

VISUALDSP++™ 3.0

Getting Started Guide for ADSP-21xx DSPs

Revision 2.0, July 2002

Part Number
82-000400-07

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



Copyright Information

©1996–2002 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

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

Trademark and Service Mark Notice

The Analog Devices logo, VisualDSP, the VisualDSP logo, SHARC, the SHARC logo, TigerSHARC, the TigerSHARC logo, and EZ-ICE are registered trademarks of Analog Devices, Inc.

VisualDSP++, the VisualDSP++ logo, CROSSCORE, the CROSSCORE logo, Blackfin, the Blackfin logo, EZ-KIT Lite, Apex-ICE, and Summit-ICE are trademarks of Analog Devices, Inc.

All other brand and product names are trademarks or service marks of their respective owners.

CONTENTS

PREFACE

Purpose of This Manual	vii
Intended Audience	vii
Manual Contents	viii
What's New in This Manual	viii
Technical or Customer Support	ix
Supported Processors	ix
Product Information	x
MyAnalog.com	x
DSP Product Information	xi
Related Documents	xii
Online Technical Documentation	xii
From VisualDSP++	xiii
From Windows	xiii
From the Web	xiv
Printed Manuals	xiv
VisualDSP++ Documentation Set	xiv
Hardware Manuals	xv
Datasheets	xv

CONTENTS

Contacting DSP Publications	xv
Notation Conventions	xvi

FEATURES AND TOOLS

VisualDSP++ Features	1-1
New VisualDSP++ Features in Release 3.0	1-5
Code Development Tools	1-9
VisualDSP++ Help System	1-11
Invoking Online Help	1-11
Viewing Context-Sensitive Help	1-12
Viewing Menu, Toolbar, or Window Help	1-13
Viewing Dialog Box Button or Field Help	1-13
Viewing Window Help	1-14
Using Help Window Navigation Buttons	1-14
Copying Example Code from Help	1-15
Printing Help	1-16
Bookmarking Frequently Used Help Topics	1-16
Placing a Bookmark at a Topic	1-17
Opening a Bookmarked Topic	1-17
Navigating in Online Help	1-17
Using the Search Features	1-19
Help System Search Rules	1-19
Rules for Full-Text Searches	1-19
Rules for Advanced Searches	1-20
Full-Text Searches	1-20

Advanced Search Techniques	1-22
Using Wildcard Expressions	1-22
Using Boolean Operators	1-23
Using Nested Expressions	1-24
Viewing Online Manuals	1-24
Default Software Installation	1-24
How to Print Online Manuals	1-25

TUTORIAL

Overview	2-1
Exercise One: Building and Running a C Program	2-3
Step 1: Start VisualDSP++ and Open a Project	2-4
Step 2: Build the dotprodc Project	2-7
Step 3: Run the Program	2-10
Step 4: Run dotprodc	2-15
Exercise Two: Modifying a C Program to Call an Assembly Routine	2-16
Step 1: Create a New Project	2-16
Step 2: Add Source Files to dot_product_asm	2-20
Step 3: Create a Linker Description File for the Project	2-21
Step 4: Modify the Project Source Files	2-25
Step 5: Use the Expert Linker to modify dot_prod_asm.ldf	2-28
Step 6: Rebuild and Run dot_product_asm	2-32
Exercise Three: Plotting Data	2-33
Step 1: Load the FIR Program	2-33

CONTENTS

Step 2: Open a Plot Window	2-35
Step 3: Run the FIR Program and View the Data	2-40
Exercise Four: Linear Profiling	2-45
Step 1: Load the FIR Program	2-45
Step 2: Open the Profiling Window	2-46
Step 3: Collect and Examine the Linear Profile Data	2-48
Exercise Five: Multiprocessor Debugging	2-50
Step 1: Create a Multiprocessor Simulator Session	2-51
Step 2: Changing Focus and Pinning Windows	2-56
Step 3: Loading Programs in a Multiprocessor Session	2-61
Step 4: Stepping in a Multiprocessor Session	2-65
Step 5: Configuring and Using Multiprocessor Groups	2-66
Adding Processors to the Default Group and Issuing MP Reset	2-67
Creating and Configuring New Multiprocessor Groups	2-69
Activating New Multiprocessor Groups	2-70
Exercise Six: Installing and Using a VCSE Component	2-71
Step 1: Start VisualDSP++ and Open the Project	2-71
Step 2: Install the EXAMPLES::CULawc Component on Your System	2-73
Step 3: Add the Component to Your Project	2-76
Step 4: Build and Run the Program	2-77

INDEX

PREFACE

Thank you for purchasing Analog Devices development software for digital signal processors (DSPs).

Purpose of This Manual

The *VisualDSP++ 3.0 Getting Started Guide for ADSP-21xx DSPs* provides a step-by-step tutorial that highlights many of the VisualDSP++™ features. By completing this tutorial you will become familiar with the VisualDSP++ environment, and you will learn how to use these features in your own DSP development projects.

Intended Audience

This manual is intended for DSP programmers who are familiar with Analog Devices DSPs. The manual assumes that the audience has a working knowledge of Analog Devices DSP architecture and the DSP instruction set.

DSP programmers who are unfamiliar with Analog Devices DSPs can use this manual, but they should supplement it with other texts (such as Hardware Reference and Instruction Set Reference manuals that describe your target architecture).

Manual Contents

This guide contains the following chapters:

- Chapter 1, “Features and Tools”

Provides an overview of VisualDSP++ features and code development tools

- Chapter 2, “Tutorial”

Provides step-by-step instructions for creating sessions, and for building and debugging projects by using examples of C/C++ and assembly sources

The tutorial is organized to follow the steps that you take in developing a typical programming project. Before you begin actual programming, you should be familiar with the architecture of your particular processor and the other software development tools.

What’s New in This Manual

The *VisualDSP++ 3.0 Getting Started Guide for ADSP-21xx DSPs* includes procedures for:

- Using the Expert Linker option to create and modify an .LDF file
- Installing and using a VCSE component

Technical or Customer Support

You can reach DSP Tools Support in the following ways.

- Visit the DSP Development Tools website at
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

VisualDSP++ currently supports the following ADSP-21xx processors.

- ADSP-2181
- ADSP-2183
- ADSP-2184/84L/84N
- ADSP-2185/85L/85M/85N
- ADSP-2186/86L/86M/86N
- ADSP-2187L/87N

Product Information

- ADSP-2188L/88N
- ADSP-2189M/89N
- ADSP-2191
- ADSP-2192-12
- ADSP-2195
- ADSP-2196
- ADSP-21990
- ADSP-21991
- ADSP-21992
- AD90747

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](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 User's Guide for ADSP-21xx DSPs

VisualDSP++ 3.0 C Compiler and Library Manual for ADSP-218x DSPs

VisualDSP++ 3.0 C/C++ Compiler and Library Manual for ADSP-219x DSPs

VisualDSP++ 3.0 Assembler and Preprocessor Manual for ADSP-21xx DSPs

VisualDSP++ 3.0 Linker and Utilities Manual for ADSP-218x and ADSP-219x DSPs

VisualDSP++ 3.0 Product Bulletin

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

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by rerunning 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 using any shortcuts you may have constructed, you can open VisualDSP++ online Help or the supplementary documentation from Windows Explorer.

Product Information

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.

In 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.

From the Web

To download the tools manuals, point your browser at 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 <i>this</i> or <i>that</i> .
[this that]	Optional items in syntax descriptions appear within brackets and separated by vertical bars; read the example as an optional <i>this</i> or <i>that</i> .
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipsis; read the example as an optional comma-separated list of <i>this</i> .
.SECTION	Commands, directives, keywords, and feature names are in text with letter gothic font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
	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 FEATURES AND TOOLS

This chapter contains the following topics.

- [“VisualDSP++ Features” on page 1-1](#)
- [“New VisualDSP++ Features in Release 3.0” on page 1-5](#)
- [“Code Development Tools” on page 1-9](#)
- [“VisualDSP++ Help System” on page 1-11](#)

VisualDSP++ Features

VisualDSP++ provides the following features.

- **Extensive editing capabilities.** Create and modify source files by using multiple language syntax highlighting, drag-and-drop, bookmarks, and other standard editing operations. View files generated by the *code development* tools.
- **Flexible project management.** Specify a project definition that identifies the files, dependencies, and tools that you will use to build projects. Create this project definition once or modify it to meet changing development needs.

VisualDSP++ Features

- **Easy access to code development tools.** Analog Devices provides the following code development tools: C/C++ compiler, VIDL compiler, assembler, linker, splitter, and loader. Specify options for these tools by using dialog boxes instead of complicated command line scripts. Options that control how the tools process inputs and generate outputs have a one-to-one correspondence to command line switches. Define options for a single file or for an entire project. Define these options once or modify them as necessary.
- **Flexible project build options.** Control builds at the file or project level. VisualDSP++ enables you to build files or projects selectively, update project dependencies, or incrementally build only the files that have changed since the previous build. View the status of your project build in progress. If the build reports an error, double-click on the file name in the error message to open that source file. Then correct the error, rebuild the file or project, and start a debug session.
- **VisualDSP++ Kernel (VDK) Support.** Add VDK support to a project to structure and scale application development. The **Kernel** tab page of the Project window enables you to manipulate kernel objects such as events, event bits, priorities, semaphores, and thread types.
- **Flexible workspace management.** Create multiple workspaces and quickly switch between them. Assigning a different project to each workspace enables you to build and debug multiple projects in a single session.
- **Easy movement between debug and build activities.** You start the debug session and move freely between editing, build, and debug activities.

Figure 1-1 shows the Integrated Development and Debugging Environment.

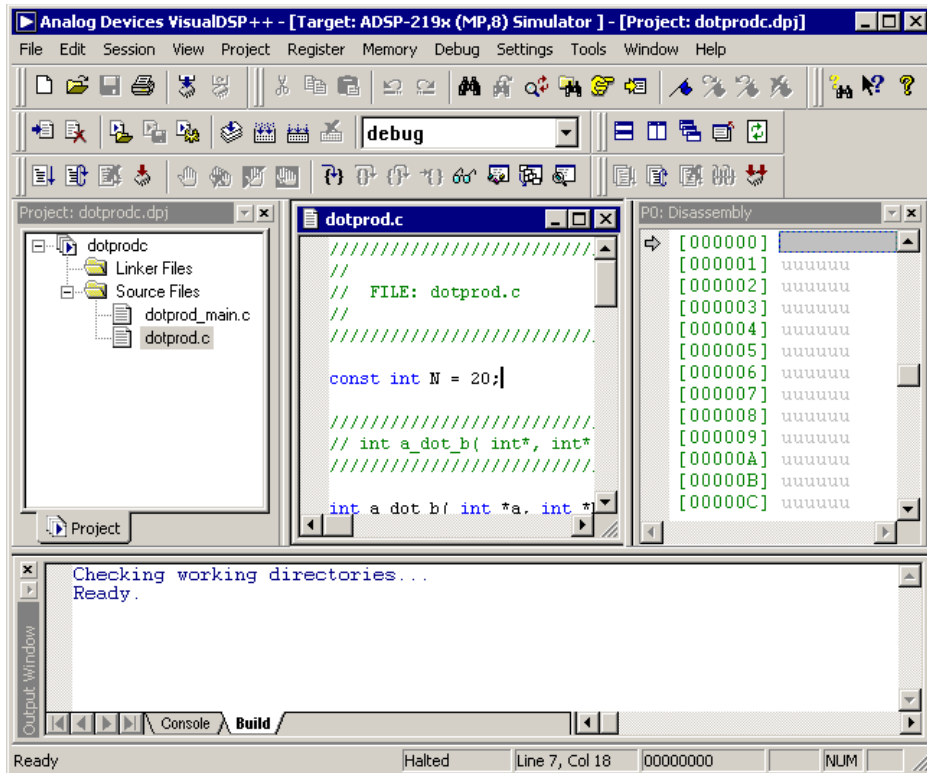


Figure 1-1. The VisualDSP++ IDDE

VisualDSP++ Features

VisualDSP++ reduces your debugging time by providing these key features.

- **Easy-to-use debugging activities.** Debug with one common, easy-to-use interface for all processor simulators and emulators, or hardware evaluation and development boards. Switch easily between these targets.
- **Multiple language support.** Debug programs written in C, C++ (ADSP-219x DSPs only), or assembly, and view your program in machine code. For programs written in C/C++, you can view the source in C/C++ or mixed C/C++ and assembly, and display the values of local variables or evaluate expressions (global and local) based on the current context.
- **Effective debug control.** Set breakpoints on symbols and addresses and then step through the program's execution to find problems in coding logic. Set watchpoints (conditional breakpoints) on registers, stacks, and memory locations to identify when they are accessed.
- **Tools for improving performance.** Use the trace, profile, and linear and statistical profiles to identify bottlenecks in your DSP application and to identify program optimization needs. Use plotting to view data arrays graphically. Generate interrupts, outputs, and inputs to simulate real-world application conditions.
- **Multiprocessor debugging** (ADSP-219x DSPs only). You can easily manage and debug any number of processors from the same debug session. Fully synchronous multiprocessor operations such as step, run, and halt allow cycle-accurate debugging of complex systems. You can arrange processors into logical groups for better control of a system's behavior.

New VisualDSP++ Features in Release 3.0

The following features are new in Release 3.0.

- **Expert Linker.** This graphical tool makes it easier for you to produce a Linker Description File (.LDF). Display and modify your DSP's memory map simply by dragging and dropping object sections into the map. After you link, run, and profile your project, the memory map is color-coded to show which objects are taking up the most execution time. Drag those object sections from slower external memory to faster internal memory to improve performance. You can see how much space is allocated for your program's heap and stack and reduce their size if they are using too much memory.
- **New Profiler.** A new linear profiler (ADSP-219x simulator only) replaces the legacy profiler, which is now merged with the statistical profiler (emulator only).

A new menu option (**New Profile**) under **Tools** and **Linear Profiling** opens the **Linear Profiling Results** window. Selecting **Properties** from the window's right-click menu enables you to specify which part of the program to profile and what profiling data to collect for viewing in the profiling window. You can also specify the profiling data to sort, how far back to trace history, and a memory range filter (entire memory, C/C++ functions, or memory range).

New VisualDSP++ Features in Release 3.0

- **Support for VisualDSP++ Component Software Engineering (VCSE).** 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.
- **Cache Visualization Tool.** This tool enables you to analyze the cache performance of a DSP application and to optimize the application's use of the cache. From the new **Cache Viewer** window, select one of four available views: configuration (information about the cache received by the API function), details (cache line information and cache event), history (events received for a cache line), and histogram (bar chart showing the cumulative events for each cache set).
- **Editor Enhancements.** New editor window features include a right-click command that toggles the view 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, a right-click command that opens header files from the filename in an `#include` statement, and support for dragging and dropping text into an open **Expressions** window.
- **Streams Interface Enhancements.** A new **Streams** dialog box, accessed from the **Settings** menu, enables you to add and modify streams. Settings for the streams are automatically saved and restored, so you do not have to set up the streams every time you start VisualDSP++.

