VisualDSP++™ 3.0

Product Bulletin for Blackfin™ DSPs

First Revision, April 2002

Part Number
82-000410-09

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

ANALOG DEVICES

Copyright Information

Printed in the USA.

Disclaimer

Trademark and Service Mark Notice

Revision 1.0

CONTENTS

PREFACE

PREFACE

Thank you for purchasing VisualDSP++™, Analog Devices development software for digital signal processors (DSPs).

Purpose of This Document

This document briefly describes the new features and enhancements provided by VisualDSP++ Release 3.0 for Blackfin™ DSPs. It also describes the differences between Release 3.0 and Release 2.0.

For details refer to the manuals in VisualDSP++ 3.0 documentation set for Blackfin DSPs and to the VisualDSP++ online Help.

Intended Audience

This manual is primarily intended for customers who are upgrading from the previous release of VisualDSP++ and for DSP programmers who want an overview of the changes to VisualDSP++ in Release 3.0.

Manual Contents

This manual consists of:

- Chapter 1, “Introduction”

  Describes VisualDSP++, the system requirements for running Release 3.0, the processors supported in this release, and the benefits provided by this release

- Chapter 2, “New Features and Enhancements”

  Describes what is new in the VisualDSP++ Release 3.0 integrated development and debugging environment, simulator, assembler, compiler, linker, loader, and documentation. It also describes the new VisualDSP++ Component Software Engineering (VCSE) function and the changes to the VisualDSP++ kernel (VDK).

- Chapter 3, “Differences Between Releases”

  Describes the differences between Release 3.0 and Release 2.0 as they pertain to assembler and compiler commands, switches, operators, directives, pragmas, keywords, macros, and VDK

Technical or Customer Support

You can reach DSP Tools Support in the following ways.

- Visit the DSP Development Tools website at
  `http://www.analog.com/technology/dsp/developmentTools/index.html`
- Email questions to
  `dsptools.support@analog.com`
- Phone questions to **1-800-ANALOGD**
- Contact your ADI local sales office or authorized distributor
- Send questions by mail to
  ```
  Analog Devices, Inc.
  DSP Division
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA
  ```

Supported Processors

The name "*Blackfin*" refers to the family of Analog Devices 16-bit, fixed-point digital signal processors. VisualDSP++ currently supports the following Blackfin processors.

- ADSP-21532 DSP
- ADSP-21535 DSP

Product Information

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

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

MyAnalog.com

MyAnalog.com is a free feature of the Analog Devices website that allows customization of a webpage to display only the latest information on products you are interested in. You can also choose to receive weekly email notification containing updates to the webpages that meet your interests. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Registration:

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

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

DSP Product Information

For information on digital signal processors, visit our website at www.analog.com/dsp, which provides access to technical publications, datasheets, application notes, product overviews, and product announcements.

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

- Email questions or requests for information to
  `dsp.support@analog.com`
- Fax questions or requests for information to
  1-781-461-3010 (North America)
  089/76 903-557 (Europe)
- Access the Digital Signal Processing Division's FTP website at
  `ftp ftp.analog.com` or **ftp 137.71.23.21**
  `ftp://ftp.analog.com`

Related Documents

For information on product related development software, see the following publications:

VisualDSP++ 3.0 Getting Started Guide for Blackfin DSPs

VisualDSP++ 3.0 C/C++ Compiler and Library Manual for Blackfin DSPs

VisualDSP++ 3.0 C/C++ Assembler and Preprocessor Manual for Blackfin DSPs

VisualDSP++ 3.0 Linker and Utilities Manual for Blackfin DSPs

VisualDSP++ Kernel (VDK) User's Guide

VisualDSP++ Component Software Engineering User's Guide

Quick Installation Reference Card

Online Technical Documentation

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

A description of each documentation file type is as follows.

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

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by running the Tools installation.

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

From VisualDSP++

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

From Windows

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

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

Using Windows Explorer

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

Using the Windows Start Button

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

From the Web

To download the tools manuals, point your browser at `http://www.analog.com/technology/dsp/developmentTools/gen_purpose.html`.

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

Printed Manuals

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

VisualDSP++ Documentation Set

VisualDSP++ manuals may be purchased through Analog Devices Customer Service at **1-781-329-4700**; ask for a Customer Service representative. The manuals can be purchased only as a kit. For additional information, call **1-603-883-2430**.

If you do not have an account with Analog Devices, you will be referred to Analog Devices distributors. To get information on our distributors, log onto `http://www.analog.com/salesdir/continent.asp`.

