supports placement of multiple initialization calls in a boot stream via the `-initcall` command-line switch. The `-initcall` syntax also enables control over multiple init block placements and invocations, including re-entrant blocks. For more information about `-initcall`, refer to the *VisualDSP++ 5.0 Loader and Utilities Manual* and online Help.

Automatic Inclusion of ROM Sections in Boot Streams

For ADSP-BF52x/BF54x processors, the newly introduced `-readall` command-line switch allows the loader utility to integrate fixed-position ROM data sections within a loader boot stream. The switch calls the splitter utility as a transparent sub-process to the loader utility. In a resulting loader file available in Intel hex-32 format, the splitter data is merged with the loader stream. For more information on `-readall`, refer to the *VisualDSP++ 5.0 Loader and Utilities Manual* and online Help.

Support for Callback and Indirect Blocks

For ADSP-BF52x/BF54x processors, the loader utility now is able to invoke a block that has been loaded indirectly into internal memory by init code. This is done via the `-callback` switch, which marks the block with the `CALLBACK` flag.

The `-callback` switch is used in conjunction with the `INDIRECT` flag, which makes indirect boot blocks using the TWI boot mode to validate the block's CRC32 value encoded in the block's header. Indirect blocks are placed into internal memory first and then placed by *MDMA*s into destination memory, rather then by direct DMAs. For more information on `-callback`, refer to the *VisualDSP++ 5.0 Loader and Utilities Manual* and online Help.

Support for Save and Quickboot Blocks

For ADSP-BF52x/BF54x processors running on low-power cycles, the loader utility now is able to mark internal memory blocks (marked with the `SAVE` flags) within the LDF-defined sections for saving. This is done using the `-save` command-line switch. This supports interrupt-based, run-time archiving of code and data during low-power or hibernation power cycles.

The `-quickboot` loader command-line switch marks blocks within the LDF-defined section name with the `QUICKBOOT` flag. The switch is used to mark blocks to skip on warm-boot cycles.

For more information on `-save` and `-quickboot`, refer to the *VisualDSP++ 5.0 Loader and Utilities Manual* and online Help.

-pFlag Parameter Interface Management

For ADSP-BF531/BF532/BF533/BF534/BF536/BF537/BF538/BF539 processors, the loader utility uses a previously revised mechanism for the `-pFlag` parameter handling. The allowed `-pFlag` value now is managed dynamically and varies with the processor, silicon revision, boot mode, and DMA width. All valid `-pFlag` parameter combinations are documented in Tables 3-11 through 3-13 of the *VisualDSP++ 5.0 Loader and Utilities Manual*. The loader utility generates warnings for invalid combinations.

The new parameter handling mechanism was introduced in the VisualDSP++ 4.5 November 2006 update. Any older Blackfin projects using `-pFlag` should be verified to ensure the correct setting is being used. The setting should also be verified whenever the processor, silicon revision, boot mode, or DMA width is changed in VisualDSP++. For more information, refer to the *VisualDSP++ 5.0 Loader and Utilities Manual* and online Help.

Splitter for SHARC Processors

The SHARC splitter utility (`elfspl21k.exe`) of the VisualDSP++ 5.0 release was extended to support ADSP-2126x and ADSP-2137x processors' *physical* external memory image generation, in addition to the logical external memory image generation introduced in the previous VisualDSP++ release. The logical external memory images are post-processed in the order that the physical external memory images are obtained.

- ADSP-21261/21262/21266/21267, ADSP-21362/21363/21364/21365/21366 processors obtain identical external memory architectures, and the splitter produces identical memory images for their external memory spaces.
- ADSP-21367/21368/21369 and ADSP-21371/21375 processors obtain identical external memory architectures and the splitter produces identical memory images for their external memory spaces.

nXXX Bit Macros with Zero Values for ADSP-BF54x Processors

ADSP-BF54x def headers contain an additional macro set. For each single bit grouping within a register, there is a corresponding "`n`" version of the macro set to zero. For example,

```
/* Bit masks for SPIx_CTL */
#define SPE 0x4000   /* SPI Enable */

#define nSPE 0x0
```

These were added to provide a means for making the programmer's intention more clear. With the `n*` version, you can list each bit in the assignment. For example,

```
*pSPI_CTL = nSPE|nWOM|MSTR|nCPOL|CPHA|nLSBF|
SIZE|nEMISO|nPSSE|nGM|nSZ|RDBR_DMA;
```

They are not intended as negation macros. (A code developer can negate a macro for dynamic clearing of individual bits using the `&= ~MACRO` syntax.)

VDK

The following new VDK features and enhancements have been added to VisualDSP++ 5.0:

- A new VDK signal type, mutex, for mutual thread exclusion
- A new mechanism for replacing the VDK history logging with a custom subroutine
- New file attributes that relate to the content of the particular VDK file; for example, semaphores. The file attribute provides greater flexibility in the mapping of VDK code. For more information

about file attributes, see "*File Attributes*" in the C/C++ documentation for your processor. For more information on VDK file attributes, see "*VDK File Attributes*" in the VDK documentation and online Help.

- New APIs for obtaining the information provided by the VDK Status Window. For a complete list of the APIs, see "*Application Status Information Functions*" in the VDK documentation and online Help.

See the *VisualDSP++ 5.0 Kernel (VDK) User's Guide* and online Help for details.

Device Drivers and System Services

VisualDSP++ 5.0 includes the following device drivers and system services new features and enhancements for Blackfin processors.

File System Service

The File System Service (FSS) provides a portable and extensible means of accessing mass storage media from the Blackfin processor. Support for the ADSP-BF548 EZ-KIT Lite development board is provided with VisualDSP++ 5.0 for FAT file systems on the ATA and the Secure Digital Host interfaces for access to the attached hard disk drive and supplied SD card.

Once the FSS is initialized and registered with the extensible standard C interface provided with the VisualDSP++ 5.0 I/O library, calls to `fopen()`, `fread()`, and so on are routed automatically to the FSS to provide seamless access to files on the various mass storage media.

The FSS has been designed such that device drivers for additional file systems and physical interfaces can be added to an application very simply, provided they are written to comply with the published guidelines.