A new check box, **Rewind at Start**, provides loop back support by allowing you to use one file to continue testing. A new **Processor** field enables you to set up streams between processors in a multi-processor session. New interfaces are supported by the targets for compiled simulation, and new Tcl commands are available for creating and deleting streams.

- **Plot Enhancements.** Scroll bars, slider controls, and zoom buttons added to the **Plot** window improve usability. Other new features include options for displaying plot statistics to the right of the plot and for playing audio data through a PC sound card.
- **Image Viewer.** The new **Image Viewer** plugin window enables you to view image and video data (BMP, JPEG, PPM, or MPEG) from DSP memory or from a file on your PC. You can adjust the gamma attributes of image, copy an image, save an image, print an image, and export an image.
- **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.
- **VDK Project Enhancements.** A memory pool manager facilitates an efficient, deterministic memory allocation scheme by organizing the memory into a set of memory pools. Release 3.0 provides messaging between threads, dynamic (runtime) creation of semaphores and device flags, and support for measuring thread stack usage. Device drivers are now part of the I/O interface.

New VisualDSP++ Features in Release 3.0

The **VDK Status** window now displays values for additional thread items (last error, stack, and message queue), semaphores, events, event bits, device flags, and memory pools.

The **VDK State History** window now displays new log events, including user-defined events, and user-configurable colors, patterns, and fonts. It also provides options for data filtering and searching.

- **External Makefile Support.** You can perform common command line operations such as view, edit, and save on a makefile without leaving VisualDSP++. You can open a makefile (.MAK or .MK extension) as a text file or a project file. Double-clicking the makefile in the **Project** window opens it as a text file for viewing and editing in an editor window. If you open the makefile as a project, VisualDSP++ looks for a project file with the same name in the makefile folder. If it finds the project, it adds the makefile to the project. Otherwise, it prompts you to create a new project and automatically adds the makefile to the new project.

Right-clicking a makefile in the **Project** window opens the makefile's **File Options** property page, where you specify the command for building the project. Use the active makefile with the gmake command to build projects. Errors from invoking the gmake command are displayed in the **Output** window. Double-clicking an error message positions the cursor on the line with the error in an editor window. An active makefile in a project disables the building of internal projects in VisualDSP++. Only one makefile can be active when you build a project.

- **Simulator Enhancements.** The simulator for the ADSP-21xx processor accurately models the core and memory.

Code Development Tools

The ADSP-21xx code development tools include:

- C/C++ compiler
- VIDL compiler
- Runtime library with over 100 math, DSP, and C runtime library routines
- Assembler
- Linker
- Loader
- Simulator
- Emulator (must be purchased separately from VisualDSP++)
- Splitter

These tools support the ADSP-21xx Family of 8-bit and 16-bit DSPs, and enable you to develop applications that take full advantage of the ADSP-21xx architecture.

For the ADSP-219x Family of DSPs, VisualDSP++ includes a linker that supports multiprocessing, shared memory, and memory overlays.

Code Development Tools

The code development tools provide the following key features.

- **Easy-to-program C, C++, and assembly languages.** Program in C/C++, assembly, or mix C/C++ and assembly in one source. The assembly language is based on an algebraic syntax that is easy to learn, program, and debug.
- **Flexible system definition.** Define multiple types of executables for a single type of DSP in one Linker Description File (.LDF). Specify input files, including objects, libraries, shared memory files, overlay files, and executables.
- **Support for overlays, multiprocessors, and shared memory executables** (ADSP-219x DSPs only). The linker places code and resolves symbols in multiprocessor memory space for use by multiprocessor systems. The loader enables you to configure multiprocessors with less code and faster boot time. Create host, link port, and PROM boot images.

Software and hardware tool kits include context-sensitive Help and manuals in PDF format.

For details about assembly syntax, refer to the *VisualDSP++ 3.0 Assembler and Preprocessor Manual for ADSP-21xx DSPs*.

VisualDSP++ Help System

The Help system is integrated into VisualDSP++ and is designed to help you obtain information quickly. You can use the Help system's table of contents, index, full-text search function, bookmark function, and extensive hyperlinks to jump to topics.

Figure 1-2 shows the **Contents**, **Index**, **Search**, and **Favorites** tabs in the VisualDSP++ Help window.

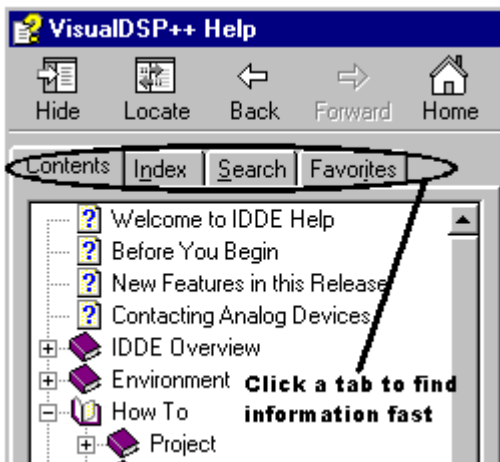


Figure 1-2. VisualDSP++ Help Window Tabs

Invoking Online Help

You can invoke online Help from VisualDSP++ or from the Windows **Start** button. You can also access Help manually via Windows Explorer.

To access online Help from VisualDSP++, choose **Contents**, **Search**, or **Index** from the **Help** menu.

VisualDSP++ Help System

To access online Help from the Windows **Start** button, click the **Start** button and choose **Programs, VisualDSP, and VisualDSP Documentation**.

The Help function is programmed to look for the Help system in the VisualDSP++ Help folder.

Note that when VisualDSP++ is installed with default values, the Help files reside in the Help folder.

If you receive an error message after invoking Help, the Help system:

- May not have been loaded onto your PC
- May have been deleted
- May reside in a directory other than the default directory

To locate the help (.CHM) files manually, use the Windows **Search** function as follows.

1. Record the Help file (.CHM) named in the error message.
2. From the Windows **Start** button, choose **Search** and **For Files or Folders**. Enter the name of the .CHM file from step 1.
3. After locating the file, launch it manually by clicking the file name from the **Search Results** window or from Windows Explorer.

Viewing Context-Sensitive Help

You can view context-sensitive Help (help pertinent to your current activity) for various items in VisualDSP++.

VisualDSP++'s context-sensitive Help is linked to toolbar buttons, menu commands, windows, and dialog box items.

Viewing Menu, Toolbar, or Window Help

1. Click the toolbar's Help button  or press **Shift+F1**.

The mouse pointer becomes a Help pointer .

2. Move the Help pointer over a menu command, toolbar button, or window.
3. Click the mouse to open the Help window. A description of the object appears in the right panel.

Viewing Dialog Box Button or Field Help

Perform one of these actions:

- Select a field or button in a dialog box and press **F1** or **Shift+F1**.
- Click the Question-Mark button  in the top-right corner of the dialog box.

The mouse pointer becomes a Help pointer .

Move the Help pointer over a dialog box control (button or field) and click the mouse. A description of the object appears in a yellow pop-up window.

- Position the mouse pointer over a label or control (button or field) in a dialog box and right-click.

A **What's This** button  appears. Move the mouse pointer over the **What's This** button and click.



“What's This” Help is not configured for all items.

Viewing Window Help

1. Click the window to make it active.
2. Press the F1 key to open the Help window.

A description of the window appears in the right panel.

Using Help Window Navigation Buttons

You can move through the Help system and view Help topics by using the Help window's navigational aids, as shown in [Figure 1-3](#).

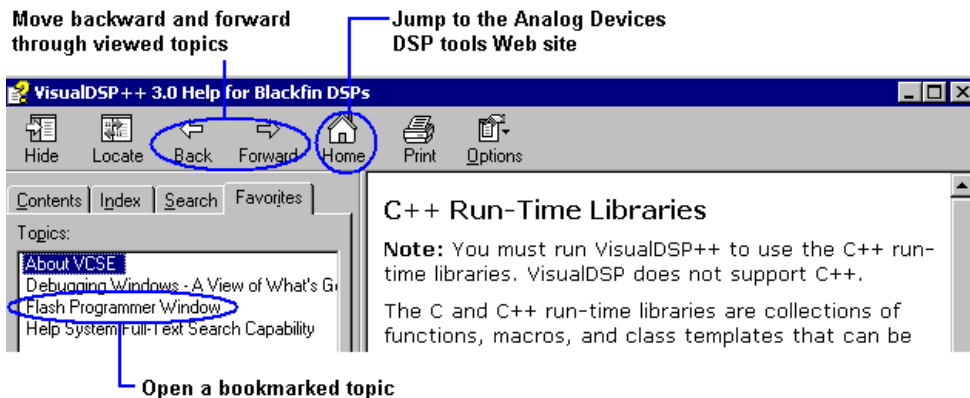


Figure 1-3. Help Window's Navigational Aids

Other standard Microsoft HTML Help buttons are described in [Table 1-1](#).

Table 1-1. Standard Microsoft HTML Help Buttons

Button	Purpose
 Hide	Hides the Help window's left pane. This button narrows the Help window.
 Show	Displays the Help window's left pane. This button restores a full view after you click Hide .
 Locate	Highlights the name of the current topic in the Contents tab page (left pane). After you jump around the Help system, this button shows the current topic's relation to other topics.

Copying Example Code from Help

You can copy code from the Help system and then paste it into your application. Be aware that the copied text may carry unwanted control codes. For example, if you copy a hyphen with a parameter, the actual code of the copied hyphen may be an ASCII 0x96 instead of an ASCII 0x2D. The hyphen may look OK, but it will cause an error when the command runs.

Printing Help

You can print a specific Help topic, or you can print multiple Help topics (an entire section of online Help).

Table 1-2. How to Print Help Topics

To print	Do this
Current topic	Right-click within the help topic and choose Print .
Selected topic	On the Contents tab: Right-click the topic  and choose Print .
Entire section of Help	On the Contents tab: Right-click a book icon  or  and choose Print . Then choose Print the selected heading and all subtopics .

Tip: From the Help window's **Contents** tab, click  , located at the top of the window.

Bookmarking Frequently Used Help Topics

You can bookmark a topic in online Help just like you might bookmark a page in a book. This feature is also called setting up favorite places.

Note: Each time you bookmark a Microsoft HTML Help topic, a record is recorded in the file, `HH.DAT`. This file not only records VisualDSP++ Help bookmarks, but also the bookmarks you place in other application Help systems that use `.CHM` files.

Once you have placed a bookmark onto a topic, you can view a list of bookmarked topics and quickly open one.

Placing a Bookmark at a Topic

1. Display the topic.
2. On the left side of the Help window, click the **Favorites** tab.
3. Click **Add**.

The Help system adds the topic and displays it in the alphabetized list.

You can remove a bookmark by selecting the name and clicking **Remove**.

Opening a Bookmarked Topic

1. On the left side of the Help window, click the **Favorites** tab.
2. Perform one of these actions:
 - Double-click the topic.
 - Select the topic and click **Display**.

Navigating in Online Help

To move around in the Help system, you can click the following.

- **A hyperlink within text.** The text is underlined and displayed in a color that is different from the regular black text.
- **A topic listed under a See Also heading.** The text is underlined and displayed in a color that is different from the regular black text.
- **A mini button or its associated text.** The button is a small gray square and the underlined text is in a different color.

- A topic name on the Contents tab (Figure 1-4)

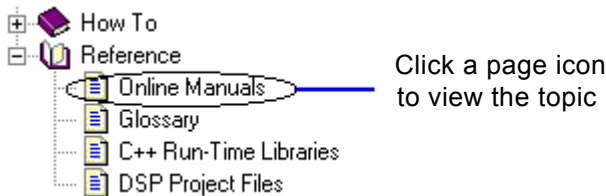


Figure 1-4. Contents Tab – Online Manuals Topic

- An index entry on the Index page (Figure 1-5)

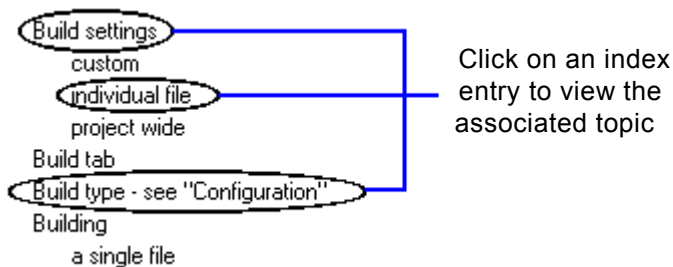


Figure 1-5. Index Entries on the Index Page

- A topic name on the Search page. The bottom portion of the Search page displays the located topics (hits) that include your search string.

Using the Search Features

VisualDSP++ Help provides both full-text and advanced search capabilities to help you find information.

Help System Search Rules

Different rules apply for each type of search.

Rules for Full-Text Searches

Observe these rules when formulating queries:

- Searches are not case-sensitive.

You can type your search in uppercase or lowercase characters. You can search for any combination of letters (a–z) and numbers (0–9).
- Searches ignore punctuation marks such as the period, colon, semi-colon, comma, and hyphen.
- Group the elements of your search by using double quotes or parentheses to set apart each element.
- You cannot search for quotation marks.

Note that if you are searching for a file name with an extension, group the entire string in double quotes, (“filename.ext”). Otherwise, the period breaks the file name into two separate terms. The default operation between terms is AND, which creates the logical equivalent to filename AND ext.

Rules for Advanced Searches

These rules apply to advanced searches:

- Expressions in parentheses are evaluated before the rest of the query.
- If a query does not contain a nested expression, it is evaluated from left to right. For example, “folder NOT file OR project” finds topics containing the word “folder” without the word “file,” or topics containing the word “project.” The expression “folder NOT (file OR project)”, however, finds topics containing the word “folder” without either of the words “file” or “project.”
- You cannot nest expressions deeper than five levels.

Full-Text Searches

The full-text search capability enables you to locate every occurrence of a text string within the Help system. You specify a particular word or phrase, and the search function finds only the topics that contain that word or phrase.

You can search previous results, match similar words, and search through the topic titles only.

A basic search consists of the word or phrase that you want to locate. You can use similar word matches, a previous results list, or topic titles to further define your search.

You can run an advanced search, which uses Boolean operators and wild-card expressions to further narrow the search criteria. [Figure 1-6 on page 1-21](#) shows an example of a Boolean search for “new AND plot”.