Hardware Manuals

Hardware reference and instruction set reference manuals can be ordered through the Literature Center or downloaded from the Analog Devices website. The phone number is **1-800-ANALOGD** (**1-800-262-5643**). The manuals can be ordered by a title or by product number located on the back cover of each manual.

Datasheets

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

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

Contacting DSP Publications

Please send your comments and recommendation on how to improve our manuals and online Help. You can contact us by:

- Emailing `dsp.techpubs@analog.com`
- Filling in and returning the attached Reader's Comments Card found in our manuals

Notation Conventions

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

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

Example	Description
Close command (**File** menu)	Text in **bold** style indicates the location of an item within the VisualDSP++ environment's menu system. For example, the **Close** command appears on the **File** menu.
`{this | that}`	Alternative required items in syntax descriptions appear within curly brackets and separated by vertical bars; read the example as `this` or `that`.
`[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 ellipsis; 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.
	A note, providing information of special interest or identifying a related topic. In the online version of this book, the word **Note** appears instead of this symbol.
	A caution, providing information about critical design or programming issues that influence operation of a product. In the online version of this book, the word **Caution** appears instead of this symbol.

1 INTRODUCTION

This chapter describes VisualDSP++ and the requirements for running Release 3.0. It also lists the supported processors and some of the benefits provided by this release.

The information is organized as follows.

Product Description

VisualDSP++ 3.0 is the DSP tools software platform for developing applications for Analog Devices DSPs. The successor to VisualDSP++ 2.0, this release incorporates a number of new features and enhancements, described in this document.

System Requirements for Running Release 3.0

To run VisualDSP++ 3.0, your PC requires the following hardware and software.

- Windows 98/NT 4.0SP3/2000/ME/XP
- At least 100 MB of available hard drive space
- At least 32 MB of RAM
- CD-ROM drive
- Internet Explorer 4.01 or later

Processor Support

This release supports the following Blackfin processors.

- ADSP-21535
- ADSP-21532 (new)

Benefits Provided by Release 3.0

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

- **Support for new architectures**. VisualDSP++ 3.0 includes support for a new processor: the ADSP-21532.
- **Rapid application development**. The tight integration between the VisualDSP++ environment and the underlying tools enables you to develop applications rapidly, as you do not have to create all the required control code manually. Using automatic code generation and file templates, you can concentrate on algorithms and control flow rather than on implementation details. VisualDSP++ provides both the means to develop code that is easy to read and maintain, and the option to optimize manually.
- **Code reusability**. Programmers often begin a new project by writing infrastructure portions that transfer data between algorithms. The VisualDSP++ kernel (VDK) provides this control logic in a standard, portable, and reusable library. The kernel's design takes advantage of commonality in user applications to encourage code reuse.

  Each thread of execution is created from a user-defined template, either at boot time or dynamically by another thread. Multiple threads can be created from the same template, but the state associated with each created instance of the thread remains unique.

Each thread template represents a complete encapsulation of an algorithm that is unaware of other threads in the system unless it has a direct dependency.

The kernel also promotes good coding practice and organization by partitioning large applications into blocks or subsystems of functionality that are easy to maintain.

- **Flexibility and extensibility**. VisualDSP++ Component Software Engineering (VCSE) tools simplify the process of developing, documenting, and validating reusable software components. VisualDSP++ support includes the means to display information about available components and to incorporate them into a project. You use the VCSE Interface Definition Language (VIDL) to define interfaces and the components that implement them. VisualDSP++ provides a VIDL compiler that enables you to create and use components without becoming familiar with the detail of the model and mechanism involved. VisualDSP++ also provides wizards for creating interfaces and components.

  VCSE support:

  - Enables the creation of software algorithms as reusable components
  - Simplifies the use of software algorithms as components from other developers
  - Provides confidence that components do not interact unpredictably or have resource conflicts
  - Allows the use of components from assembler, C or C++ programs irrespective of the language that the component was implemented in
  - Encourages and simplifies the effective reuse of software

2 NEW FEATURES AND ENHANCEMENTS

This chapter describes the new features in Release 3.0 and the enhancements to the VisualDSP++ environment, tools, and documentation.

The information is organized as follows.