EPPI

The EPPI device driver supports direct connection of EPPI to LCD panels, parallel A/D and D/A converters, video encoders and decoders, CMOS sensors, and other general-purpose peripherals. The ADV7183 video decoder, ADV717x video encoder, and the LQ043T1DG01 LCD drivers leverages the EPPI driver to perform its video-related tasks. Example EPPI configuration tables for various operating modes are provided in the EPPI driver document. The EPPI device driver has been tested on the ADSP-BF548 EZ-KIT Lite development board.

Pixel Compositor

The pixel compositor driver provides a complete, easy, and flawless control over the pixel compositor. The driver contains built-in look-up tables to support all 22 mainstream (and up to 7 special case) color space conversion and data overlay combinations. The pixel compositor driver has been tested on the ADSP-BF548 EZ-KIT Lite development board.

Touch Screen

The AD7877 driver provides an effective and easy way to control the Analog Devices AD7877 touch screen controller. The driver can be configured to monitor the AD7877 interrupt signals (`PENIRQ`, `DAV`, `ALERT`) and take appropriate actions and post callbacks, which highly reduce application code overhead. The AD7877 driver has been tested on the ADSP-BF548 EZ-KIT Lite development board.

USB

The USB driver stack encompasses a set of USB class drivers, peripheral drivers, and a USB core library. The ADSP-BF548 USB OTG controller driver and PLX NET2272 controller driver support USB device mode, in which the Blackfin processor becomes a peripheral; this capability permits a host PC to have visibility over the contents of the hard disk drive on a ADSP-BF548 EZ-KIT Lite development board. Device mode loopback and device mode mass storage examples are included.

The USB driver stack has been tested on the ADSP-BF548 EZ-KIT Lite development board and on the ADSP-BF533 EZ-KIT Lite development board with a Blackfin USB-LAN EZ-Extender® attached.

Rotary Wheel

The rotary wheel driver allows the application to control the rotary counter interface of the ADSP-BF548 Blackfin processor. The device driver supports the following features of the counter module:

- Operation mode, including `QUAD_ENC`, `BIN_ENC`, `UD_CNT`, `DIR_CNT`, and `DIR_TMR`
- Inputs de-bounce filtering
- Zero marker operation, including push-button, ZM-zeroes-counter, ZM-error, and Zero-once mode
- Boundary comparison modes, including `BND_COMP`, `BND_ZERO`, `BND_CAPT`, and `BND_AEXT`
- Counter events (interrupts)

The device driver has been tested on the ADSP-BF548 EZ-KIT Lite development board.

Keypad

The keypad driver allows the application to control the keypad interface (KPAD) of the ADSP-BF548 Blackfin processor. The driver supports press-release mode and has no hardware dependencies. This driver has been tested on the ADSP-BF548 EZ-KIT Lite development board.

3 VISUALDSP++ 5.0 MAJOR CHANGES

This chapter summarizes major changes in VisualDSP++ 5.0 compared with the VisualDSP++ 4.5 release.

The chapter details:

Please note that new features and enhancements are listed in Chapter 2, and all obsolete and removed features are listed in Chapter 4. All newly-supported processors are listed in "New Blackfin Processor Support in VisualDSP++ 5.0" on page 2-4.

Licensing Changes

The following sections describe changes to the floating license server tools and validation codes.

Floating License Server Tools Upgrade

The floating license server tools for VisualDSP++ 5.0 have been upgraded to version 2.00.7. This upgrade enables support for license borrowing as well as MAC address authentication on the server.

If your site uses floating licenses and you upgrade to VisualDSP++ 5.0, upgrade your floating license server tools to version 2.00.7. This server upgrade is compatible with all VisualDSP++ versions greater than and including 4.0.

The latest license server tools can be found at:
`http://www.analog.com/processors/technicalSupport/toolsUpgrades.html`

Change to Licensing Validation Codes

In VisualDSP++ 5.0, support has been added to use FLEXnet license borrowing, to allow use of a MAC address as host ID, and to permit node-locked licenses to work with Remote Desktop (refer to "Licensing and Registration" on page 2-2 for further information).

These changes have lead to a modification to the format of VisualDSP++ license strings, and therefore validation codes. This means that when you register your serial number and host ID, two validation codes will be e-mailed to you. One is valid for VisualDSP++ 4.5 and earlier versions; the other is valid for VisualDSP++ 5.0 and later versions.

The `license.dat` files created by earlier versions of VisualDSP++ will be accepted by VisualDSP++ 5.0, but will not permit the use of the new features. If you copy an existing `license.dat` file from an earlier version of

VisualDSP++ during installation of VisualDSP++ 5.0 and later wish to use any of the new features, you must move your `license.dat` file out of the way, re-install your serial numbers, re-register them and re-validate them. For more information see the new *VisualDSP++ 5.0 Licensing Guide*.

Default Silicon Revision Changes

The default silicon revisions for some processors have been changed to reflect the availability of new processor revisions. The default silicon revision for all processors is now the latest revision.

To identify the default silicon revision for a given processor, refer to the silicon anomalies section of the `ADSP-`*xxxxx*`-compiler.xml` file found in the `System/ArchDef` subdirectory of the VisualDSP++ installation, where *xxxxx* is the processor you are using (for example, `BF533`). The default is specified by the command-line-default attribute of the silicon-revisions element.

For example:

```
<silicon-revisions command-line-default="0.5">
```

The default silicon revision changes only have an effect when **Revision** in the **Project Options** dialog box is set to **Automatic** and no hardware is connected, or no silicon revision is specified on the command line.

Assembler/Linker Changes

In VisualDSP++ 5.0, there are the following changes in the assembler and linker operation.

- The description of the `.ALIGN` directive is changed as follows:

  "The linker stops allocating padding for symbols aligned by 16 or more."

- In previous releases, the assembler would attempt to determine whether a floating-point literal value should be interpreted as an IEEE floating-point constant or a fractional constant based on context. This was ambiguous and led to incorrect data. In VisualDSP++5.0, the "`r`" suffix is required for fractional constants to remove ambiguity.

- Fractional (fract) constants are specially marked floating-point constants to be represented in fixed-point format. A fract constant uses the floating-point representation with a trailing "`r`", where r stands for `fract`. Now, an error message (`ea5002`) will be reported when a fractional value without a trailing "`r`" is detected.