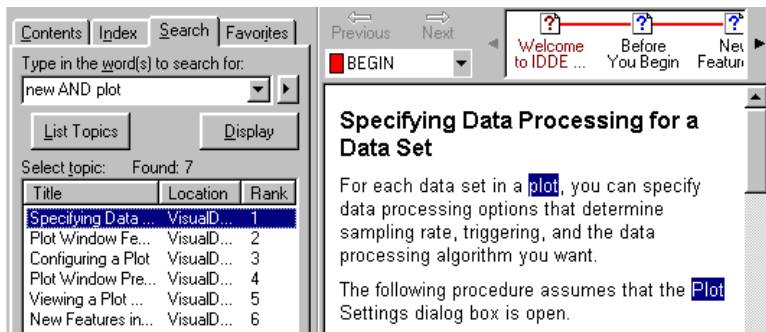


Figure 1-6. Boolean Search for “new AND plot”

To find information with full-text search:

1. Click the Help viewer’s **Search** tab.
2. In **Type in the word(s) to search for**, type the word or phrase you want to find.
3. Select **Search previous results** to narrow your search.
4. Select **Match similar words** to find words that are similar to the search string.
5. Select **Search titles only** to search only the topic titles.
6. Click the **Options** button  at the top of the Help Viewer window to highlight all instances of search terms found in topic files. Then choose **Search Highlight On**.
7. Click **List Topics**, select the topic you want, and then click **Display**.

Note that you can sort the topic list by clicking the **Title**, **Location**, or **Rank** column heading.

Advanced Search Techniques

You can use the following search techniques to narrow your searches for more precise results.

- Wildcard expressions
- Boolean operators
- Nested expressions

Using Wildcard Expressions

Wildcard expressions enable you to search for one or more characters by using a question mark or asterisk. [Table 1-3](#) describes the results of these different kinds of searches.

Table 1-3. How to Use Wildcard Expressions to Define a Search

To find	Example	Results
A single word	project	Locates topics that contain the word “project.” Other grammatical variations, such as “projects” are located.
A phrase	“project window” (note the quotation characters) project window	Locates topics that contain the literal phrase “project window” and all its grammatical variations. Without the quotation characters, the query is equivalent to specifying “project AND window,” which finds topics containing both of the individual words, instead of the phrase.
Wildcard expressions	link* -or- .C??	Locates topics that contain the terms “linker,” “linking,” “links,” and so on. The asterisk cannot be the only character in the term. Locates topics that contain the terms “.CPP” or “.CXX.” The question mark cannot be the only character in the term.

Using Boolean Operators

Use the Boolean AND, OR, NOT, and NEAR operators to precisely define your search by creating a relationship between search terms.

Insert a Boolean operator by typing the operator (AND, OR, NEAR, or NOT) or by clicking the arrow button.

Note that if you do not specify an operator, AND is used. For example, the query “call stack” is equivalent to “call AND stack.”

[Table 1-4](#) describes the results of using Boolean operators to define a search.

Table 1-4. How to Use Boolean Operators to Define a Search

To find	Example	Results
Both terms in the same topic	new AND plot	Locates topics that contain both the words “new” and “plot”
Either term in a topic	new OR plot	Locates topics that contain either the word “new” or the word “plot” or both
The first term without the second term	new NOT plot	Locates topics that contain the word “new”, but not the word “plot”
Both terms in the same topic, close together	new NEAR plot	Locates topics that contain the word “new” within eight words of the word “plot”

You cannot use the |, &, or ! characters as Boolean operators. You must use OR, AND, or NOT.

Using Nested Expressions

Use nested expressions to create complex searches for information. For example, new AND ((plot OR waterfall) NEAR window) finds topics containing the word “new” along with the words “plot” and “window” close together, or topics containing “new” along with the words “waterfall” and “window” close together.

Viewing Online Manuals

VisualDSP++ includes three types of user documentation, described in [Table 1-5](#).

Table 1-5. Types of User Documentation

Files	Purpose
.CHM	These Microsoft HTML Help (.CHM) files comprise the VisualDSP++ Help system. The .CHM files are located in the VisualDSP\Help directory. They require Internet Explorer 4.0 (or higher) or the installation of a component that provides a .CHM file viewer.
.PDF	These Adobe Portable Document Format (.PDF) files enable you to print DSP Tools documentation. The .PDF files are located in the VisualDSP\Docs directory, and one .PDF file exists for each manual. The .PDF files require a PDF file reader, such as Adobe Acrobat Reader 4.0 (or higher).
.HTM or .HTML	These files describe the Dinkum abridged C++ library and FlexLM network license manager software. HTML files require a browser, such as Microsoft Explorer.

Default Software Installation

By default, the VisualDSP++ software installation procedure places the complete set of user documentation in the VisualDSP\Docs directory. If you did not install the user documentation during software installation, you can install the manuals later from the VisualDSP++ CD-ROM.

How to Print Online Manuals

You can print documentation that is in PDF format.

To print a manual or portion of a manual:

1. From the Windows **Start** button, choose **VisualDSP** and **Documentation for Printing**.
2. Choose the name of the manual.

Adobe Acrobat Reader opens and displays the book.

3. Choose **File** and **Print** to specify the pages that you want to print and other print options.

2 TUTORIAL

This chapter contains the following topics.

- [“Overview” on page 2-1](#)
- [“Exercise One: Building and Running a C Program” on page 2-3](#)
- [“Exercise Two: Modifying a C Program to Call an Assembly Routine” on page 2-16](#)
- [“Exercise Three: Plotting Data” on page 2-33](#)
- [“Exercise Four: Linear Profiling” on page 2-45](#)
- [“Exercise Five: Multiprocessor Debugging” on page 2-50](#)
- [“Exercise Six: Installing and Using a VCSE Component” on page 2-71](#)

Overview

This tutorial demonstrates key features and capabilities of the VisualDSP++ Integrated Development and Debugging Environment (IDDE). The exercises use sample programs written in C, C++, and assembly for ADSP-21xx DSPs. The ADSP-219x simulator is used for all exercises.

You can use different ADSP-21xx processors with only minor changes to the Linker Definition Files (.LDFs) included with each project.

Overview

VisualDSP++ includes basic Linker Definition Files for each processor type in the `ldf` folder. The default installation path is:

```
Analog Devices\VisualDSP\21xx\ldf folder
```

The source files for these exercises are installed during the VisualDSP++ software installation.

The tutorial contains six exercises.

- In **Exercise One**, you will start up VisualDSP++, build a project containing C source code, and profile the performance of a C function.
- In **Exercise Two**, you will create a new project, use Expert Linker to create a Linker Description File for the project, modify sources to call an assembly routine, use Expert Linker to modify the `.LDF` file, rebuild the project, and profile the performance of the assembly language routine.
- In **Exercise Three**, you will plot the various waveforms produced by a Finite Impulse Response (FIR) algorithm.
- In **Exercise Four**, you will use linear profiling to examine the efficiency of the FIR algorithm used in Exercise Three. Using the collected linear profile data, you will pinpoint the most time-consuming areas of the algorithm, which are likely to require hand tuning in the assembly language.
- In **Exercise Five**, you will explore the multiprocessor debugging capabilities of VisualDSP++ (for ADSP-219x DSPs only), including synchronous control of multiple targets, window pinning, and the use of multiprocessor groups to control complex systems.
- In **Exercise Six**, you will install a VCSE component on your system and add the component to the project. Then you will build and run the program with the component.

Tip: Become familiar with the VisualDSP++ toolbar buttons, shown in [Figure 2-1](#). They are shortcuts for menu commands such as **File**, **Open**. Toolbar buttons and menu commands that are not available for the task that you are performing are disabled and displayed in gray.

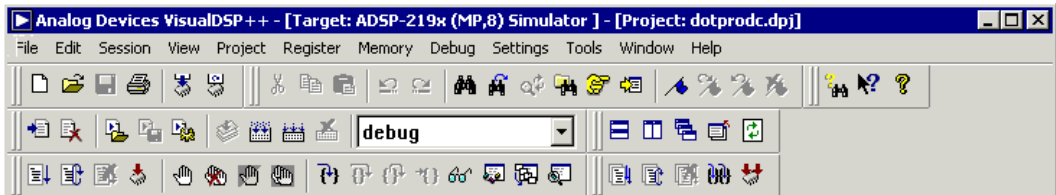


Figure 2-1. VisualDSP++ Toolbar Buttons

Exercise One: Building and Running a C Program

In this exercise, you will:

- Start up the VisualDSP++ environment
- Open and build an existing project
- Examine windows and dialog boxes
- Run the program

Exercise One: Building and Running a C Program

The sources for this exercise are in the `dot_product_c` folder. The default installation path is:

```
Program Files\Analog Devices\VisualDSP\219x\Examples\  
Tutorial\dot_product_c
```

Step 1: Start VisualDSP++ and Open a Project

To start VisualDSP++ and open a project:

1. Click the Windows **Start** button and select **Programs, VisualDSP, and VisualDSP++ Environment**.

If you are running VisualDSP++ for the first time, the New Session dialog box ([Figure 2-6 on page 2-11](#)) opens to enable you to set up a session. Select the values shown in [Table 2-1 on page 2-11](#). Then click **OK**. The VisualDSP++ main window appears.

If you have already run VisualDSP++ and the **Reload last project at startup** option is selected in the **Project Options** dialog box, VisualDSP++ opens the last project that you worked on.

To close this project, choose **Close** from the **Project** menu, and then click **No** when prompted to save the project. Since you have made no changes to the project, you do not have to save it.

2. From the **Project** menu, choose **Open**.

VisualDSP++ displays the **Open Project** dialog box.

3. In the **Look in** box, open the `Program Files\Analog Devices` folder and double-click the following subfolders in succession.

```
VisualDSP\219x\Examples\Tutorial\dot_product_c
```



This path is based on the default installation.

4. Double-click the `dotprodc` project (`.dpj`) file.

VisualDSP++ loads the project in the **Project** window, as shown in [Figure 2-2](#).

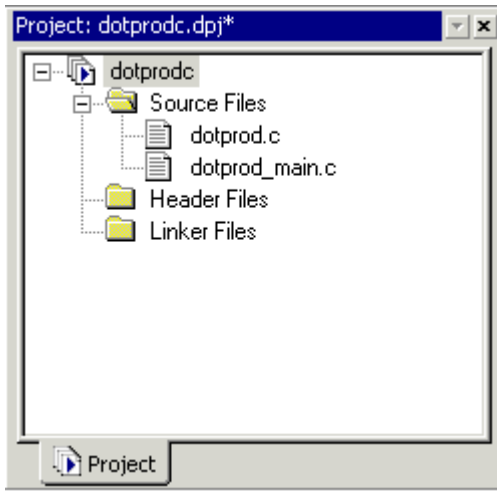


Figure 2-2. Project Loaded in the Project Window

The environment displays messages in the **Output** window as it processes the project settings and file dependencies.

i The first time that you open projects installed from the software kit, VisualDSP++ may detect that files, folders, or both have moved. If you receive a “Project has been moved” message, click **OK** to continue.

The `dotprodc` project comprises two C language source files, `dotprod.c` and `dotprod_main.c`, which define the arrays and calculate their dot products.

Exercise One: Building and Running a C Program

- From the **Settings** menu, choose **Preferences** to open the **Preferences** dialog box, shown in [Figure 2-3](#).

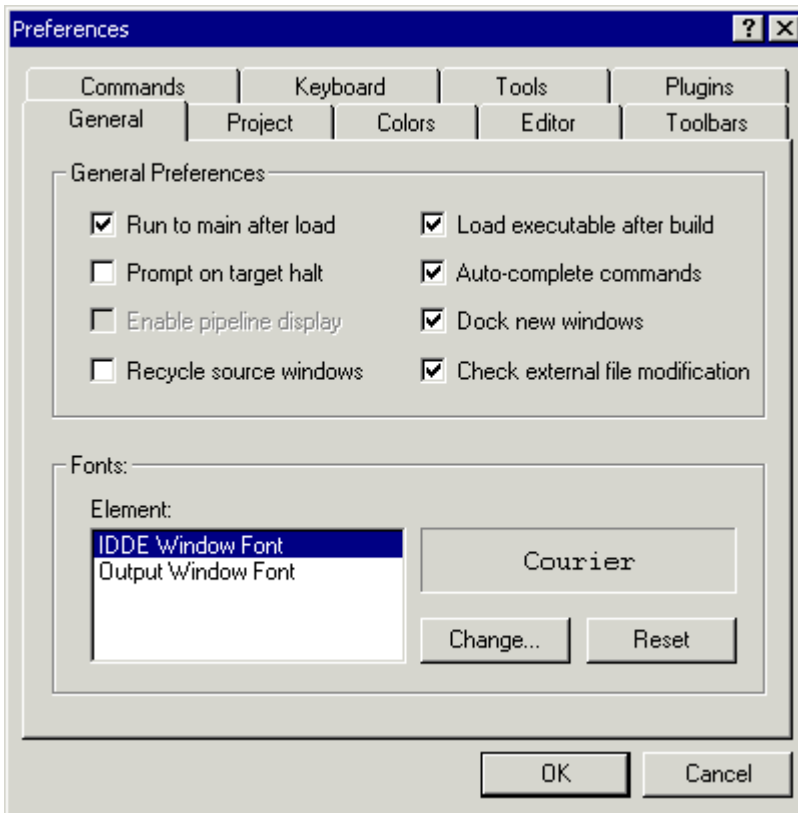


Figure 2-3. Preferences Dialog Box

- On the **General** page, under **General Preferences**, make sure that the following options are selected.
 - Run to main after load**
 - Load executable after build**

7. Click **OK** to close the **Preferences** dialog box.

The VisualDSP++ main window appears. You are now ready to build the project.

Step 2: Build the dotprodc Project

To build the `dotprodc` project:

1. From the **Project** menu, choose **Build Project**.

VisualDSP++ first checks and updates the project dependencies and then builds the project by using the project source files.

As the build progresses, the **Output** window displays status messages (error and informational) from the tools. For example, when a tool detects invalid syntax or a missing reference, the tool reports the error in the **Output** window.

If you double-click the file name in the error message, VisualDSP++ opens the source file in an editor window. You can then edit the source to correct the error, rebuild, and launch the debug session. If the project build is up-to-date (the files, dependencies, and options have not changed since the last project build), no build is performed unless you run the **Rebuild All** command. Instead, you see the message “Project is up to date.” If the build has no errors, a message reports “Build completed successfully.”

Exercise One: Building and Running a C Program

In this example ([Figure 2-4](#)) notice that the compiler detects an undefined identifier and issues the following error message in the **Output** window.