Integrated Development and Debugging Environment (IDDE)

The IDDE provides these new features and enhancements:

- External makefile execution
- New profiler interface
- New streams interface
- Image Viewer
- Background Telemetry Channel (BTC)
- Flash Programmer
- Editor enhancements

External Makefile Execution

The IDDE has been enhanced to support the loading and building of external gmake-compatible makefiles. This feature enables you to develop, maintain, and use your own external makefile without having to leave the IDDE. When you open a makefile, it is displayed in the **Project** window as a single file node. Double-clicking on the makefile icon in the **Project** window opens the makefile in an editor window. A syntax highlighting file is provided for easier makefile editing.

The IDDE does not consume, parse, or perform any type of syntax checking on the makefile and does not modify the loaded makefile in any way. While a makefile is opened, all project-related commands with the exception of **Build** and **Build All** are disabled. When a **Build** or **Build All** command is issued, the gmake engine is executed and all output is displayed on the **Build** page of the **Output** window.

If an error occurs, the error text, which includes the line number and name of the file, appears on the **Build** page of the **Output** window. The error text conforms to the syntax described in the section "Error Message Numbering and Help" on page 2-17. Double-clicking the text opens the specified file and places the cursor on the line that has the error.

New Profiler Interface

The legacy profiling tool supported by simulators has been merged with the new statistical/linear profiler.

The **Profile** option has been removed from the **Tools** menu and from the **View/Debug Windows** menu since the legacy profiler is no longer supported.

Linear Profiling

A new linear profiling interface is available for the ADSP-21532 simulator only. Emulators will continue to use the statistical profiling interface. The simulator uses linear profiling to pass profiling data (such as the count of memory reads and writes per PC address) that was available only in the legacy profiler. The memory reads and writes now appear in extra columns in the **Linear Profiling Results** window, as shown in Figure 2-1.

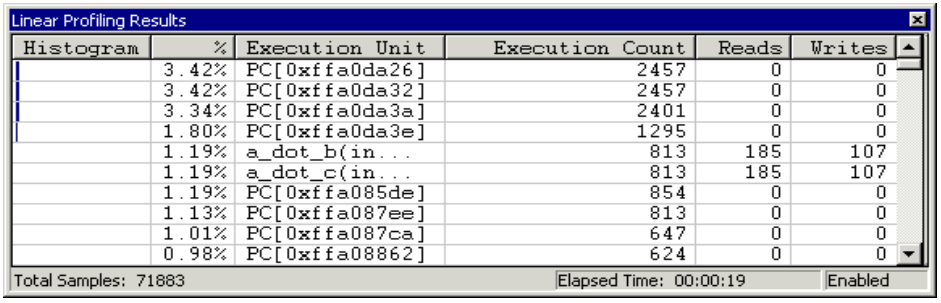

Figure 2-1. New Linear Profiling Results Window

The linear profiling interface enables the simulator to pass additional profiling data, such as cache hits and misses, that can be displayed in additional columns. To view more information at one time, right-click in the **Linear Profiling Results** window to display the **Profile Window Properties** dialog box (Figure 2-2).

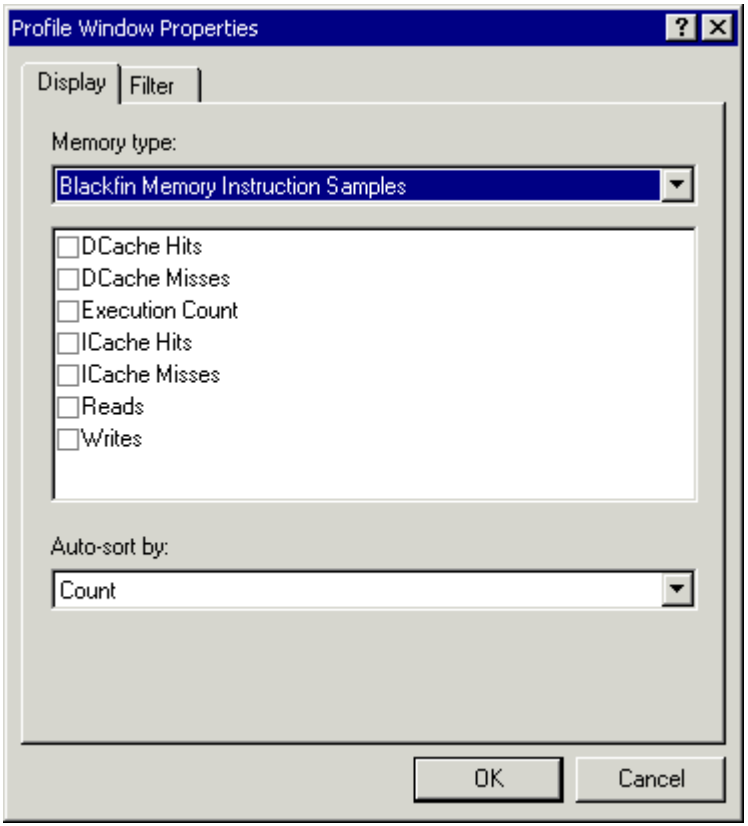

Figure 2-2. Profile Window Properties Dialog Box

Click the **Display** tab and select the profiling data items that you want to display.