L1_scratch Sections Inputs use NO_INIT Qualifier

The Blackfin `.ldf` files generated by the Project Wizard have been modified to make `MEM_L1_SCRATCH` (L1 scratch memory output section) `NO_INIT` qualified. This is because the Boot ROM and Memory Initializer use L1 Scratchpad as stack and working space during booting, and therefore cannot safely map initialized data to this memory area. Ensure that you do not have initialized data mapped to section "`L1_scratch`". If your application maps uninitialized data to input section "`L1_scratch`", ensure that the section is marked as `NO_INIT`, or the linker will raise warning `li2131`.

For example, in C you might require to change:

```
#pragma section("L1_scratch")
char alt_heap[32];
```

to:

```
#pragma section("L1_scratch", NO_INIT)
char alt_heap[32];
```

In assembly, the change would be from:

```
.section L1_scratch;
.byte _alt_heap[32];
```

to:

```
.section/NO_INIT L1_scratch;
.byte _alt_heap[32];
```

Common Compiler Changes

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

Increased Code Motion for asm Constructs

The `asm` constructs describe their behavior with inputs, outputs and register/memory clobber declarations. By default, it is assumed that these declarations are the *only* effects of the `asm` construct and as such, the compiler is free to move or remove `asm` constructs where analysis shows they are redundant, or do not interfere with the rest of the function.

The VisualDSP++ 5.0 compiler is better able to detect these opportunities, and therefore will move or remove more `asm` constructs than previously. If an `asm` construct has additional side-effects not indicated by its inputs and outputs, the `asm` construct must be declared as `volatile` to prevent the compiler making unwarranted assumptions.

FORCE_CONTIGUITY Changes

C++ exceptions and global C++ constructors and destructors rely for their operation on tables assembled in memory during the linking process. These tables are generated by the compiler on a per-module basis and mapped into contiguous memory by the linker. If the tables are not contiguous, undefined behavior can occur at runtime. As of this release, the `.ldf` files ensure contiguity by using the `FORCE_CONTIGUITY` linker directive. This change can cause some differences in how the linker maps other code and data.

Cycle Count Macro

With previous releases of VisualDSP++ the parameterized macros that are defined in the header files `cycle_count.h` and `cycles.h` had expanded into multiple statements. This implementation led to some unexpected side-effects when, for example, the instructions generated by the following statements would not result in the code that was intended:

```
if (condition)
   STOP_CYCLE_COUNT(cnt2, cnt2);
```

Because the macro expands into multiple statements, only the first statement of the expanded macro will be executed conditionally. The remaining statements will always be executed and will therefore lead to incorrect cycle counts being generated.

To address this issue, the macros are now implemented using a statement block. With the previous cycle count macros being a sequence of instructions it was valid to use the macros without a trailing semi-colon. To assist with porting existing code, the compile-time macro `__USE_CYCLE_MACRO_REL45__` can be defined to enable legacy support and avoid any modifications to the existing use of the cycle count macros.

-workaround all Change

Under previous releases, the `-workaround` switch accepted the parameter '`all`"; it enabled all workarounds, which was not a clearly-defined set. As of this release, the "`-workaround all`" option is equivalent to "`-si-revision any`", in that it enables any workarounds that apply to any revision of the specified target processor.

Declaration of Compiler Built-in Functions

All compiler built-in functions are now defined in a new header file, `builtins.h`, which is included by the platform header file. This means that the platform header file for VisualDSP++ 5.0 will declare more names than for VisualDSP++ 4.5. Long names for built-in functions are always declared, with a prefix of "`__builtin_`". The double-underscore prefix is reserved by the C standard for use by the compiler, so these names should never conflict with user code. Short names are defined for many built-in functions, and may be disabled altogether by the `-no-builtins` switch or on a case-by-case basis using preprocessor defines of the form `__DISABLE_`*`NAME`*, the precise details of which may be found in `builtins.h`.

If you get an error with a function definition in `builtins.h`, it is almost certainly because you have a function or a preprocessor define which has the same name as a built-in. Please check, and either remove your own definition or disable the compiler's definition as described above.

Built-in functions now defined in `ccblkfn.h`:

abs_fr1x32	abs_fr2x16	abs_i1x32	add_as_fr2x16
add_fr1x16	add_fr1x32	add_fr2x16	add_i4x8
add_i4x8_r	add_sa_fr2x16	add_u2x16	addclip_hi
addclip_hir	addclip_lo	addclip_lor	adi_core_b_enable
adi_core_id	align16	align24	align8
aligned	avg_i2x8_hi	avg_i2x8_hir	avg_i2x8_hit
avg_i2x8_hitr	avg_i2x8_lo	avg_i2x8_lor	avg_i2x8_lot
avg_i2x8_lotr	avg_i4x8	avg_i4x8_r	avg_i4x8_t
avg_i4x8_tr	bytepack	cmplx_add	cmplx_mac
cmplx_mac_s40	cmplx_msu	cmplx_msu_s40	cmplx_mul
cmplx_mul_s40	cmplx_sub	compose_i64	dealloca
diff_hl_fr2x16	diff_lh_fr2x16	emuclk	expected_false
expected_true	extract_hi	extract_lo	flush
flushinv	flushinvmodup	flushmodup	funcsize
halt	heap_address_from_ index	iflush	iflushmodup
lmax	lmin	loadbytes	lvitmax2x16
max	max_fr1x16	max_fr1x32	max_fr2x16
min	min_fr1x16	min_fr1x32	min_fr2x16
mult_fr1x16	mult_fr1x32	mult_fr1x32x32	mult_fr1x32x32NS
mult_fr2x16	multr_fr1x16	multr_fr1x32x32	multr_fr2x16
multu_fr1x16	multu_fr2x16	negate_fr1x32	negate_fr2x16
norm_fr1x16	norm_fr1x32	ones	prefetch
prefetchmodup	round_fr1x32	sat_fr1x32	shl_fr1x16
shl_fr1x32	shl_fr1x32_clip	shl_fr2x16	shl_fr2x16_clip
shl_i1x16	shl_i1x32	shl_i2x16	shll_i1x16
shll_i1x32	shll_i2x16	shr_fr1x16	shr_fr1x32

shr_fr1x32_clip	shr_fr2x16	shr_fr2x16_clip	shr_i1x16
shrl_fr2x16	shrl_fr2x16_clip	sub_fr1x16	sub_fr1x32
sub_fr2x16	sub_i4x8	sub_i4x8_r	sub_u2x16
sum_fr2x16	testset	testset_05000248	untestset

Built-in functions now defined in the various SHARC platform headers (for example, `21060.h`):

aligned	alloca	conv_FtoR	conv_LRtoR	conv_RtoF
conv_RtoLR	dealloca	emuclk	expected_false	expected_true
logbf	mantf	pm_aligned	sysreg_bit_xor	timer_off
timer_on	timer_set			

Compiler and Library for Blackfin Processors

The following are changes in Blackfin compiler and library features.