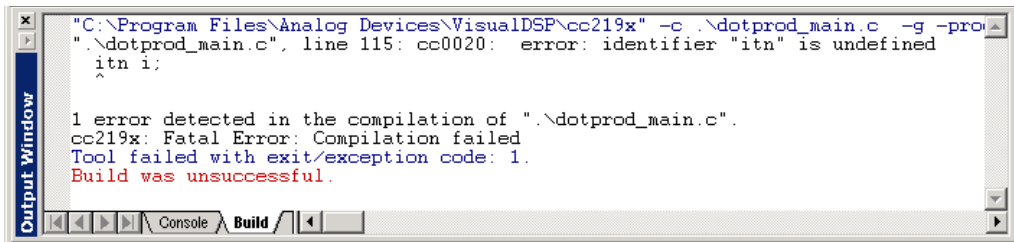


Figure 2-4. Example of Error Message

2. Double-click the error message (black) text in the **Output** window.

VisualDSP++ opens the C source file `dotprod_main.c` in an editor window and places the cursor on the line that contains the error (see [Figure 2-5 on page 2-9](#)).

The editor window in [Figure 2-5](#) shows that the integer variable declaration `int` has been misspelled as `itn`.

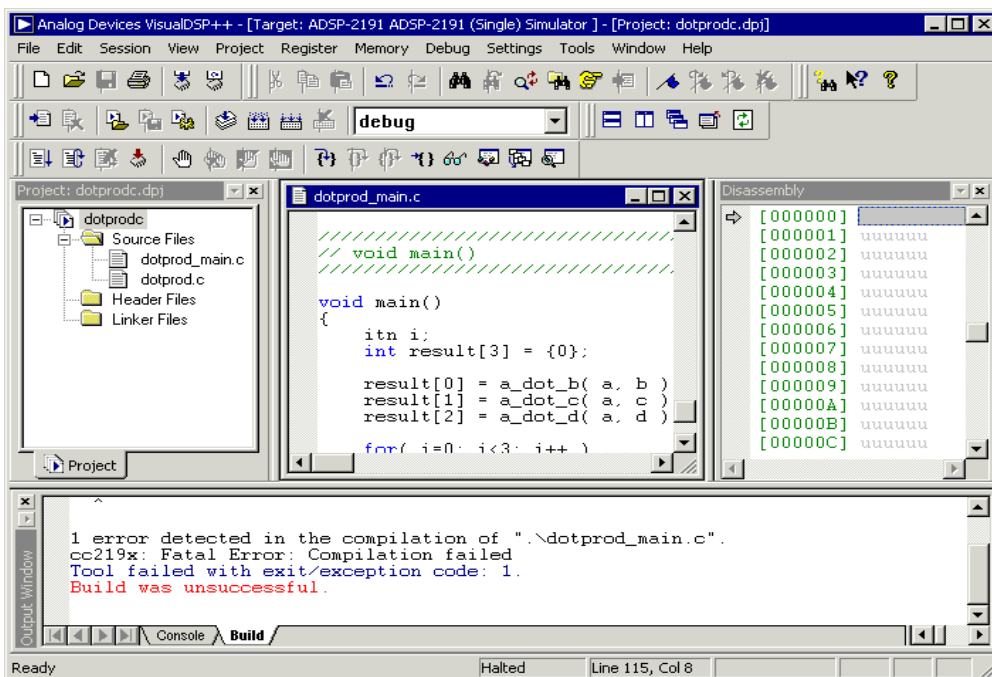


Figure 2-5. Output Window and Editor Window

3. In the editor window, click on `itn` and change it to `int`. Notice that `int` is now color coded to signify that it is a valid C keyword.
4. Save the source file by choosing **Save** from the **File** menu.
5. Build the project again by choosing **Build Project** from the **Project** menu. The project is now built without any errors, as reported on the **Build** page in the **Output** window.

Now that you have built your project successfully, you can run the example program.

Exercise One: Building and Running a C Program

Step 3: Run the Program

In this procedure, you will:

- Set up the debug session before running the program
- View debugger windows and dialog boxes

Since you enabled **Load executable after build** on the **General** page in the **Preferences** dialog box, the executable file `dotprodc.dxe` is automatically downloaded to the target.

If the debug session's processor does not match the project's build target, VisualDSP++ reports this discrepancy and asks if you want to select another session before downloading the executable to the target. If VisualDSP++ does not open the **Session List** dialog box, skip steps 1–4.

To set up the debug session:

1. In the **Session List** dialog box, click **New Session** to open the **New Session** dialog box, shown in [Figure 2-6 on page 2-11](#).

For subsequent debugging sessions, use the **New Session** command on the **Sessions** menu to open the **New Session dialog box**.

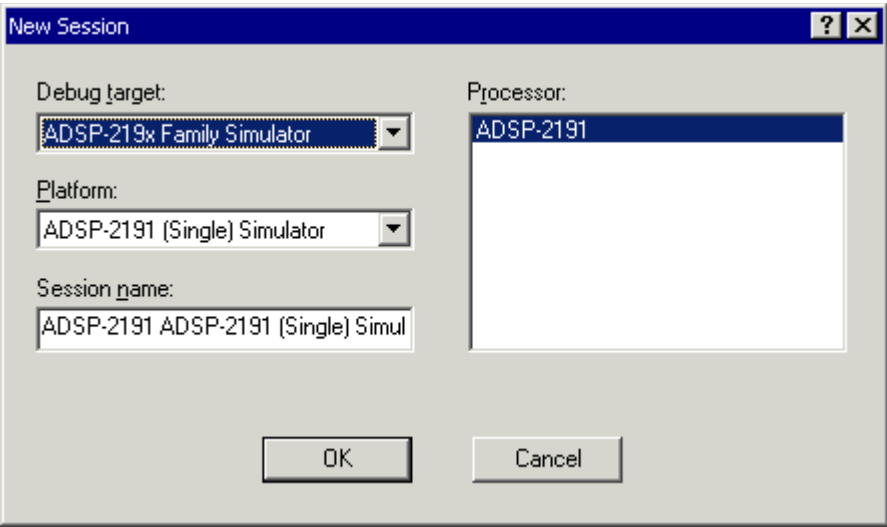


Figure 2-6. New Session Dialog Box

2. Specify the target and processor information listed in [Table 2-1](#).

Table 2-1. Session Specification

Box	Value
Debug Target	ADSP-219x Family Simulator
Platform	ADSP-2191 (Single) Simulator
Session Name	ADSP-2191 ADSP-2191 (Single) Simulator
Processor	ADSP-2191

3. Click **OK** to close the **New Session** dialog box and return to the **Session List** dialog box.

Exercise One: Building and Running a C Program

4. With the new session name highlighted, click **Activate**.



If you do not click **Activate**, the session mismatch message appears again.

VisualDSP++ closes the **Session List** dialog box, automatically loads your project's executable file (dotprodc.dxe), and advances to the main function of your code (see [Figure 2-7](#)).

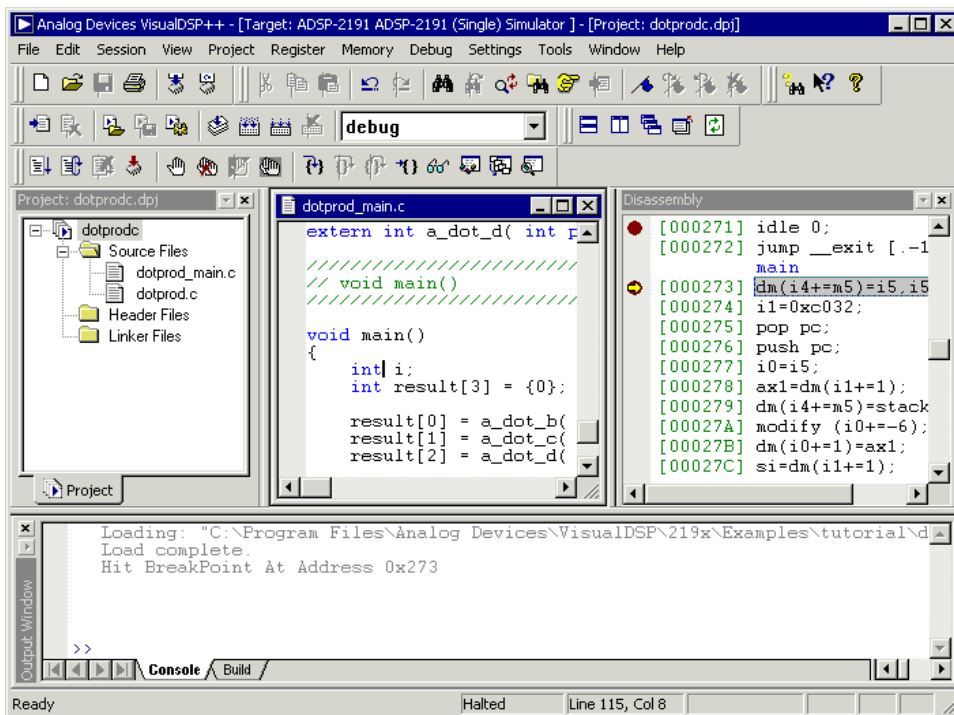


Figure 2-7. Loading dotprodc.dxe

5. Look at the information in the open windows.

The **Output** window's Console page contains messages about the status of the debug session. In this case, VisualDSP++ reports that the `dotprodc.dxe` load is complete.

The **Disassembly** window displays the machine code for the executable. Use the scroll bars to move around the **Disassembly** window.

Note that a solid red circle  appears at address `0x271` and at address `0x273`. A yellow arrow  appears at address `0x273`.

The solid red circle indicates that a breakpoint is set on that instruction, and the yellow arrow indicates that the processor is currently halted at that instruction. When VisualDSP++ loads your C program, it automatically sets two breakpoints, one at the beginning and one at the end of code execution.

Exercise One: Building and Running a C Program

- From the **Settings** menu, choose **Breakpoints** to view the breakpoints set in your program. VisualDSP++ displays the **Breakpoints** dialog box, shown in [Figure 2-8](#).

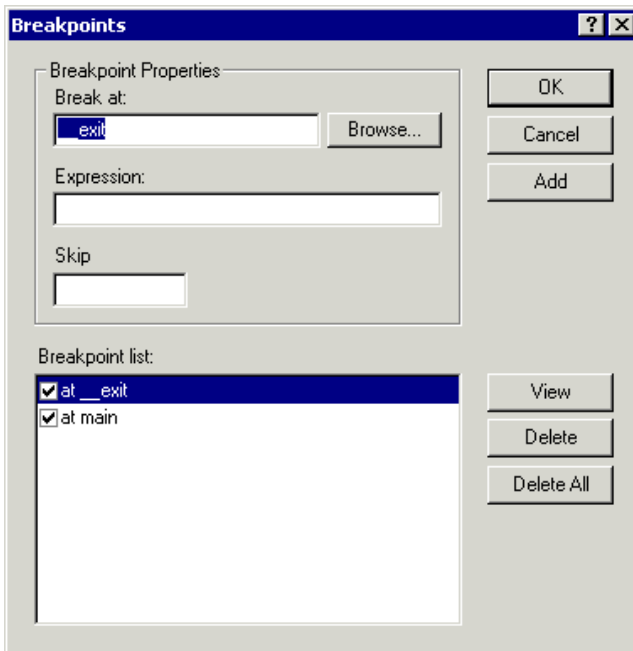


Figure 2-8. Breakpoints Dialog Box

The breakpoints are set at these C program labels:

- `__exit`
- `__main`

The **Breakpoints** dialog box enables you to view, add, and delete breakpoints and to browse for symbols. In the **Disassembly** and editor windows, double-clicking on a line of code toggles (adds or deletes) breakpoints. In the editor window, however, you must place the cursor in the gutter before double-clicking.

Use these tool buttons to set or clear breakpoints:



Toggles a breakpoint for the current line



Clears all breakpoints

7. Click **OK** or **Cancel** to exit the **Breakpoints** dialog box.

Step 4: Run dotprodc

To run `dotprodc`, click the **Run** button  or choose **Run** from the **Debug** menu.

VisualDSP++ computes the dot products and displays the following results on the **Console** page (Figure 2-9) in the **Output** window.

```
Dot product [0] = -30213
Dot product [1] = -17646
Dot product [2] = 24326
```



Figure 2-9. Results of the dotprodc Program

You are now ready to begin Exercise Two.

Exercise Two: Modifying a C Program to Call an Assembly Routine

Exercise Two: Modifying a C Program to Call an Assembly Routine

In Exercise One, you built and ran a C program. In this exercise, you will modify this program to call an assembly language routine, create a Linker Description File to link with the assembly routine, and rebuild the project. The project files are largely identical to those of Exercise One. Minor modifications illustrate the changes needed to call an assembly language routine from C source code.

Step 1: Create a New Project

To create a new project:

1. From the **Project** menu, choose **Close** to close the dotprodc project. Click **Yes** when prompted to close all open editor windows. If you have modified your project during this session, you are prompted to save the project. Click **No**.
2. From the **Project** menu, choose **New** to open the **Save New Project As** dialog box, shown in [Figure 2-10](#).

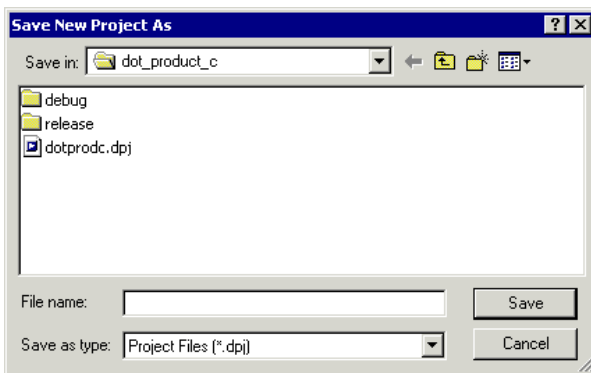


Figure 2-10. Save New Project As Dialog Box

3. Click the up-one-level button  until you locate the `dot_product_asm` folder, and then double-click this folder.
4. In the **File name** box, type `dot_product_asm`, and click **Save**.

The **Project Options** dialog box (Figure 2-11) appears.