Selecting the **Filter** tab enables you to choose one of the following items to profile.

- **Entire memory space**. The linear and statistical profilers have the same behavior, as in VisualDSP++ 2.0.
- **C/C++ functions**. You can choose functions to profile from a list of available functions. The profiler discards any collected PC samples that are not within the selected functions.
- **Memory ranges**. You can specify the start and end addresses of the memory ranges to profile. The profiler discards any collected PC samples that are not within the selected memory ranges. The **Profiling Results** window displays the start and end addresses in the **Execution Unit** column (see Figure 2-1 on page 2-3).

Most of the time, when building debug versions of the code, you profile without concern for non-debug code like the C runtime library initialization code. A check box provides the option of filtering out any PC samples that have no corresponding debug information. If you put a check mark in the box, the profiler discards any PC samples that have no debug information.

Statistical Profiling

The **Statistical Profiling Results** window was modified to include a horizontal scroll bar at the bottom of the window. The scroll bar is useful when the window is not big enough to display both the histogram and source panes.

New Streams Interface

The **Streams** dialog box was overhauled to make it easier to create a new stream and to allow current streams to be modified. The dialog box contains a list of current streams. You can add new streams by pressing the **Add** button to invoke a new dialog box that guides you through the setup procedure. Modify current streams by selecting a stream from a list of current streams and by pressing the **Edit** button. A new dialog box appears to guide you through the edit procedure.

The settings for the streams are automatically saved and restored to ensure that you do not have to set up the streams every time you start the IDDE. The settings are saved on a per-session basis.

Image Viewer

Within an IDDE plug-in window, you can display image and video data or DSP memory stored in a PC image file. Supported file types include `.BMP`, `.JPEG`, `.PPM`, and `.MPEG`.

Image Viewer includes these dialog boxes:

- The **Configuration** dialog box enables you to define the image source (memory or file) and image attributes. If the image is located in DSP memory, you must specify the image address, size, and format. The image is read from DSP memory, based on the parameters entered. If the image is located in a file, the image size is determined automatically. The image data is read from the file and is transferred to DSP memory, starting at the specified start address.

- The **Image Viewer** window renders the image. Scroll bars are enabled if the image is larger than the window. Buttons for zooming the image in and out are available. Each press of the zoom in (+) or out (–) button enlarges or decreases the size of the image by a factor of 2. As you move the mouse, the status bar displays:
    - DSP address where the selected pixel is located
    - Red, green, and blue (RGB) pixel values
    - Column and row in pixel coordinates
- The **Gamma Correction** dialog box provides slider controls for adjusting the red, green, and blue (RGB) pixel values of the image. You can link the sliders to adjust the brightness of the image. Clicking **OK** writes the adjusted image data to DSP memory.
- The **Export** dialog box enables you to send the image to a printer or copy the image to the clipboard or a file.
- The **Property** dialog box displays image address and attributes.
- The **Play Video** dialog box enables you to play a video clip of MPEG data stored in DSP memory.

Background Telemetry Channel (BTC)

A set of BTC dedicated registers accessible to both JTAG and the DSP core enables the debugger to exchange data with the target while the DSP is executing code. Impact on the real-time operations of the target is minimal.

Flash Programmer

The VisualDSP++ Flash Programmer provides a convenient, generic interface to numerous DSPs and flash memory devices.

This utility simplifies the process of changing data values on a flash device and modifying its memory. You no longer have to remove the flash memory from the board, use a separate Flash Programmer, and then replace the flash.

Editor Enhancements

New features added to VisualDSP++ editor windows include:

- Right-click menu command that toggles the display of line numbers
- Brace matching based on the position of the text cursor
- Auto-positioning of closing curly brackets for proper alignment with preceding opening brace
- Right-click command to open header files from the file name in an `#include` statement
- Support for dragging and dropping text into an open **Expressions** window

Simulator

The simulator provides these new features and enhancements:

- Cycle-accurate simulator
- ADSP-21532 cache visualization

Cycle Accurate Simulator

The simulator for the ADSP-21532 processor accurately models the core and memory.