Compiler May Modify Local Parameters

The Blackfin ABI requires all calling functions to reserve stack space for the first twelve bytes (R0-R2) of parameter space for a callee, even when the callee does not require that much space. In VisualDSP++ 5.0, the compiler makes increased use of this stack space to store temporary values, if it does not find that the space is needed for other purposes (such as storing the register-based parameter itself). Therefore, all assembly functions that call compiled functions must follow the correct procedure; with VisualDSP++ 5.0, the compiler is making more efficient use of stack space, but there is a corresponding risk that functions that violate the ABI may find live values are corrupted in the process.

Saturation and Optimization

The compiler now applies saturation consistently. Previously, the compiler would:

- Apply saturation at 32 bits when optimizing.
- Apply saturation at 40 bits when not optimizing.
- Apply these saturation limits to all accumulator-based operations, whether generated internally for performance reasons, or generated for an explicit built-in function.

The compiler now applies saturation in the following manner:

- For accumulator-based operations generated by built-in function, apply saturation at the boundary specified by the function regardless of whether optimization is enabled. New variants of the built-in functions allow specification of 40-bit saturation, while the existing built-in functions specify 32-bit saturation.
- For accumulator operations generated internally (for performance reasons), always apply saturation at the 40-bit boundary.

These rules allow the compiler to generate the fastest-performing code while reducing unintended saturation. As with earlier releases, automatic generation of saturating code can be disabled using the `-no-saturation` command-line switch.

#pragma retain_name

This pragma prevents the compiler or linker from eliminating the following symbol (code or data), even if there are no references to the symbol.

The pragma was supported in VisualDSP++ 4.0 to prevent elimination of symbols by the compiler, for cases where the only references came from outside the C/C++ sources, but did not prevent linker elimination.

The pragma was deprecated in VisualDSP++ 4.5 as redundant, as the compiler detected references from libraries or assembly-only objects, but linker elimination would still eliminate the symbol if there were no such references.

Output Section Placement Control

The compiler's support for placing code and data into specific output sections has been extended and modified for consistency. These changes affect:

- `-section id=`*`name`* switch
- `#pragma section`
- `#pragma default_section`

The compiler's `section()` keyword continues to operate as before.

The changes are as follows:

- New section identifiers are supported, for different types of compiler-generated output, for strings, aggregate initializers and constant data.
- The `-section` switch and `section/default_section` pragmas now support the same set of identifiers.
- Where section identifiers refer to groups of other section identifiers (for example, `alldata`), a flat model is now used: using a group section identifier is equivalent to specifying the section for each identifier in the group in turn. This clarifies the model where both a group identifier and an individual identifier within the group have distinct sections specified. However, it means that there are some changes to the groupings. For example, `alldata` no longer includes `switch`.

- If not all declarations of a symbol are in the scope of the same `default_section` pragma, the scope at definition takes precedence.
- The `default_section` pragma is now only valid at global scope (outside function definitions).

IPA Information Stored in Object Files

When a C/C++ file is compiled to object form with the `-ipa` switch, the compiler now stores gathered information about the source's functions within an `.ipa` section of the object file, instead of into an `.opa` temporary file, as used in previous releases. This allows means a source file can be compiled with `-ipa` and then included into a library (`.dlb`) file, and still contribute to the overall interprocedural analysis at link-time, even though the file's source is inaccessible. Library objects files do not themselves benefit from this change, as the lack of source means they cannot be recompiled in the context of the full analysis, but source files for the application being linked can gain benefit from the previously-opaque object file.

Because the object files now contain IPA information (when built with `-ipa`), they will be larger. This additional size is due to the IPA meta-data and, like debug information, does not indicate an increase in the code or data that will be linked into the final executable for the target.

Additional .pgi Files Created During Some PGO Builds

When building an application which uses both multiple source files and multiple execution profiles (`.pgo` files, used in Profile-Guided Optimization), the compiler will generate `.pgi` files which record the execution profiles used during the build. This is to ensure that, if recompilation is necessary (for example due to C++ template instantiation or as a result of Interprocedural Analysis), a consistent set of execution profiles will be used in each compilation.

If the compiler detects an inconsistency (caused by changing any of the execution profiles during the course of the build), the compiler will issue an error.

The `.pgi` files are not created when the build only requires a single source file, or a single execution profile.

Applications Using C++ Exceptions Require Recompilation

The C++ exception support in the compiler and runtime libraries has been modified, to be more efficient and more robust. The revised model uses compiler-generated tables to direct the stack-unwinding mechanism. Applications using the C++ try/throw/catch statements must be rebuilt in the following way:

- C++ source files merely need recompilation (with the `-eh` switch). The compiler will generate the necessary tables.
- C functions that may have exceptions thrown through them (that is, C++ function `x()` calls C function `y()` which calls C++ `z()`, and `z()` throws an exception caught by `x()`) must be recompiled so that the compiler can generate tables for the functions in question. This can be done using the `-eh` switch (to generate tables for all functions in the file) or using the `generate_exceptions_tables` pragma (to generate tables for specific functions).
- Assembly functions that may have exceptions thrown through them need exceptions tables crafting manually.

-no-annotate Switch Extended

In addition to disabling compiler-generated annotations, the `-no-annotate` switch disables insertion of compiler-generated comments (such as line numbering or stall-identification comments) into assembly files.

For example, while the normal assembly output might contain a mixture of line-number comments and instructions:

```
// line "x.c":2
R0 = 17;
// line 3
FP = [SP++];
```

when the `-no-annotate` switch is used, the assembly file will just contain instructions, like this:

```
R0 = 17;
FP = [SP++];
```

Radix-2 FFT Prototypes and Functions Modified

The prototypes for the radix-2 Fast Fourier Transform (FFT) functions `cfft_fr16`, `ifft_fr16` and `rfft_fr16`, as defined in `filter.h`, have been modified to facilitate enhanced implementations of these functions. Existing code that makes use of any of these functions will require to be either modified to use the new implementations or built with the compile-time macro `__USE_FFT_REL45__` defined to enable legacy support.