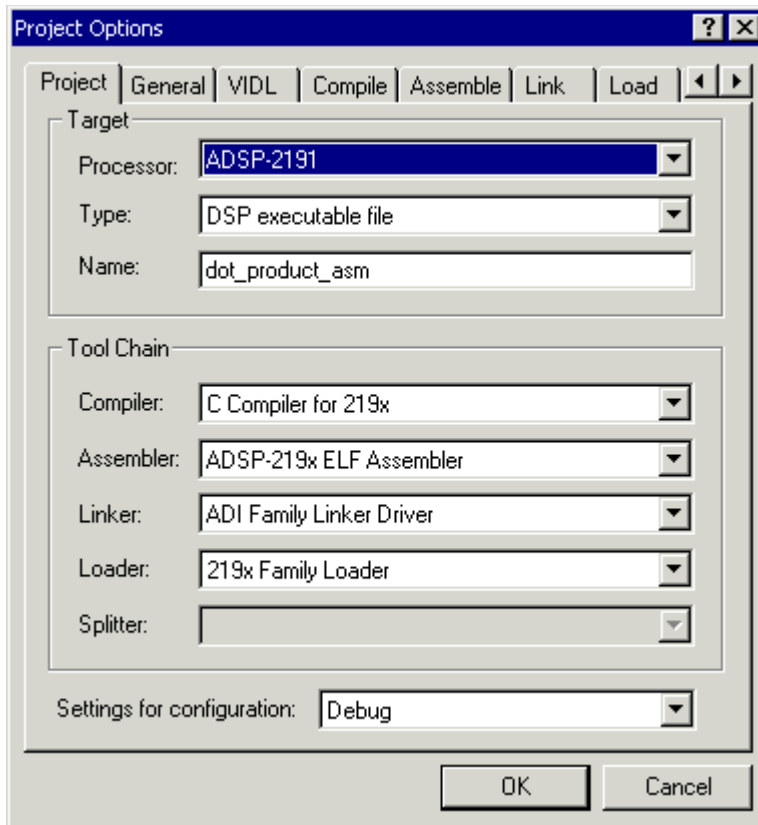


Figure 2-11. Project Options Dialog Box: Project Page

This dialog box enables you to specify project build information.

Exercise Two: Modifying a C Program to Call an Assembly Routine

5. Take a moment to view the tabbed pages in the **Project Options** window: **Project**, **General**, **VIDL**, **Compile**, **Assemble**, **Link**, **Load**, and **Post Build**. On each page, you specify the tool options used to build the project.
6. On the **Project** page ([Figure 2-11 on page 2-17](#)), specify the following values.

Table 2-2. Completing the Project Page

Box	Value
Processor	ADSP-2191
Type	DSP executable file
Name	dot_product_asm
Settings for configuration	Debug

These settings specify information for building an executable file for the ADSP-2191. The executable contains debug information, so you can examine program execution.

7. Click the **Compile** tab to display the **Compile** page, shown in [Figure 2-12 on page 2-19](#).

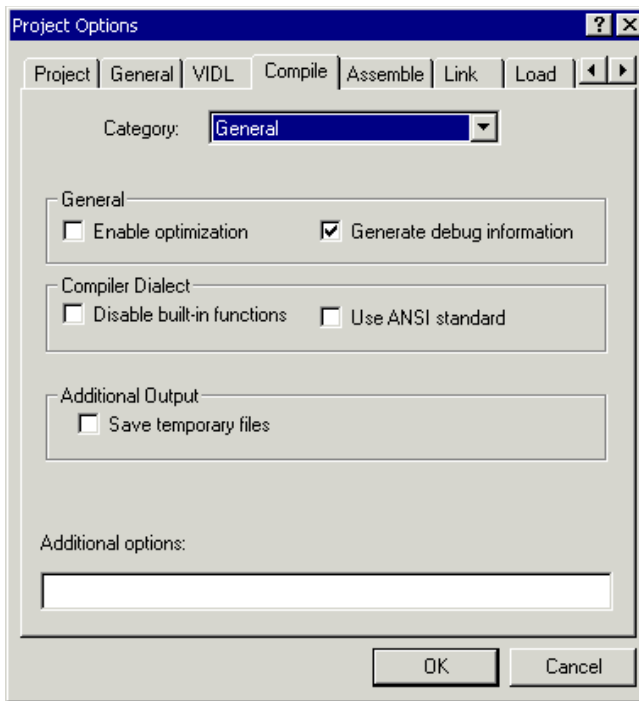


Figure 2-12. Project Options Dialog Box: Compile Page

8. In the **General** group box, select the **Generate debug information** check box, if it is not already selected, to enable debug information for the C source.
9. Click **OK** to apply changes to the project options and to close the **Project Options** dialog box. When prompted to add support for the VisualDSP++ kernel, click **No**.

You are now ready to add the source files to the project.

Exercise Two: Modifying a C Program to Call an Assembly Routine

Step 2: Add Source Files to dot_product_asm

To add the source files to the new project:

1. Click the **Add File** button , or from the **Project** menu, choose **Add to Project** and then choose **File(s)**.

The **Add Files** dialog box (Figure 2-13) appears.

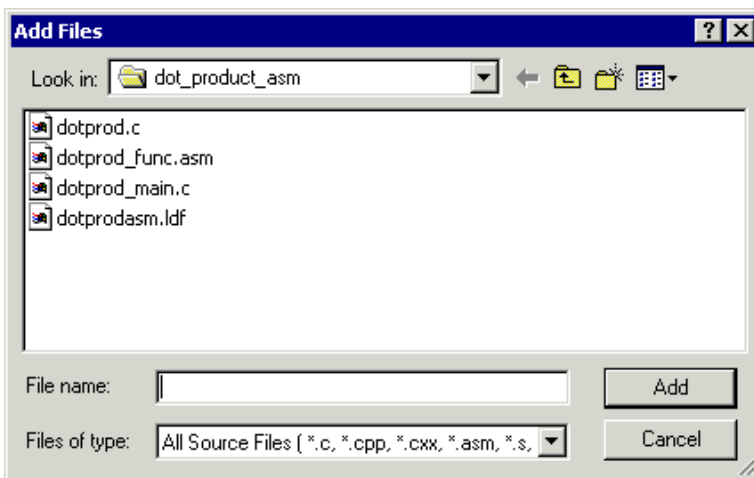


Figure 2-13. Add Files Dialog Box: Adding Source Files to the Project

2. In the **Look in** box, locate the project folder, `dot_product_asm`.
3. In the **Files of type** box, select **All Source Files**.
4. Hold down the **Ctrl** key and click `dotprod.c` and `dotprod_main.c`. Then click **Add**.

To display the files that you added in step 4, open the **Source Files** folder in the **Project** window.

You are now ready to create a Linker Description File for the project.

Step 3: Create a Linker Description File for the Project

In this procedure, you will use the Expert Linker to create a Linker Description File for the project.

To create a Linker Description File:

1. From the **Tools** menu, choose **Expert Linker** and then choose **Create LDF** to open the **Create LDF Wizard**, shown in [Figure 2-14](#).

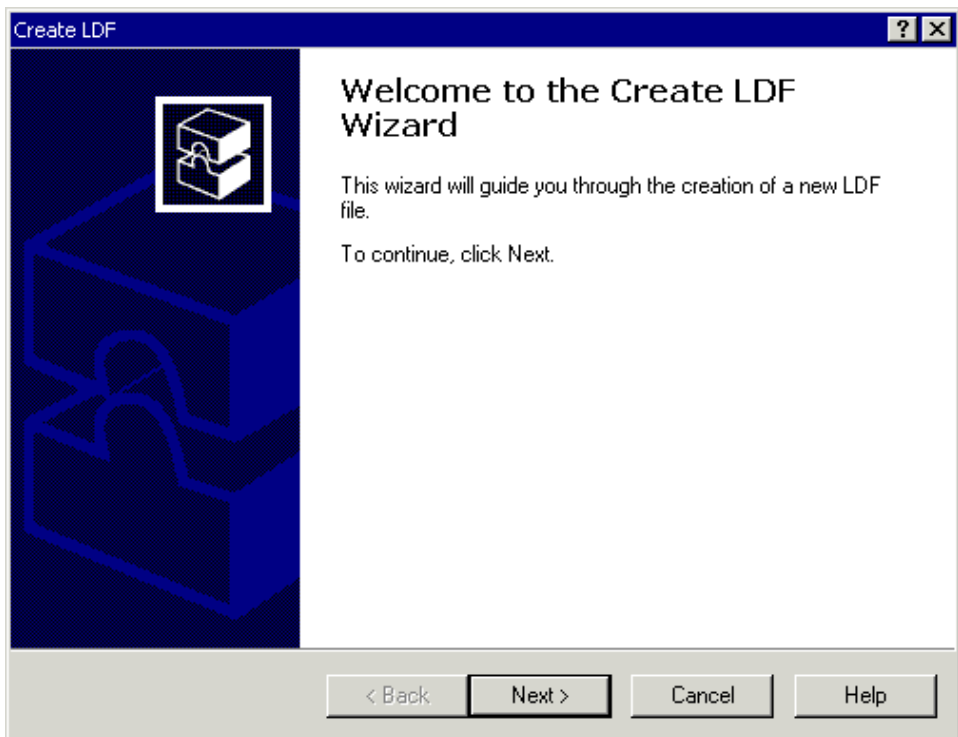


Figure 2-14. Create LDF Wizard

Exercise Two: Modifying a C Program to Call an Assembly Routine

2. Click **Next** to display the **Create LDF – Step 1 of 3** page, shown in [Figure 2-15](#).

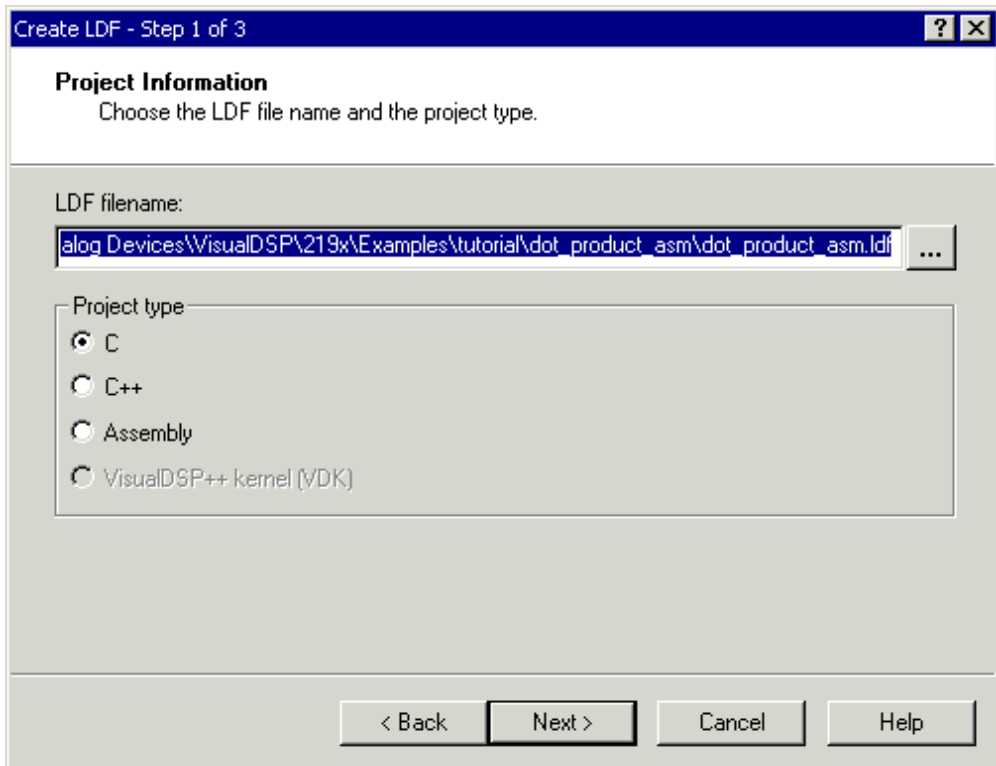


Figure 2-15. Create LDF – Step 1 of 3 Page

This page enables you to assign the **LDF file name** (based on the project name) and to select the **Project type**.

3. Accept the values selected for your project and click **Next** to display the **Create LDF – Step 2 of 3** page, shown in [Figure 2-16 on page 2-23](#).

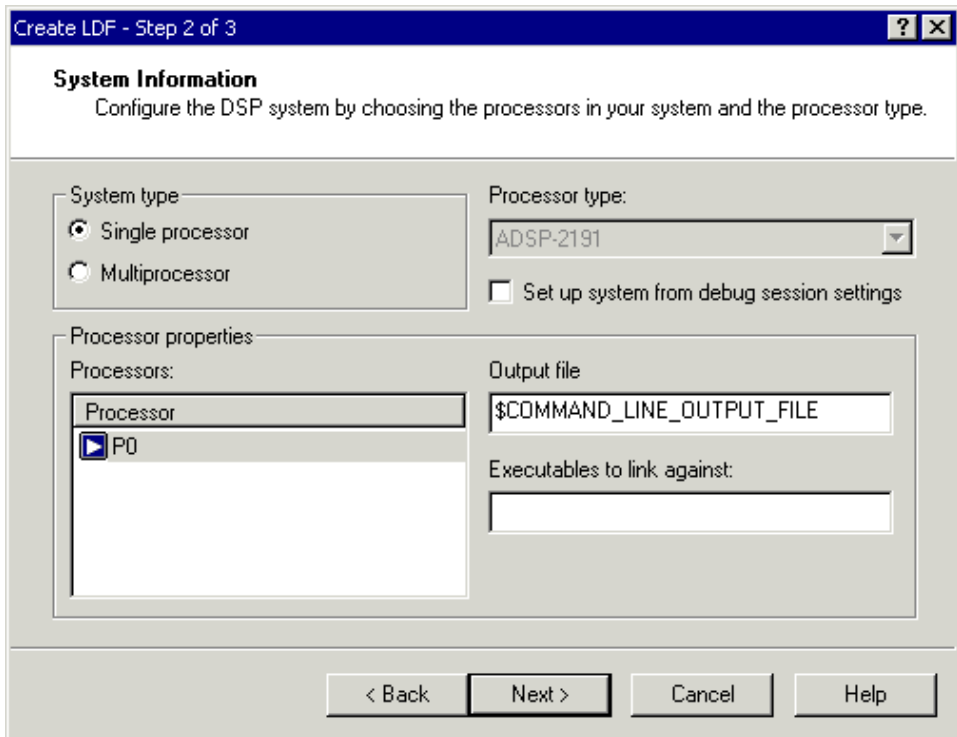


Figure 2-16. Create LDF – Step 2 of 3 Page

This page enables you to set the **System type** (defaulted to **Single processor**), the **Processor type** (defaulted to **ADSP-2191** to match the project), and the name of the linker **Output file** (defaulted to the name selected by the project).

4. Accept the default values and click **Next** to display the **Create LDF – Step 3 of 3** page, shown in [Figure 2-17 on page 2-24](#).