ADSP-21532 Cache Visualization

With this new simulator feature, cache statistics are collected in the Blackfin simulator and are associated with both the PC/Source Line and the Cache Line/Set. The following statistics are collected.

- Total cache accesses
- Cache hits and cache misses, with some or all of the following types
  - Compulsory miss (fill)
  - Conflict miss (replace)
  - Capacity miss

The above statistics can be correlated to named arrays or other data objects and to sections of code.

Assembler

VisualDSP++ Release 3.0 provides new assembler features.

New Command Line Switches

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

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

```
-stallcheck=cond
```

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

Multicycle behavior exists when an instruction, sometimes only under certain circumstances, takes more than one cycle to complete. You cannot avoid this cycle loss without removing the instruction that caused it. Such cases (in addition to conditional stalls) are reported when you use the following switch:

```
-stallcheck=all
```

Other New Features

- Conditional assembly (see "Conditional Assembly" on page 3-5)
- Structure import (see "Structure Import" on page 3-6)

- Support for importing C header files and access to structures defined in C header files

Compiler

The compiler provides these new features and enhancements:

- Optimization pragmas
- Circular buffer intrinsics
- Blackfin-specific enhancements
- Preprocessing support for `.IDL` files

Optimization Pragmas

When you are ready to compile your code, you can change the optimization options on a per-function basis by using:

```
#pragma optimize_for_space

#pragma optimize_for_speed

#pragma optimize_off
```

These pragmas are subject to the following restrictions:

- They apply only to back-end optimizations. Front-end optimizations always work with the optimizations specified on the command line or by the IDDE.
- They are valid only before a function definition.
- They are ignored if optimization is not requested in the IDDE or on the command line.
- They are ignored if Interprocedural Analysis (IPA) is enabled.

Circular Buffer Intrinsics

The circular buffer intrinsics are designed to support C++ classes that encapsulate the functionality of the hardware circular buffer registers. Two models of the C++ circular buffer class are supported: circular pointer and circular index. Each uses a compiler intrinsic.

```
ptrdiff_t __builtin_circindex(ptrdiff_t i,
                              ptrdiff_t m,
                              size_t l);


void * __builtin_circptr(const void *i,
                         ptrdiff_t m,
                         const void *b,
                         size_t l);
```

Both intrinsics have the same implementation.

An optimization pass converts `__builtin_circindex` to `__builtin_circptr`.

The compiler allocates circular buffer registers to chains and cycles of `__builtin_circptr` operations with the same base and length.

Registers are allocated only within a function. The compiler has no mechanism for reserving circular buffer registers across multiple functions.

In addition, the compiler uses circular buffers where the modulus operator is used in an address expression and the operation is equivalent to a conditional subtract. A compiler option `-circbuf` forces this interpretation.

Blackfin Specific Compiler Enhancements

This release includes the following Blackfin specific compiler enhancements.

- Improved intrinsics for complex math
- Improved vectorization
- Pragma pad for specifying minimum alignments of structure members
- `SYSREG` support (including interrupt enable/disable/raise)
- Support for larger hardware loops, beyond the size that `LSETUP` supports
- Support for very large stack frames (greater than 131K is not recommended)
- Generation of mac-and-extract instructions
- Better register allocation after loop unrolling
- Ability to use a single pointer for induction variables with constant offset
- Ability to configure `L1` as cache
- Ability to make `SMALL` configuration the default
- Library implementation of `SYSREG` support
- Upgrade to C++ `fract` class
- Improved C++ `complex` class

VIDL Compiler

The VCSE Interface Definition Language (VIDL) compiler supports the preprocessing of Interface Definition Language (`.IDL`) files.

Linker

The linker provides these new features and enhancements:

- Expert Linker
- Data elimination
- New default LDF behavior

Expert Linker

The Expert Linker provides a GUI-based means of supplying the link commands currently supplied to the linker in a Linker Description File (`.LDF`). The tool also provides graphical and hypertext representations of text output for cross-reference, the linker map, and `ELFDUMP`.

Expert Linker enables you to visualize the commands in an `.LDF` file, make changes by manipulating graphical representations of the `.LDF` file, and generate an `.LDF` file. Any `.LDF` file that the Expert Linker supports can be read, modified, and written out to ensure support from both the linker and the Expert Linker. Key features include:

- Ability to display the memory layout (memory maps of supported parts) in the **Expert Linker** window and change the memory description. Use the **Memory Map** pane's right-click menu to access various commands.