Previous VisualDSP++ releases of these FFT functions always used static scaling to avoid results overflowing at each FFT stage. The new implementation has been enhanced to allow selection of either static scaling, dynamic scaling or no scaling. The new implementation no longer requires the temporary array `temp`, and the `block_exponent` argument is passed by reference rather than value.

The prototype for `cfft_fr16` has been changed from:

```
void cfft_fr16(const complex_fract16 input[],
               complex_fract16       temp[],
               complex_fract16       output[],
               const complex_fract16 twiddle_table[],
               int                   twiddle_stride,
               int                   fft_size,
               int                    block_exponent,
               int                    scale_method);
```

to:

```
void cfft_fr16(const complex_fract16 input[],
               complex_fract16       output[],
               const complex_fract16 twiddle_table[],
               int                   twiddle_stride,
               int                   fft_size,
               int                   *block_exponent,
               int                   scale_method);
```

The changes made to the `ifft_fr16` and `rfft_fr16` prototypes and functions are identical to the `cftt_fr16` changes. The necessary modifications required to use the new revisions are:

- Delete passing the `temp` array parameter from all calls to cftt_fr16, `ifft_fr16` and `rfft_fr16`.
- Modify the `block_exponent` parameter to be a pointer to an integer in all calls. The new functions return the scaling steps performed using this parameter.
- Ensure that the scale method parameter `scale_method` is correct. Setting `scale_method` to 1 selects static scaling for minimal behavior changes.

Additionally, you may consider attempting to remove allocations of the array `temp` if it is unused as a consequence of no longer being passed to `cfft_fr16`, `ifft_fr16` and `rfft_fr16`. Also you may choose to use the alternate scaling methods now supported. To select dynamic scaling, which is good for maintaining precision of results, set `scale_method` to 2. For no scaling, the most optimal implementation for well conditioned inputs that are known will not overflow, set `scale_method` to 3.

To avoid modifying existing code that utilizes the `cfft_fr16`, `ifft_fr16` and `rfft_fr16` functions the compile-time macro `__USE_FFT_REL45__` can be defined. Note that any existing compiled code that call the FFT routines will use the legacy implementations.

Interrupt and Exception Handlers

Interrupt Service Routines (ISR) and hardware exception handlers implemented in C using `#pragma interrupt` and `#pragma exception` incur less overhead than before. Under previous releases, a fixed-size context was saved and restored regardless of the requirements of the routine. In VisualDSP++5.0, only those registers modified by the ISR routine are saved and restored. See also "SAVE_REGS Macro" on page 4-6.

Compiler and Library for SHARC Processors

The following are changes in SHARC compiler and library features.

#pragma retain_name

This pragma prevents the compiler or linker from eliminating the following symbol (code or data), even if there are no references to the symbol.

The pragma was supported in VisualDSP++ 4.0 to prevent elimination of symbols by the compiler, for cases where the only references came from outside the C/C++ sources, but did not prevent linker elimination.

The pragma was deprecated in VisualDSP++ 4.5 as redundant, as the compiler detected references from libraries or assembly-only objects, but linker elimination would still eliminate the symbol if there were no such references.

Output Section Placement Control

The compiler's support for placing code and data into specific output sections has been extended and modified for consistency. These changes affect:

- `-section id=`*`name`* switch
- `#pragma section`
- `#pragma default_section`

The compiler's `section()` keyword continues to operate as before.

The changes are as follows:

- New section identifiers are supported, for different types of compiler-generated output, for strings, aggregate initializers and constant data and PM data.
- The `-section` switch and `section/default_section` pragmas now support the same set of identifiers.
- Where section identifiers refer to groups of other section identifiers (for example, `alldata`), a flat model is now used: using a group section identifier is equivalent to specifying the section for each identifier in the group in turn. This clarifies the model where both a group identifier and an individual identifier within the group have distinct sections specified. However, it means that there are some changes to the groupings. For example, `alldata` no longer includes `switch`.
- If not all declarations of a symbol are in the scope of the same `default_section` pragma, the scope at definition takes precedence.
- The `default_section` pragma is now only valid at global scope (outside function definitions).

Additional .pgi Files Created During Some PGO Builds

When building an application which uses both multiple source files and multiple execution profiles (`.pgo` files, used in Profile-Guided Optimization), the compiler will generate `.pgi` files which record the execution profiles used during the build. This is to ensure that, if recompilation is necessary (for example due to C++ template instantiation or as a result of Interprocedural Analysis), a consistent set of execution profiles will be used in each compilation.

If the compiler detects an inconsistency (caused by changing any of the execution profiles during the course of the build), the compiler issues an error.

The `.pgi` files are not created when the build only requires a single source file, or a single execution profile.

Applications Using C++ Exceptions Require Recompilation

The C++ exception support in the compiler and run-time libraries has been modified, to be more efficient and more robust. The revised model uses compiler-generated tables to direct the stack-unwinding mechanism. Applications using the C++ try/throw/catch statements must be rebuilt in the following way:

- C++ source files merely need recompilation (with the `-eh` switch). The compiler will generate the necessary tables.
- C functions that may have exceptions thrown through them (that is, C++ function `x()` calls C function `y()` which calls C++ `z()`, and `z()` throws an exception caught by `x()`) must be recompiled so that the compiler can generate tables for the functions in question. This

can be done using the `-eh` switch (to generate tables for all functions in the file) or using the `generate_exceptions_tables` pragma (to generate tables for specific functions).

- Assembly functions that may have exceptions thrown through them need exceptions tables crafting manually.

-no-annotate Switch Extended

In addition to disabling compiler-generated annotations, the `-no-annotate` switch disables insertion of line-numbering comments into assembly files. For example, while the normal assembly output might contain a mixture of line-number comments and instructions:

```
// line “x.c”:2
r0 = 17;
// line 3
i12=dm(m7,i6);
```

when the `-no-annotate` switch is used, the assembly file will just contain instructions, like this:

```
r0 = 17;
i12=dm(m7,i6);
```

Loader Changes

The new ADSP-BF52x and ADSP-BF54x Blackfin processors use a loader file format incompatible with that of the ADSP-BF53x and ADSP-BF561 Blackfin processors. The new `.ldr` file format consists of a 16-byte block header for each boot block, extending the flag field from 16 bits to 32 bits, and adding a new 32-bit field for argument passing.