Exercise Two: Modifying a C Program to Call an Assembly Routine

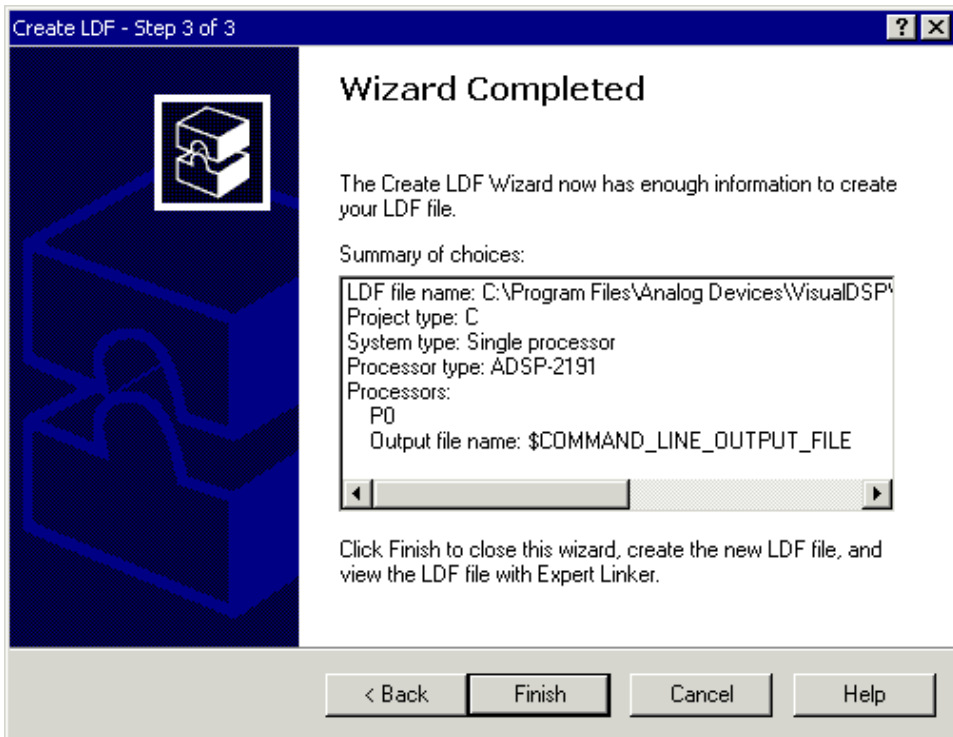


Figure 2-17. Create LDF – Step 3 of 3 Page

5. Review the **Summary of choices** and click **Finish** to create the .LDF file.

You now have a new .LDF file in your project. The new file is in the **Linker Files** folder in the **Project** window.

The **Expert Linker** window opens with a representation of the .LDF file that you created. This file is complete for this project. Close the **Expert Linker** window.

6. Click the **Build Project** button  to build the project. The C source file opens in an editor window, and execution halts.

The C version of the project is now complete. You are now ready to modify the sources to call the assembly function.

Step 4: Modify the Project Source Files

In this procedure, you will:

- Modify `dotprod_main.c` to call `a_dot_c_asm` instead of `a_dot_c`
- Save the modified file

To modify `dotprod_main.c` to call the assembly function:

1. Resize or maximize the editor window for better viewing.
2. From the **Edit** menu, choose **Find** to open the **Find** dialog box, shown in [Figure 2-18](#).

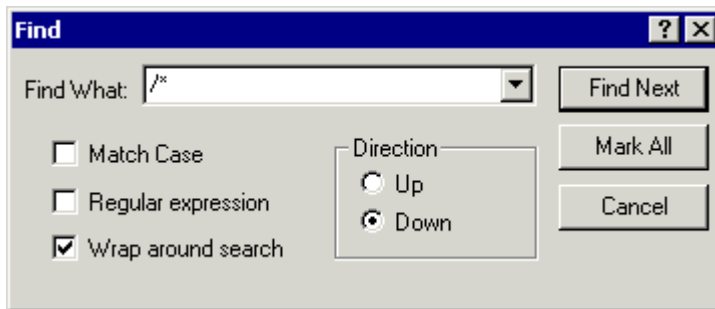


Figure 2-18. Find Dialog Box: Locating Occurrences of /*

Exercise Two: Modifying a C Program to Call an Assembly Routine

3. In the **Find What** box, type `/*`, and then click **Mark All**.

The editor bookmarks all lines containing `/*` and positions the cursor at the first instance of `/*` in the `extern int a_dot_c_asm` declaration.

4. Select the comment characters `/*` and use the **Ctrl+X** key combination to cut the comment characters from the beginning of the `a_dot_c_asm` declaration. Then move the cursor up one line and use the **Ctrl+V** key combination to paste the comment characters at the beginning of the `a_dot_c` declaration. Because syntax coloring is turned on, the code will change color as you cut and paste the comment characters.

Repeat this step for the end-of-comment characters `*/` at the end of the `a_dot_c_asm` declaration. The `a_dot_c` declaration is now fully commented out, and the `a_dot_c_asm` declaration is no longer commented.

5. Press **F3** to move to the next bookmark.

The editor positions the cursor on the `/*` in the function call to `a_dot_c_asm`, which is currently commented out. Note that the previous line is the function call to the `a_dot_c` routine.

6. Press **Ctrl+X** to cut the comment characters from the beginning of the function call to `a_dot_c_asm`. Then move the cursor up one line and press **Ctrl+V** to paste the comment characters at the beginning of the call to `a_dot_c`.

Repeat this step for the end-of-comment characters `*/`. The `main()` function should now be calling the `a_dot_c_asm` routine instead of the `a_dot_c` function, previously called in Exercise One.

[Figure 2-19 on page 2-27](#) shows the changes made in step 6.

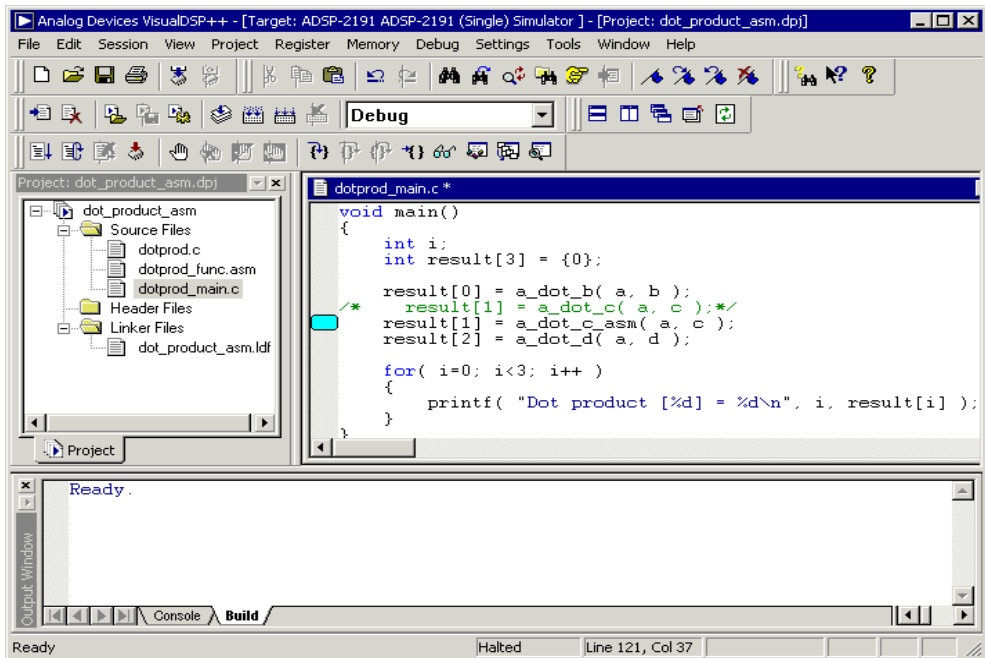


Figure 2-19. Editor Window: Modifying dotprod_main.c to Call a_dot_c_asm

7. From the **File** menu, choose **Save** to save the changes to the file.
8. Place the cursor in the editor window. Then, from the **File** menu, choose **Close** to close the dotprod_main.c file.

You are now ready to modify dotprodasm.ldf.

Exercise Two: Modifying a C Program to Call an Assembly Routine

Step 5: Use the Expert Linker to modify dot_prod_asm.ldf

In this procedure you will:

- View the Expert Linker representation of the .LDF file that you created
- Modify the .LDF file to map in the section for the a_dot_c_asm assembly routine

To examine and then modify dot_prod_asm.ldf to link with the assembly function:

1. Click the **Add File** button  .
2. Select dotprod_func.asm and click **Add**.
3. Try to build the project by performing one of these actions:
 - Click the **Build Project** button  .
 - From the **Project** menu, choose **Build Project**.

Notice the linker error in the **Output** window, shown in [Figure 2-20](#).

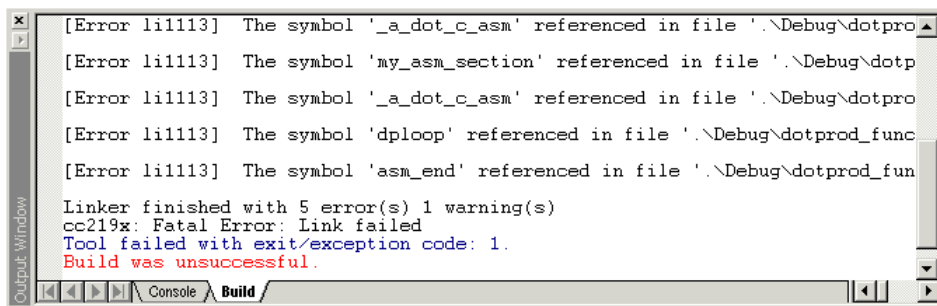


Figure 2-20. Output Window: Linker Error

4. In the **Project** window, double-click in the `dot_prod_asm.ldf` file. The **Expert Linker** window (Figure 2-21) opens with a representation of your file.

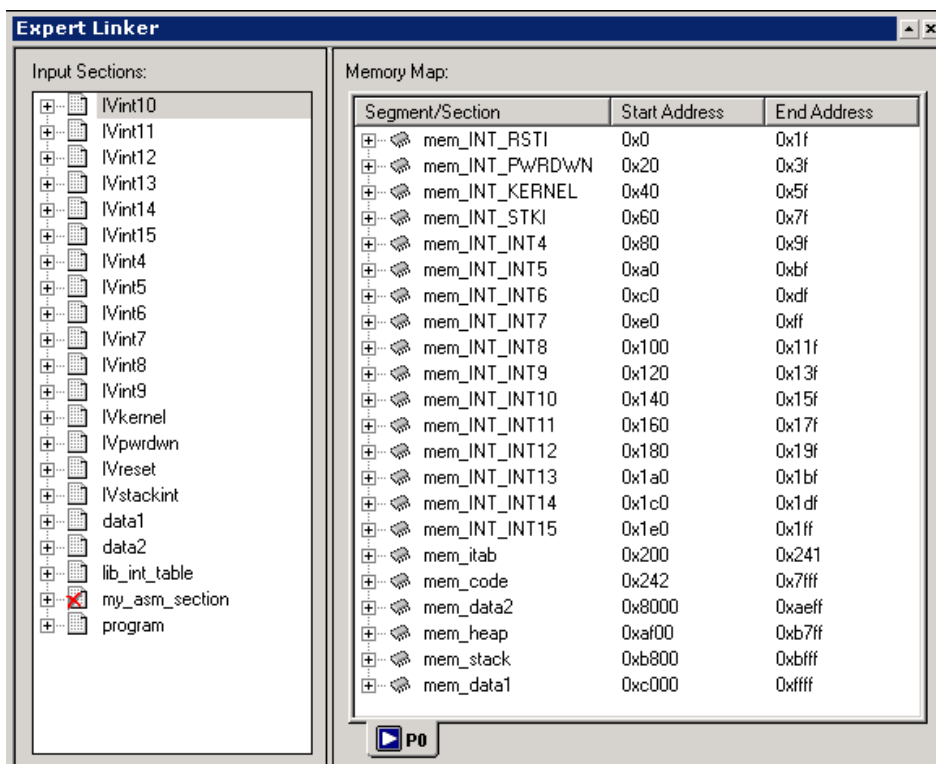


Figure 2-21. Expert Linker Window

The left pane contains a list of the **Input Sections** that are in your project or are mapped in the `.LDF` file. A red **X** is over the icon in front of the section named "my_asm_section" because the Expert Linker has determined that the section is not mapped by the `.LDF` file.

Exercise Two: Modifying a C Program to Call an Assembly Routine

The right pane contains a graphical representation of the memory segments that the Expert Linker defined when it created the .LDF file. Change the view mode by right-clicking in the right pane and choosing **View Mode**. Then choose **Memory Map Tree** to display the tree view shown in [Figure 2-21](#).

5. Map the section `my_asm_section` into the memory segment named `mem_code` as follows.

Open the `my_asm_section` input section by clicking on the plus sign in front of it. The input section expands to show that the linker macro `$OBJECTS` and the object file `dotprod_func.doj` both have a section that has not been mapped. Drag the icon in front of `$OBJECTS` to the memory map pane and onto the `mem_code`. As shown in [Figure 2-22 on page 2-31](#), the red X should no longer appear because the section `my_asm_section` has been mapped.

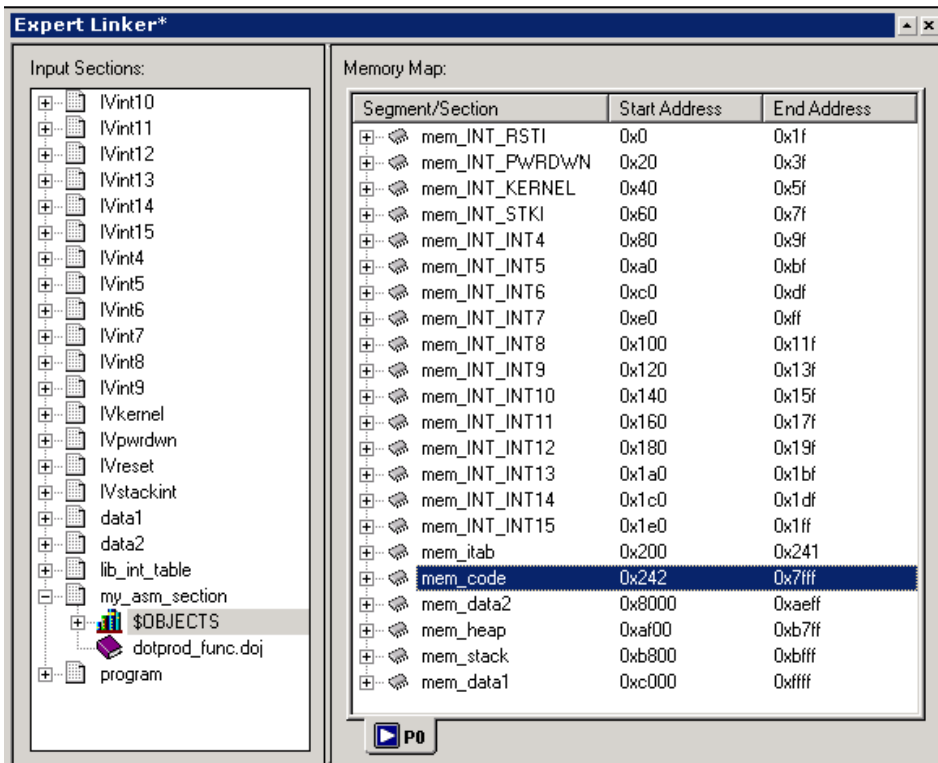


Figure 2-22. Dragging \$OBJECTS onto mem_code

- From the **Tools** menu, choose **Expert Linker** and **Save** to save the modified file. Then close the **Expert Linker** window.