- Ability to drag and drop (add, move) object files, sections, or both
- Representation of object files after "pre-linking." A determination of the sections appears in the link after you eliminate unused functions and objects.
- Zoom functions that provide greater detail of executables and objects. You can zoom down to symbols and lengths.

Data Elimination

The style of assembly code generated by the compiler was modified to provide the linker with greater scope for eliminating unneeded data and code. This modification does not affect the efficiency or size of the generated code.

New Default LDF Behavior

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

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

Loader

The new Blackfin loader (`elfloader.exe`) generates a boot-loadable file for ADSP-2153x DSPs by processing executable files. To generate a loadable file, the loader processes data from a boot kernel file (`.DXE`), the linker's executable file (`.DXE`), and in some cases overlay files (`.OVL`). The ADSP-21532 DSPs use one on-chip ROM bootstrap kernel for automatic booting from an external memory device. The ADSP-21535 DSPs are supported by two booting kernels. Boot loading through the boot kernel is currently supported only on ADSP-21535 DSPs.

Once you have fully debugged your program, use the loader to generate a set of boot-loadable files for your target system. The loader produces one output file (`.LDR`) or two output files (boot kernel file `.LDR` and application code file `.KLN`), depending on your loader setup selections. Loading the loader output into a simulator session in the VisualDSP++ debugger enables you to simulate the boot process and the boot-loaded application.

Documentation

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

Online Help

New VisualDSP++ online Help features include error message numbering, a unified index for all manuals in the documentation set, and the ability to search for keywords across the documentation set.

Error Message Numbering and Help

Tools that do batch processing can produce a list of errors and warnings when returning a result. Error reporting has been enhanced as follows.

- Every error is uniquely identified with a number that is consistent from release to release. Consistent numbering enables you to become familiar with the meaning of an error message.
- Tool identifiers prepended to error numbers identify what tool was involved when you report an error to customer support. The tool prefixes are defined in the following examples.

```
ar0001 - archiver
cc0001 - compiler error
ea0001 - elf assembler error
el0001 - Expert Linker
ld0001 - loader
li0001 - linker error
pp0001 - assembler/linker preprocessor error
vc0001 - VIDL compiler
```

- Every error is documented. Detailed information about an error includes the severity level, what condition caused the error and what you can do to fix it, examples, and related information.

- Error messages are output in a format that can be parsed by the IDDE. Where applicable, they include file names and line number information. Error messages that do not refer to a particular source location do not include file names and line number information.
- Every tool uses the same hierarchy of error message severity, described as follows from the highest to the lowest level of severity.

fatal	Identifies an error so severe that further processing of the input is suspended. The tool reports a failure.
error	Identifies a problem that causes the tool to report a failure. Further processing of the input might be allowed to continue to enable additional problems to be reported.
warning	Identifies a situation that does not prevent the tool from processing the input, but may indicate potential problems
remark	Provides information of possible interest

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

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

Unified Index and Search

Index entries from all the Release 3.0 manuals have been merged into one unified index, which you can access by clicking the **Index** tab in the online Help. You can use the **Search** function in the online Help to search for keywords and instructions across all Release 3.0 publications. The Help window lists each occurrence and its location (manual).

Online Manuals

All manuals in the Release 3.0 documentation set have been converted to HTML Help and merged to facilitate searches in the online Help. Each book is still available in PDF format.

You can access VisualDSP++ online documentation from the VisualDSP++ **Help** menu or from the Windows **Start** button as follows.

- To access VisualDSP++ online Help, click the **Start** button and choose **Programs**, **VisualDSP**, and **VisualDSP++ Help**.
- To access the `.PDF` files, click the **Start** button and choose **Programs**, **VisualDSP**, and **Documentation for Printing**, and the name of the manual.

VisualDSP++ Component Software Engineering (VCSE)

VCSE consists of a combination of tools and guidelines that simplify the process of developing components and help to document and validate such components. These tools and guidelines:

- Enable applications to incorporate and use software algorithm components from other developers easily and with confidence
- Ensure that components from multiple vendors will not interact with each other in unpredictable ways or have resource clashes
- Allow components to be developed in assembler, C, or C++ and be used from applications developed in any of these languages
- Allow components to be reused easily
- Allow comparison of algorithms that offer the same functionality
- Encourage third party developers to provide the implementation of algorithms as easily used components

Interface Definition Language and Compiler

VCSE supports an Interface Definition Language (IDL) and compiler that enable developers to specify and then create and use components without having to become familiar with the detail of the model and its mechanisms. The VIDL compiler processes the specification of the interfaces that a component supports and generates the framework code needed to implement the component. The developer of the component can then concentrate on providing the implementation of the methods that the component is to provide. In addition, the VIDL compiler can generate a simple test harness to help in testing the component.