The loader utility supports both formats, depending on the processor specified. Ensure a correct target processor name appears on the command line used to generate a loader file, preventing a boot process from failing. Existing ADSP-BFBF533 and ADSP-BF561 projects can be migrated to the new processors by rebuilding the sources with the desired targets.

For more information, see the *VisualDSP++ 5.0 Loader and Utilities Manual* and online Help.

Changes to Existing VDK Projects

When porting existing VDK projects from VisualDSP++ 4.5 to VisualDSP++ 5.0, the following changes need to be taken into account.

- The VDK Status Window has been redesigned to display information on projects built without debugging information. As a result, the VDK Status window no longer shows kernel information for executables built with VisualDSP++ 4.0 or earlier.
- In VisualDSP++ 5.0, the `VDK_Public.h` and `VDK_Internals.h` include files no longer include `stdlib.h`. If your VDK application code uses definitions from `stdlib.h` but does not explicitly include the file, build errors will be encountered. To rectify the situation, add an include of `stdlib.h` in a source that uses definitions from `stdlib.h` (add `#include <stdlib.h>`).
- In VisualDSP++ 5.0, the `VDK_Public.h` and `VDK_Internals.h` include files no longer include `ccblkfn.h`. If your VDK application code uses definitions from `ccblkfn.h` and does not explicitly include it build errors will be encountered. To rectify the situation, add an include of `ccblkfn.h` in a source that uses definitions from `ccblkfn.h` (add `#include <ccblkfn.h>`).
- VDK now reserves the `SFT2I` and `SFT3I` interrupts for the reschedule interrupt because of difficulties found with the reschedule interrupt in SHARC processors. These interrupts cannot be used in any other manner. This change was introduced in the VisualDSP++ 4.5 June 2007 update.
- The run-time libraries use the standard namespace. Commensurate changes are required in source code for VDK projects. This change was documented in VisualDSP++ 4.5, but has not been available for new VDK projects until VisualDSP++ 5.0. Projects created with VisualDSP++ 4.5 should build as expected because the `-ignore-std` option is passed to the compiler.

- VDK now flags an additional value used as a timeout in the `PendSemaphore()`, `PendMessage()`, `PendEvent()`, and `PendDeviceFlag()` APIs as being invalid. Passing the value (`0|kNoTimeoutError`) as the timeout now results in a `kInvalidTimeout` error. In previous versions of VDK, the value was accepted silently; however, specifying that no error should be dispatched in the event of a timeout (also that the pend call should never timeout) was not useful. It also was ambiguous as to the expected result. The issuing of an error in this case now draws attention to this fact. This change was introduced in the VisualDSP++ 4.5 November 2006 update.
- For Blackfin processors, VDK's exception handler now passes exception `0x23` (data access CPLB protection violation) to the `cplb_mgr()` function, instead of passing the exception to `UserExceptionHandler`.

For porting projects created with versions older than VisualDSP++ 4.5 refer to the *VisualDSP++ 5.0 Kernel (VDK) User's Guide*.

Device Drivers and System Services

This section describes the major changes in the device drivers and system services between VisualDSP++ 4.5 and VisualDSP++ 5.0 releases.

Interrupt Manager

Enumeration values for the `ADI_INT_PERIPHERAL_ID` enumeration are changed slightly within VisualDSP++ 5.0. No changes are required within applications using the system services libary; however all applications and libraries should be recompiled to insure the proper enumeration values are used.

DMA Manager/Interrupt Manager

In VisualDSP++ 4.5, a single interrupt handler was used to service all DMA data transfer interrupts. When that handler was executed, the handler checked each open DMA channel to see if it was asserting and if so, processed the interrupt. Aside from inefficiencies with this approach, a situation could exist where a DMA interrupt was pending, but no channel appeared to be asserting. This caused the processor to appear hung (constantly looking for an interrupt to be serviced).

To illustrate this problem, take the case of two DMA channels, one operating at a higher interrupt priority than the other. Assume the lower priority DMA channel triggers and the lone DMA interrupt handler begins to execute, but before we exit the interrupt handler, the higher priority channel triggers. Within the interrupt handler, because it loops through all the channels, the handler would service both the higher and lower DMA channels then exit the handler. However, the higher priority interrupt is still latched so the interrupt handler would be immediately executed again. The handler would loop through all the channels and, because the higher priority channel has already been serviced, the handler wouldn't find any channels asserting so it would exit. Because the interrupt is still latched, the processor remains in an infinite loop.

In VisualDSP++ 5.0, this was changed in such a manner that a different handler is installed for each DMA channel. The basic functionality of each handler is the same as in VisualDSP++ 4.5 except it only looks at the one channel to which the handler is assigned; it does not loop through each channel. This not only corrects the infinite loop issue but also addresses the inefficiency in processing interrupts.

While functionally the same, with VisualDSP++ 5.0 the system now uses more interrupt handlers than were used in the past. As a result the user may need to supply additional memory to the `adi_int_Init()` function to accommodate the additional interrupt handlers. From a simplistic perspective, in VisualDSP++ 4.5, a system using two DMA channels would need to supply enough memory to the `adi_int_Init()` function for one interrupt handler but in VisualDSP++ 5.0, memory needs to be supplied for two interrupt handlers. However, this is worst case and assumes that both DMA channels are mapped to the same IVG level and at least one other interrupt handler also used that IVG level. If the two DMA channels are mapped to different IVG levels and they are the only interrupt handlers on those IVG levels, then no additional memory would be required when moving from VisualDSP++ 4.5 to VisualDSP++ 5.0. The actual amount of memory required depends on the number of DMA channels that are used, how they are mapped to IVG levels, and what other interrupts are mapped to the same IVG levels.

To recognize when additional memory to the `adi_int_Init()` function is required, look for the following return codes from the system services library and device drivers:

```
ADI_INT_RESULT_NO_MEMORY (0x50004)
ADI_DMA_RESULT_CANT_HOOK_INTERRUPT (0x30011)
ADI_PWR_RESULT_CANT_HOOK_SUPPLEMENTAL_INTERRUPT
ADI_DEV_RESULT_CANT_HOOK_INTERRUPT (0x40000011)
```

If any of these error codes are returned to the application from within the system services or device drivers, then additional memory should be supplied to the `adi_int_Init()` function.