If you forget to save the file and then rebuild the project, VisualDSP++ will see that you modified the file and will automatically save it.

You are now ready to rebuild and run the modified project.

Exercise Two: Modifying a C Program to Call an Assembly Routine

Step 6: Rebuild and Run dot_product_asm

To run dot_product:

1. Build the project by performing one of these actions:

- Click the **Build Project** button  .
- From the **Project** menu, choose **Build Project**.

At the end of the build, the **Output** window displays “Build completed successfully” on the **Build** page. VisualDSP++ loads the program, runs to main, and displays the **Output**, **Disassembly**, and editor windows (shown in [Figure 2-23](#)).

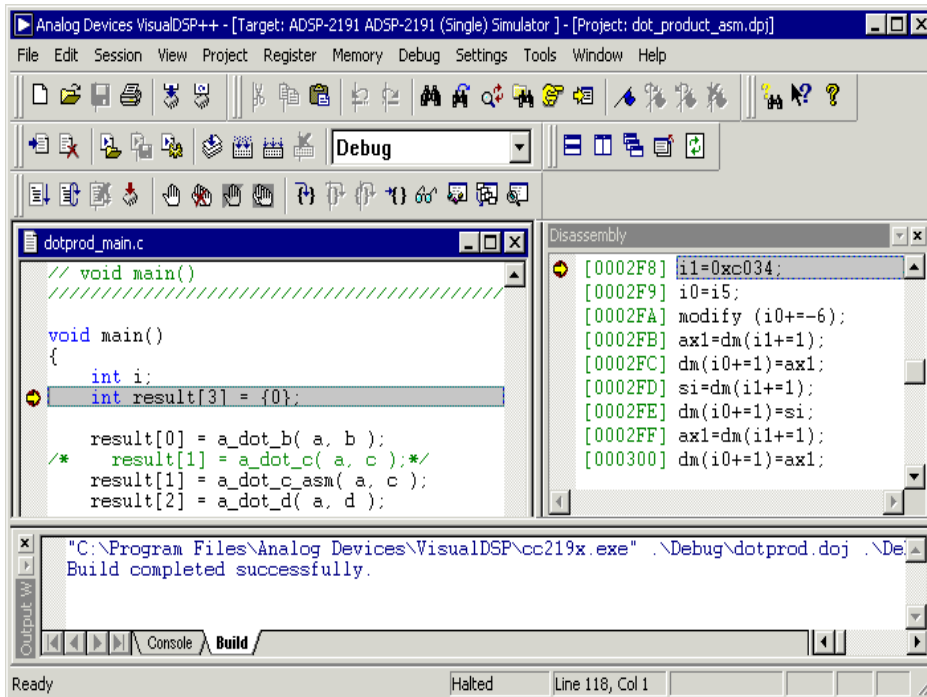


Figure 2-23. Windows Left Open from the Previous Debugger Session

2. Click the **Run** button  to run `dot_product_asm`.

The program calculates the three dot products and displays the results on the **Console** page in the **Output** window. When the program stops running, the message “Halted” appears in the status bar at the bottom of the window. The results, shown below, are identical to the results obtained in Exercise One.

```
Dot product [0] = -30213
Dot product [1] = -17646
Dot product [2] = 24326
```

You are now ready to begin Exercise Three.

Exercise Three: Plotting Data

In this exercise, you will load and debug a pre-built program that applies a simple Finite Impulse Response (FIR) filter to a buffer of data. You will use VisualDSP++’s plotting engine to view the different data arrays graphically, both before and after running the program.

Step 1: Load the FIR Program

To load the FIR program:

1. Keep the **Disassembly** window and **Console** page (in the **Output** window) open, but close all other windows.
2. From the **File** menu, choose **Load Program** or click . The **Open a Processor Program** dialog box appears.

Exercise Three: Plotting Data

3. Select the FIR program to load as follows.
 - a. Open your Analog Devices folder and double-click the VisualDSP\219x\Examples\Tutorial\fir\Debug subfolder.
 - b. Double-click FIR.DXE to load the program in an editor window (Figure 2-24).

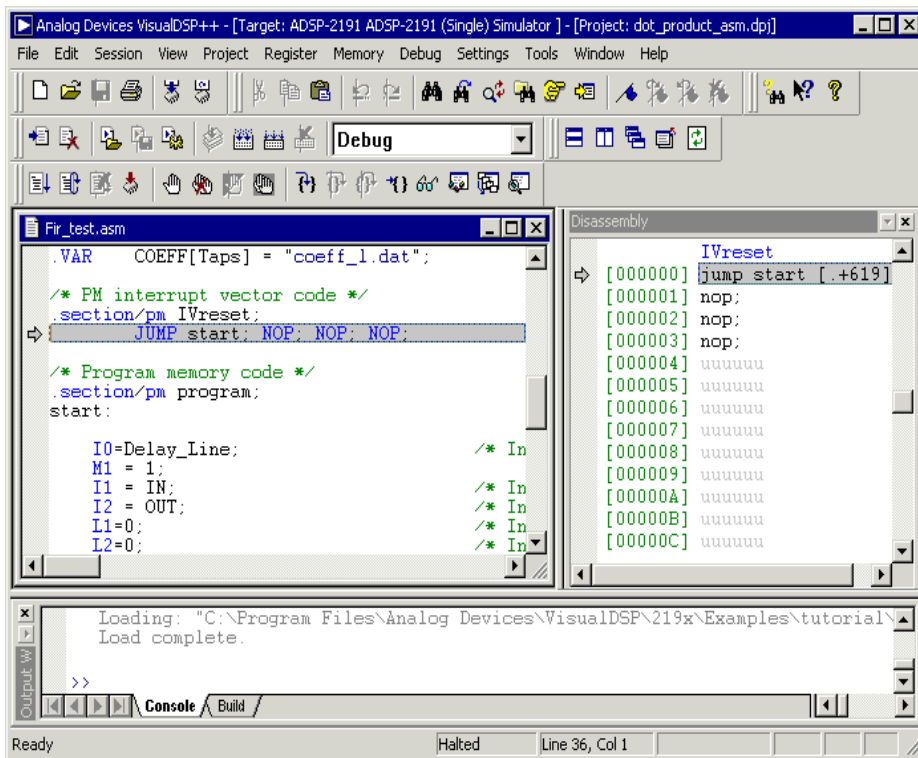


Figure 2-24. Loading the FIR Program

4. Look at the source code of the FIR program.

You can see two global data arrays:

- IN
- OUT

You can also see one function, `fir`, that operates on these arrays.

You are now ready to open a plot window.

Step 2: Open a Plot Window

To open a plot window:

1. From the **View** menu, choose **Debug Windows** and **Plot**. Then choose **New** to open the **Plot Configuration** dialog box, shown in [Figure 2-25 on page 2-36](#).

Exercise Three: Plotting Data

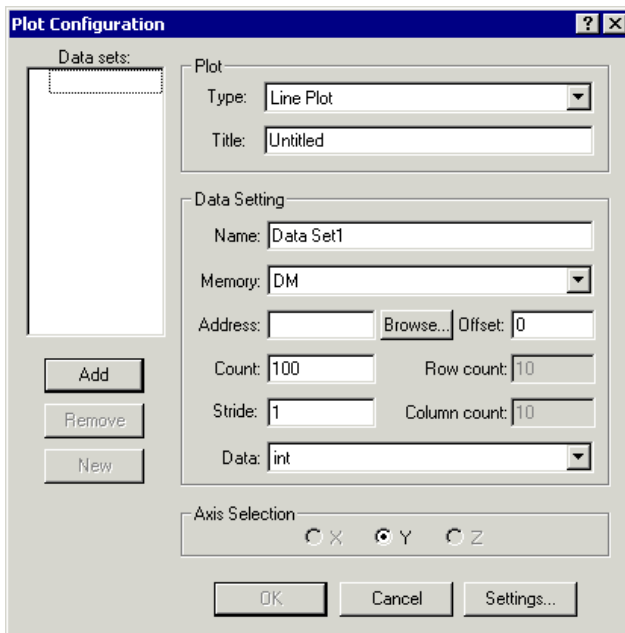


Figure 2-25. Plot Configuration Dialog Box

Here you will add the data sets that you want to view in a plot window.

2. In the **Plot** group box, specify the following values.
 - In the **Type** box, select **Line Plot** from the drop-down menu.
 - In the **Title** box, type **fir**.

3. Enter two data sets to plot by using the values in [Table 2-3](#).

Table 2-3. Two Data Sets: Input and Output

Box	Input Data Set	Output Data Set	Description
Name	Input	Output	Data set
Memory	DM	DM	Data memory
Address	IN	OUT	The address of this data set is that of the Input or Output array. Click Browse to select the value from the list of loaded symbols.
Count	128	128	The array is 260 elements long, but you are plotting the first 128 elements.
Stride	1	1	The data is contiguous in memory.
Data	int	int	Input and Output are arrays of int values.

After entering each data set, click **Add** to add the data set to the **Data Sets** list. The **Plot Configuration** dialog box should now look like the one in [Figure 2-26 on page 2-38](#).

Exercise Three: Plotting Data

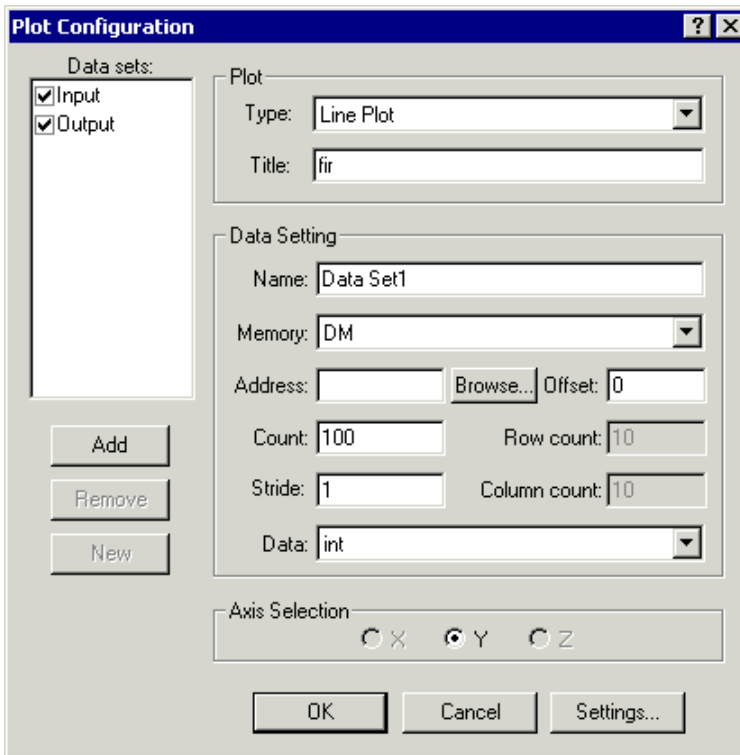


Figure 2-26. Plot Configuration Dialog Box with Input/Output Data Sets

4. Click **OK** to apply the changes and to open a plot window with these data sets.

The plot window now displays the two arrays. Since, by default, the simulator initializes memory to zero, the Output data set appears as one horizontal line, shown in [Figure 2-27](#).

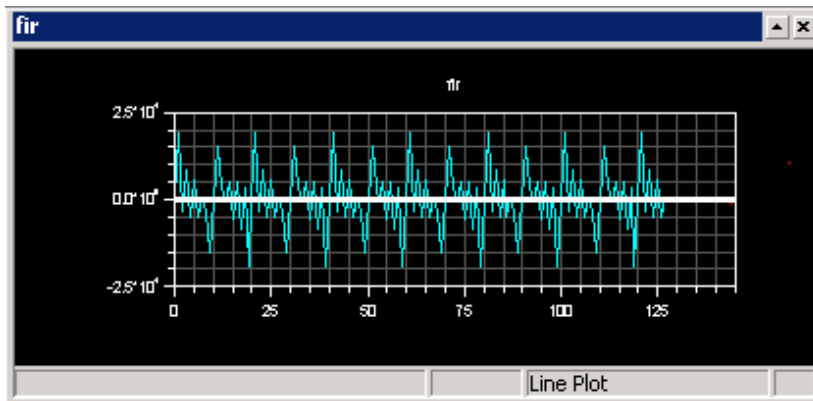


Figure 2-27. Plot Window: Before Running the FIR Program

To display the legend box (not shown) in the plot window, right-click in the plot window and choose **Modify Settings**. Then, on the **General** page, in the **Options** group box, select **Legend**.

Exercise Three: Plotting Data

Step 3: Run the FIR Program and View the Data

To run the FIR program and view the data:

1. Press F5 or click the **Run** button  to run to the end of the program.

When the program halts, you see the results of the FIR filter in the Output array. The two data sets are visible in the plot window, as shown in [Figure 2-28](#).

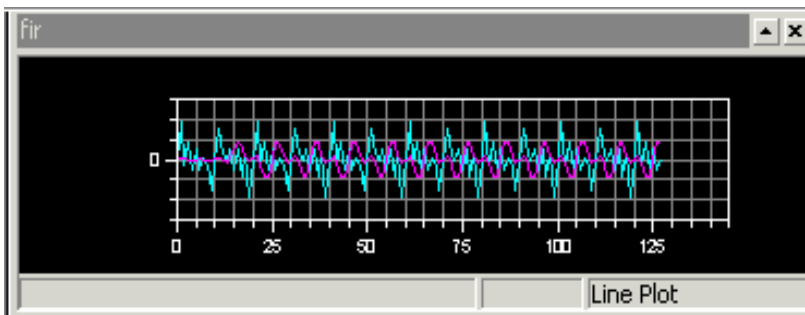


Figure 2-28. Plot Window After Running the FIR Program to Completion

Next you will zoom in on a particular region of interest in the plot window to focus in on the data.

- Click the left mouse button inside the plot window and drag the mouse to create a rectangle to zoom into. Then release the mouse button to magnify the selected region.