Integration with VisualDSP++

Integration of VCSE with VisualDSP++ simplifies the process of creating components and of incorporating and utilizing components from a variety of developers.

VCSE menu options accessed from the **Tools** menu enable you to:

- Use a wizard to create an new interface specification
- Use a wizard to create a VCSE component and a project to build the component
- Create a project to support the creation of a VCSE component
- Install and a view components on the local system
- Download and install new or updated components from the ADI web site
- Add an installed component to a project

VisualDSP++ Kernel (VDK)

The following VDK features were added in Release 3.0.

- Support for messaging between threads
- A block based memory pool manager
- Dynamic creation of semaphores and device flags
- Provision for counting semaphores
- Support for measuring thread stack usage
- Thread local storage
- Sizeable reductions in the costs of many common API functions and a significant reduction in interrupt latency
- **Kernel** page support for the new functionality

You can use VDK messages to:

- Communicate information between threads
- Control access to a shared resource
- Signal the occurrence of an event and communicate information about the event
- Allow two threads to synchronize

The maximum number of supported messages can be configured separately for each project, and the messages owned by each thread can be viewed in the **VDK Status** window.

The memory pool manager provides support for deterministic and highly efficient memory allocation for one or more memory pools. Each memory pool contains memory blocks of a single size, but multiple pools can be defined, each with a different block size.

Device driver support has been redesigned and device drivers are now constructed as input/output objects.

Enhanced VDK Status Window

The **VDK Status** window's tree view was extended to display values for:

- Additional thread items (last error, stack, and message queue)
- Semaphores
- Events
- Event words
- Device flags
- Memory pools

Enhanced State History Window

The **VDK State History** window displays:

- Log events, including user-defined events
- User-configurable colors, patterns, and fonts

Each thread state can have a color and pattern. Each event can have a color but not a pattern. Font selection applies to all text in the display. Selecting **Use as default** loads the custom settings each time the **VDK State History** window is created.

Object Protection

DSP tools now support the encryption and decryption of object files. This feature enables algorithm providers to distribute objects and libraries under license protection.

3 DIFFERENCES BETWEEN RELEASES

This chapter describes the differences between Release 3.0 and Release 2.0.

The information is organized as follows.

Assembler Differences

This section describes the differences between the Release 3.0 and Release 2.0 assembler.

Commands

This release includes one new preprocessor command, described in the following table.

Command	Description
`#pragma`	Implementation-specific command that could modify the preprocessor behavior Takes any sequence of tokens

Command Line Switches

Release 3.0 includes these new assembler command line switches.

Switch	Description
`-flags-compiler -opt1 [,-opt2...]`	Passes each comma-separated option to the compiler Used when compiling `.C` header files
`-proc processorID`	Specifies a processor for which the assembler should produce suitable code
`-wnumber`	Suppresses any report of the specified warning

The following assembler command line switches are no longer supported.

Switch	Description
`-BLACKfin`	Specifies a processor type
`-r`	Removes preprocessor information from a listing file

Operators

The following table describes the new preprocessor operators in this release.

Operator	Description
?	Generates unique labels for repeated macro expansions
. . .	Specifies a variable length argument list

Directives

The following assembler directives are new in this release.

Directive	Description
.ENDIF	Ends all conditional assembly blocks
.ELIF	Used within the .IF and .ENDIF conditional directives
.ELSE	Used within the .IF and .ENDIF conditional directives
.IF	Begins all conditional assembly blocks
.IMPORT	Provides the assembler with the structure layout (C struct) information
.LIST	Starts listing of source lines
.LIST_DATA	Starts listing of data opcodes
.LIST_DATFILE	Starts listing of data initialization files
.LIST_DEFTAB	Sets the default tab width for listings
.LIST_LOCTAB	Sets the local tab width for listings
.LIST_WRAPDATA	Starts wrapping opcodes that do not fit listing column
.NOLIST	Stops listing of source lines
.NOLIST_DATA	Stops listing of data opcodes
.NOLIST_DATFILE	Stops listing of data initialization files

(Cont'd)

Directive	Description
.NOLIST_WRAPDATA	Stops wrapping opcodes that do not fit listing column
.OFFSETOF	Calculates the offset of a specified member from the beginning of its parent data structure. For Blackfin DSPs, units are in bytes.
.SIZEOF	Returns the amount of storage associated with an imported C struct or data member
.STRUCT	Defines and initializes data objects based on C type defs from .IMPORT C header files
.WEAK	Creates a weak definition or reference

Conditional Assembly

This release includes the following conditional assembler directives for evaluating assembly-time constants using relational expressions.