DMA Manager

In VisualDSP++ 4.5, the memory copy functions `adi_dma_MemoryCopy()` and `adi_dma_MemoryCopy2D()` did not use the memory DMA interrupt to detect when memory DMA transfers completed, but rather a polling mechanism was used. Within VisualDSP++ 5.0, these functions now rely on the DMA interrupt of the destination DMA channel being asserted and serviced in order for the transfer to be recognized as complete. In asynchronous mode, this behavioral change is unlikely to be noticed, however in synchronous mode, where these functions do not return control until the transfer has completed, care must be taken to insure the destination channel's DMA interrupt can be serviced.

If the memory DMA interrupt is not serviced, then these DMA memory copy functions do not return back to the application, so the processor appears to be hung. By default, the memory DMA interrupt is fairly low priority, so if these functions are called synchronously from an interrupt priority level that is higher than the memory DMA interrupt priority level, then the memory DMA interrupt will never be serviced and the processor appears hung.

To help mitigate this issue, logic has been added into the debug version of the DMA manager such that when these functions are called synchronously, the interrupt priority level that the processor is currently running at is checked and compared to the memory DMA interrupt priority. If the memory DMA priority is equal to or lower than the currently executing priority level, an error code (`ADI_DMA_RESULT_INCOMPATIBLE_IVG_LEVEL`) is returned to the application. Note that this check is not performed in the release version of the system services library. If you use these functions, be sure to check the return value.

If this error code is returned:

- Ensure you do indeed wish to do a synchronous memory DMA transfer in the background of an interrupt level process. Often the simple `memcpy()` function of the C Run-Time Library will be more efficient than DMA.
- Or adjust your IVG mappings (via the `adi_int_SICSetIVG()` function) such that the memory DMA IVG is a higher priority (lower IVG number) than the IVG level from which the function is called.

Included with VisualDSP++ 5.0 are new functions that allow the memory DMA transfers to be queued to the service. These functions are all prefixed with `adi_dma_MemoryQueueXXX()`.

Semaphore Service

Necessarily, a semaphore service is now provided within the system services library. It is used primarily in support of the file system service. However, Analog Devices strongly discourages the service being used for anything outside the internal workings of the file system service.

USB Drivers

VisualDSP++ 5.0 contains a completely new implementation of the USB device drivers. These new drivers support the existing PLX-NET2272 device and the integrated USB controller for ADSP-BF54x processors.

These new drivers are consistent with the library naming conventions and locations as described in Chapter 1 of the *VisualDSP++ 5.0 Device Drivers and System Services Manual for Blackfin Processors*.

For example:

- `libusb5xx.dlb` - contains all core USB functionality, class drivers, etc.
- `libdrv54x.dlb` - contains the USB controller driver for ADSP-BF54x processors
- The `<install_path>\Blackfin\include\drivers\usb` subdirectories contain all USB include files
- The `<install_path>\Blackfin\lib\src\drivers\usb` subdirectories contain all USB source files

See "USB Drivers" on page 4-9 for information about migrating applications to these new USB drivers.

4 VISUALDSP++ 5.0 OBSOLETE OR REMOVED FEATURES

This chapter describes the features that have been deprecated or removed in VisualDSP++ 5.0. 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++ 4.5. Similarly, new projects created with VisualDSP++ 5.0 cannot be used by earlier versions of the tools.

This chapter contains lists of obsolete or removed features:

- "Discontinued Processor Support" on page 4-2
- "VisualDSP++ IDDE" on page 4-2
- "Compilers and Libraries" on page 4-2
- "VDK" on page 4-7
- "Definition Header Macros" on page 4-7

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

You may also want to visit the Software Tools Upgrades Web site to check if an update is available for your VisualDSP++ 5.0 installation:
`http://www.analog.com/processors/technicalSupport/toolsUpgrades.html`

Installing an update ensures that your software contains the latest processor support, example code, and bug fixes.

Discontinued Processor Support

VisualDSP++ 5.0 does not provide support for ADSP-218x and ADSP-219x processors. Therefore, refer to VisualDSP++ 3.5 documentation and the online Help if you need information on how to develop and run projects on ADSP-21xx processors. VisualDSP++ 3.5 continues to be available for ADSP-218x and ADSP-219x developers.

Please note that VisualDSP++ 5.0 does not provide support for ADSP-BF566 processors.

VisualDSP++ IDDE

- VisualDSP++ 5.0 does not support the Apex-ICE emulator.
- **Saving Register Contents.** Earlier versions of VisualDSP++ included a **Save Registers** command under the **Registers** menu in the IDDE. As of VisualDSP++ 5.0, this capability is accomplished via the core file support; refer to "Core File Support" on page 2-6 for details. Users who want the previous capability can run a script; refer to the "Saving Register Contents" topic in VisualDSP++ Help for an example VBScript that performs this task via the VisualDSP++ automation API.

Compilers and Libraries

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

Refer to Chapter 3, "VisualDSP++ 5.0 Major Changes" for more information about the changes to the C/C++ compilers and run-time libraries.

Removed Compiler Switches

The following compiler command-line switches have been removed from the VisualDSP++ 5.0 release.

Table 4-1. Obsolete Blackfin Compiler Switches

Switch	Description
`-jcs21+`	Deprecated, Same as `-jcs21`.
`-jump<constdata|data|code>`	Removed. Used to determine which section the compiler uses to store jump-tables. Use `-section` instead.
`-no-demangle`	Removed. Used to stop the compiler driver piping errors through the stand-alone demangler.
`-pedantic`	Removed. Used to warn about non-ANSI constructs.
`-pedantic-errors`	Removed. Used to produce error diagnostic for non-ANSI constructs.
`-progress-rep-gen-opt`	Removed. Replaced by`-progress-rep-opt`
`-progress-rep-mc-opt`	Removed. Replaced by`-progress-rep-opt`.
`-sat32`	Removed. Used to saturate all accumulations at 32 bits.
`-sat40`	Removed. Used to saturate all accumulations at 40 bits.

Although the `-jump<constdata|data|code>` compiler switches are deprecated, similar effects can be obtained using the `-section` compiler flag. For instance, the effect of the `-jump-data` flag can be achieved using the `-section SWITCH=data1`.

The `-no-demangle` switch has been removed as the behavior it was intended to suppress – filtering compiler errors through a “demangler” to give more readable C++ symbol names – was removed in an earlier release.