- .IF
- .ELIF
- .ELSE
- .ENDIF

These directives are reserved keywords.

Structure Import

You can use the compiler to process structure definitions provided by a C header file. This release includes the following directives and macro that support this feature.

- Associated directives: `.STRUCT`, `.IMPORT`, `.OFFSETOF`, `.SIZEOF`
- Macro: `__LANGUAGE.C`

Macros

The predefined macros in Release 2.0 have been renamed in Release 3.0. They are now called *feature* macros.

The following table lists the feature macros defined by the assembler when it invokes the preprocessor.

Macro Definition	Notes
`-D__ADSPBLACKFIN__=1`	Always present
`-D_LANGUAGE_ASM =1`	Always present
`-D__ADSP21535__=1`	Present when running `easmblkfn -proc` ADSP-21535 (for ADSP-21535 DSP)
`-D__ADSP21532__=1`	Present when running `easmblkfn -proc` ADSP-21532 (for ADSP-21532 DSP)
`-D_LANGUAGE_C`	Used for C compiler calls to specify `.IMPORT` headers Replaces `_LANGUAGE_ASM`

Compiler Differences

Compiler Common Switches

This release provides the following new compiler common switches.

Switch	Description
`-21532`	Generates code for the ADSP-21532 DSP
`-circbuf`	Causes the compiler to treat array references as circular buffer references, and to generate code accordingly
`-debug-types`	Supports building a `*.h` file directly and writing a complete set of debugging information for the header file
`-proc processorID`	Specifies a processor for which the compiler should produce suitable code

Pragmas

This section describes new pragmas provided by Release 3.0. Note that some of these pragmas were provided in patches for Release 2.0.

Data Alignment Pragmas

Release 3.0 includes the following data alignment pragmas.

Pragma	Description
`pack (alignopt)`	Applies to `struct` definitions that follow, until the default alignment is restored by omitting `alignopt`. This pragma reduces the default alignment of the `struct` to be aligned. Use this pragma to `struct` definitions.
`pad (alignopt)`	Applies to `struct` definitions that follow, until the default alignment is restored by omitting `alignopt`. This pragma reduces the default alignment of the `struct` to be aligned and also increases the alignment of fields that default to an alignment less than `alignopt`. Use this pragma to `struct` definitions.

Interrupt Handler Pragma

Release 3.0 includes the following interrupt handler pragma.

Pragma	Description
`interrupt_reentrant`	Used with the `interrupt` pragma to specify that the function's context-saving prologue should also arrange for interrupts to be re-enabled for the duration of the function's execution

Optimization Pragmas

Release 3.0 includes the following optimization pragmas.

Pragma	Description
`optimize_off`	Turns off the optimizer if it was enabled
`optimize_for_space` `optimize_for_speed`	Turn the optimizer back on if it was disabled, or switch focus between reducing code size and increasing performance if the optimizer was already enabled

Linking Pragma

Release 3.0 includes the linking pragma described below.

Pragma	Description
`retain_name`	Indicates the external function or variable declaration that follows the pragma is not removed even though interprocedural analysis sees that it is not used. Use this pragma if you have C functions that are called only from assembler routines, such as the startup code sequence invoked before `main()`.

VDK Differences

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

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

  While the underlying changes to device drivers have a minimal affect on the use of a device driver, the implementation of the device driver has changed significantly.

  Hence, device drivers created under VisualDSP++ 2.0 are incompatible with the device driver model in VisualDSP++ 3.0. Additionally, device flags are no longer associated with an individual device driver, but are instead created as global objects. The VDK User's Guide contains a section describing how to convert device drivers created under VisualDSP++ 2.0 for use in projects built with VisualDSP++ 3.0.

- In VisualDSP++ 2.0 all semaphores were binary semaphores created at boot time. In VisualDSP++ 3.0 all semaphores are counting semaphores that can be dynamically created.