The `-pedantic` and `-pedantic-errors` switches have been deprecated as the meaning was ambiguous. The compiler would fault non-ANSI C common extensions and some Analog Devices’ specific extensions, but not all. Removing the options was judged the best solution in resolving the ambiguity.

The `-sat32` and `-sat40` switches have been removed. Now the compiler `cmac_fr16` and `cmsu_fr16` builtins always saturate at 32 bits (as per the `-sat32` switch). This is the same default behavior as previous versions of the compiler. For more information, see “Saturation and Optimization” on page 3-10.

Table 4-2. Obsolete SHARC Compiler Switches

Switch	Description
`-default-linkage-{asm|C|C++}`	Deprecated. Sets the default linkage type (`C`, `C++`, `asm`)
`-no-demangle`	Removed. Used to stop the compiler driver piping errors through the stand-alone demangler.
`-pedantic`	Removed. Used to warn about non-ANSI constructs.
`-pedantic-errors`	Removed. Used to produce error diagnostic for non-ANSI constructs.
`-switch-pm`	Removed Specified that the switch tables should be placed in the pm memory.

Table 4-2. Obsolete SHARC Compiler Switches (Cont'd)

Switch	Description
`-bss`	Removed Specified that the global data should be placed into a BSS-style section (called "bsz"), rather than into a normal global data section.
`-no-bss`	Removed Specified that the zero-initialized and non-zero-initialized data should be kept in the same data section, rather than separating zero-initialized data into a different BSS-style section

The `-no-demangle` switch has been removed as the behavior it was intended to suppress – filtering compiler errors through a "demangler" to give more readable C++ symbol names – was removed in an earlier release.

The `-pedantic` and `-pedantic-errors` switches have been deprecated as the meaning was ambiguous. The compiler would fault non-ANSI C common extensions and some Analog Devices' specific extensions, but not all. Removing the options was judged the best solution in resolving the ambiguity.

Although the `-switch-pm` compiler flag is deprecated, similar effects can be obtained using the `-section SWITCH=seg_pmco` compiler flag.

Table 4-3. Obsolete TigerSHARC Compiler Switches

Switch	Description
`-no-demangle`	Removed. Used to stop the compiler driver piping errors through the stand-alone demangler.
`-pedantic`	Removed. Used to warn about non-ANSI constructs.
`-pedantic-errors`	Removed. Used to produce error diagnostic for non-ANSI constructs.

The `-no-demangle` switch has been removed as the behavior it was intended to suppress – filtering compiler errors through a "demangler" to give more readable C++ symbol names – was removed in an earlier release.

The `-pedantic` and `-pedantic-errors` switches have been deprecated as the meaning was ambiguous. The compiler would fault non-ANSI C common extensions and some Analog Devices' specific extensions, but not all. Removing the options was judged the best solution in resolving the ambiguity.

SAVE_REGS Macro

In previous releases of VisualDSP++, the Blackfin compiler provided support for "trap" operations, wherein an application could request a service from some operating system by invoking a software exception: the application would indicate the required service by specific register values, and the exception handler would perform the service and return results by way of modifying registers before restoring the application context. VisualDSP++ supported this process by saving a fixed context during all interrupts and exceptions, and by providing a `SAVE_REGS()` macro that gave access to the saved context.

This mechanism is intended for applications that must transition from User Mode to Supervisor Mode to request services, which has not been supported in recent releases of VisualDSP++ (VisualDSP++ only supports Supervisor Mode). Therefore, the context-saving and its corresponding `SAVE_REGS()` macro have been discontinued.

As a result, interrupt handlers and hardware exception handlers in VisualDSP++5.0 are far more efficient. See also "Interrupt and Exception Handlers" on page 3-16.

VDK

VDK no longer supports `0.x` silicon revision of the ADSP-TS20x processors. In order to use VDK on these TigerSHARC processors, use silicon revision 1.0 or higher.

See "Changes to Existing VDK Projects" on page 3-21 for information on the kernel changes.

Definition Header Macros

This section describes changes to definition header macros.

Blackfin Def Header Change: DMA32

The macro definition for the `DMA32` bit in PPI registers was erroneously provided in the following single-core Blackfin processor headers in `<install_dir>\Blackfin\include`:

- `defBF532.h`
- `defBF534.h`
- `defBF539.h`
- `defBF52x_base.h`

The complete list of processors affected by these headers is:

- ADSP-BF531 / ADSP-BF532 / ADSP-BF533 / ADSP-BF538
- ADSP-BF534 / ADSP-BF536 / ADSP-BF537
- ADSP-BF539
- ADSP-BF522 / ADSP-BF525 / ADSP-BF527

Do not use the `DMA32` macro when building for these targets. The ability to pack DMA onto a 32-bit bus is an ADSP-BF561 feature, and it was an error to have `DMA32` in any of the single-core headers in past VisualDSP++ releases.

If used in assembly code that previously assembled, the build will now result in an assembler warning, as follows:

```
[Warning ea1092] "<filename>":line Symbol 'DMA32' is undefined.
```

If used in C code that was previously compiled, the build will now result in the following error:

```
"<filename>", line x: cc0020: error: identifier "DMA32" is
undefined.
```

In the past, this was a "do nothing" situation, but today any legacy ADSP-BF531/ADSPBF532/ADSP-BF533 code that had the `DMA32` macro included in its initialization code for the `PPI_CONTROL` register will now enable the alternate timing functionality (`ALT_TIMING` bit) built into new silicon. This will result in PPI timing changes in your application when using new silicon.

SHARC Def Header Change: FAR

The definition of bit 20 in the `SDCTL` register has been changed from "`FAR`" to "`FARF`" to avoid redefining `FAR` in the `zlib` library source.

This is a non-backwards compatible change.

The affected header files include:

- `213xx/include/def21369.h`
- `213xx/include/def21369.h`
- `213xx/include/def21375.h`

Replace any usage of `FAR` with `FARF` in your applications.

USB Drivers

VisualDSP++ 5.0 contains a completely new implementation of the USB device drivers; refer to "USB Drivers" on page 3-26. All existing example projects have been updated to use these new drivers.

If you are migrating applications to these new USB drivers, review the existing Blackfin examples and refer to `USB_Application_Porting_Guide.pdf` located in `<install_path>\Blackfin\docs\drivers\usb`.