Figure 2-29 shows the selected region, and Figure 2-30 shows the magnified result.

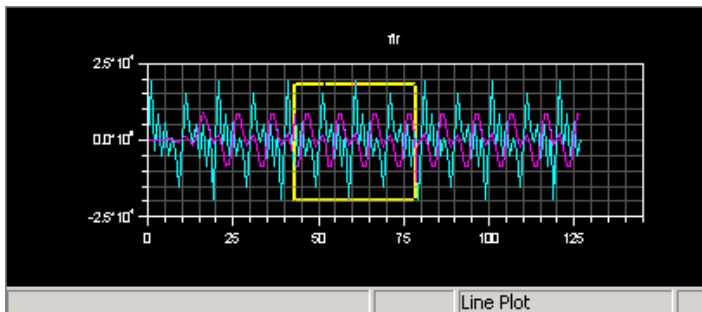


Figure 2-29. Plot Window: Selecting a Region to Magnify

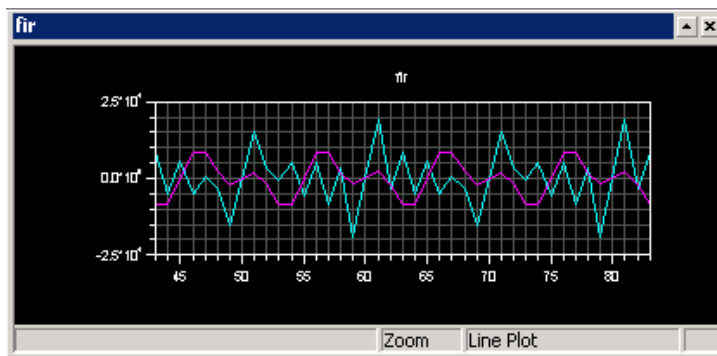


Figure 2-30. Plot Window: Magnified Result

To return to the view before magnification, right-click in the plot window and choose **Reset Zoom** from the menu. You can view individual data points in the plot window by enabling the data cursor, as explained in the next step.

Exercise Three: Plotting Data

3. Right-click inside the plot window and choose **Data Cursor** from the popup menu. Then move through the individual data points in the current data set by pressing and holding the Left ($\leftarrow$) and Right ($\rightarrow$) arrow keys on the keyboard. The value of the current data point appears in the lower-left corner of the plot window, as shown in Figure 2-31.

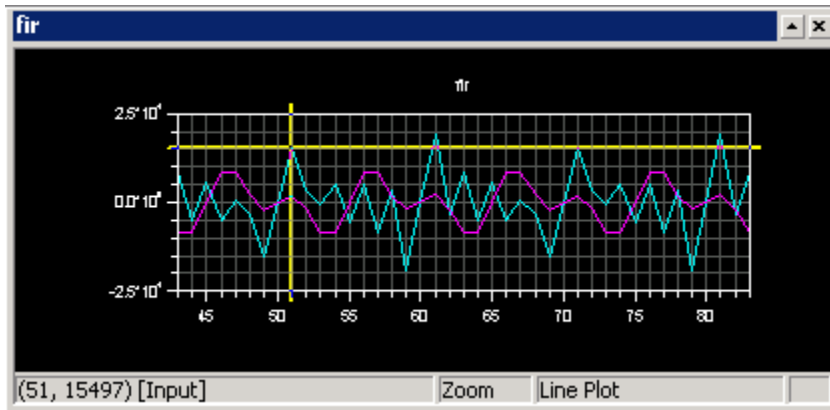


Figure 2-31. Plot Window: Using the Data Cursor Feature

To switch data sets, press the Up ($\uparrow$) and Down ($\downarrow$) arrow key.

4. Right-click in the plot window and choose **Data Cursor** from the pop-up menu.

In the next step, you will look at data sets in the frequency domain.

5. Right-click in the plot window and choose **Modify Settings** to open the **Plot Settings** dialog box.

6. Complete these steps:
 - a. Click the **Data Processing** tab to display the **Data Processing** page, shown in [Figure 2-32](#).

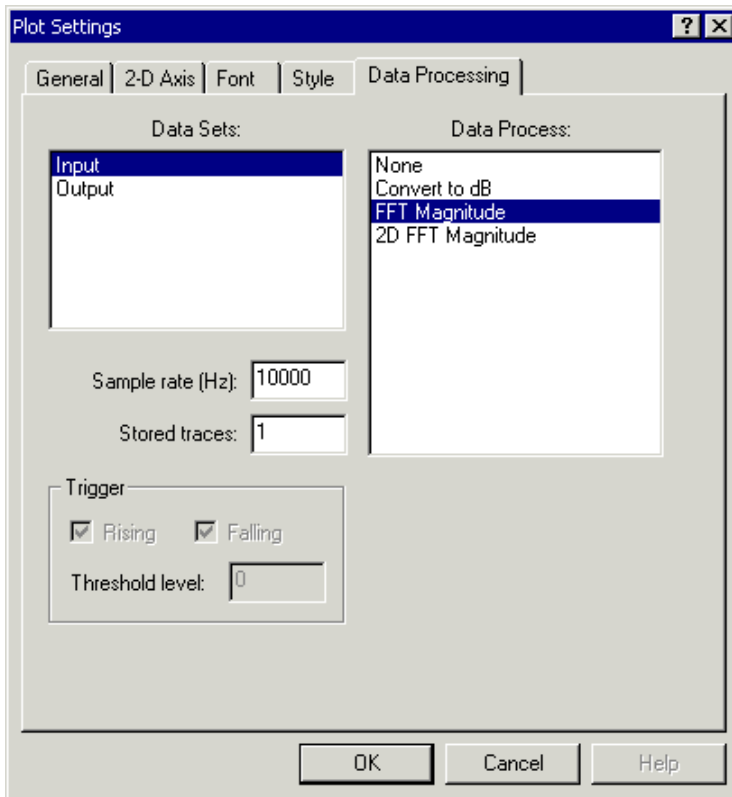


Figure 2-32. Data Processing Page

- b. In the **Data Sets** box, make sure that **Input** (the default) is selected. In the **Data Process** box, choose **FFT Magnitude**.
 - c. In the **Sample rate (Hz)** box, type 10000.

Exercise Three: Plotting Data

- d. In the **Data Sets** box, select **Output**. In the **Data Process** box, choose **FFT Magnitude**
- e. Click **OK** to exit the **Data Processing** page.

VisualDSP++ performs a Fast Fourier Transform (FFT) on the selected data set before it is plotted. FFT enables you to view the signal in the frequency domain, as shown in [Figure 2-33](#).

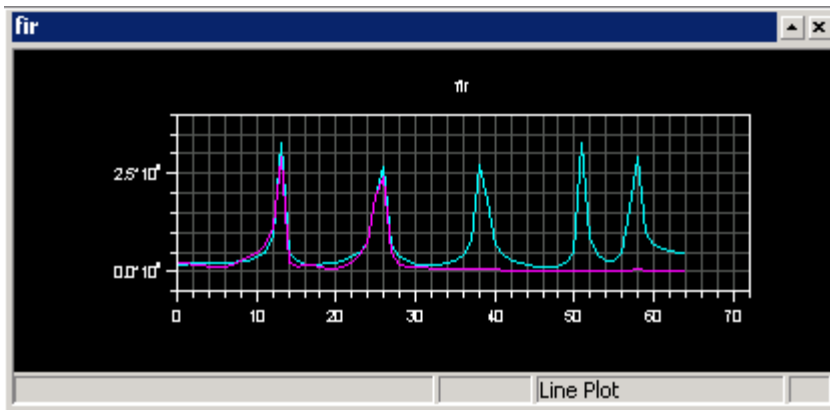


Figure 2-33. FFT Performed on a Selected Data Set

You are now ready to begin Exercise Four.

Exercise Four: Linear Profiling

In this exercise, you will load and debug the FIR program from the previous exercise. You will then use linear profiling to evaluate the program's efficiency and to determine where the application is spending the majority of its execution time in the code.

VisualDSP++ supports two types of profiling: linear and statistical.

- You use linear profiling with a simulator. The count in the **Linear Profiling Results** window is incremented every time a line of code is executed.
- You use statistical profiling with a JTAG emulator connected to a DSP target. The count in the **Statistical Profiling Results** window is based on random sampling.

Step 1: Load the FIR Program

To load the FIR program:

1. Close all open windows except for the **Disassembly** window and the **Output** window.
2. From the **File** menu, choose **Load Program**, or click . The **Open a Processor Program** dialog box appears.
3. Select the program to load as follows.
 - a. Open the **Analog Devices** folder and double-click the **VisualDSP\219x\Examples\Tutorial\fir\Debug** subfolder.
 - b. Double-click **FIR.DXE** to load and run the FIR program. VisualDSP++ opens an editor window.

You are now ready to set up linear profiling.

Exercise Four: Linear Profiling

Step 2: Open the Profiling Window

To open the linear profiling window:

1. From the **Tools** menu, choose **Linear Profiling** and then choose **New Profile**.

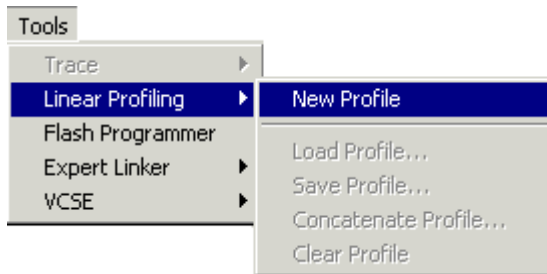


Figure 2-34. Setting Up Linear Profiling for the FIR Program

The **Linear Profiling Results** window opens.

2. For a better view of the data, use the window's title bar to drag and dock the window to the top of the VisualDSP++ main window, as shown in [Figure 2-35 on page 2-47](#).

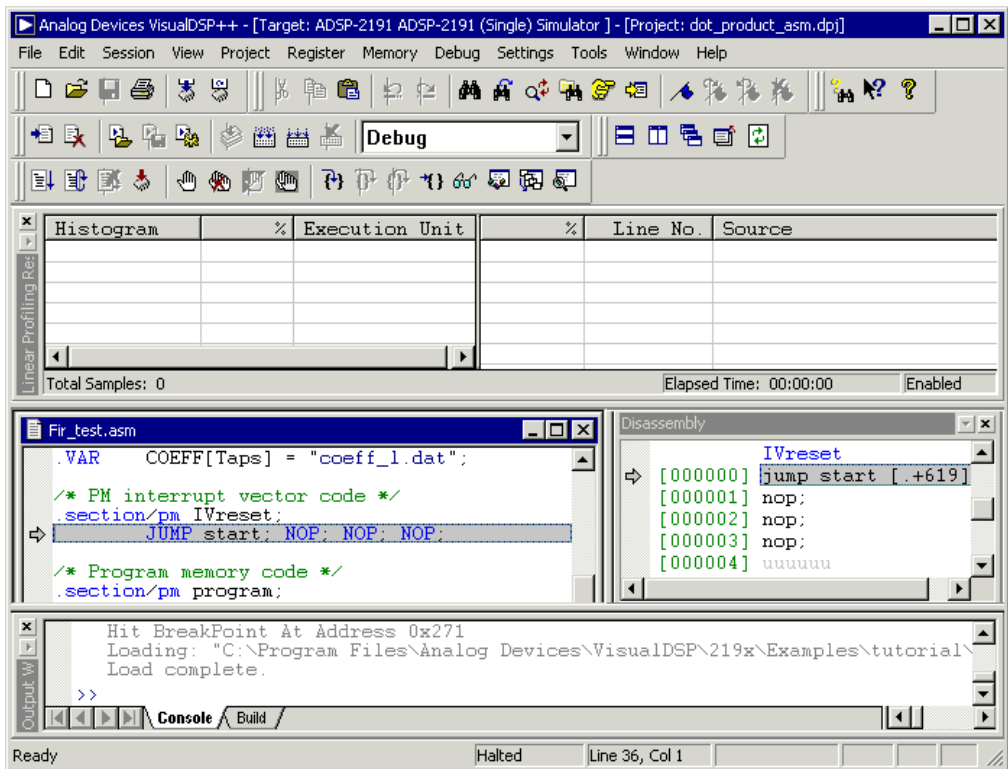


Figure 2-35. Linear Profiling Results Window (Empty)

The **Linear Profiling Results** window is initially empty. Linear profiling will be performed when you run the FIR program. After you run the program and collect data, this window displays the results of the profiling session.

You are now ready to collect and examine linear profile data.

Exercise Four: Linear Profiling

Step 3: Collect and Examine the Linear Profile Data

To collect and examine the linear profile data:

1. Press F5 or click  to run to the end of the program.

When the program halts, the results of the linear profile appear in the **Linear Profiling Results** window.

2. Examine the results of your linear profiling session.

The **Linear Profiling Results** window is divided into two, three-column panes. The left pane shows the results of the profile data. Double-clicking on a line in the left pane displays the corresponding source code for the profile data in the right pane, as shown in [Figure 2-36](#).

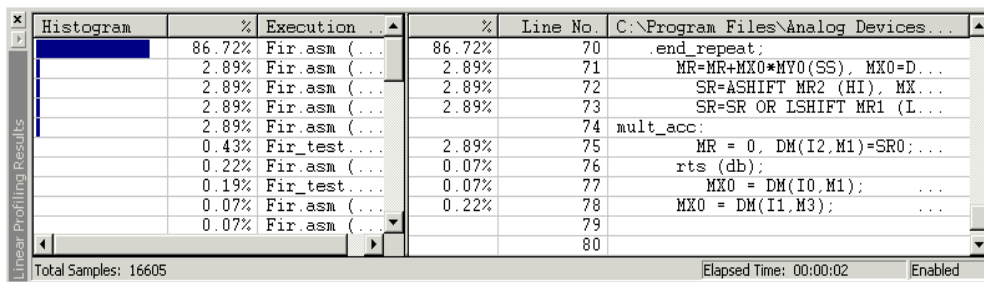


Figure 2-36. Linear Profiling Results of Analyzing the Performance of the FIR Program

The field values in the left pane are defined as follows.

Histogram	A graphical representation of the percentage of time spent in a particular execution unit. This percentage is based on the total time that the program spent running, so longer bars denote more time spent in a particular execution unit. The Linear Profiling Results window always sorts the data with the most time-consuming (expensive) execution units at the top.
%	The numerical percent of the same data found in the Histogram column. You can view this value as an absolute number of samples by right-clicking in the Linear Profiling Results window and by selecting View Sample Count from the popup menu.
Execution Unit	The program location to which the samples belong. If the instructions are inside a C function or a C++ method, the execution unit is the name of the function or method. For instructions that have no corresponding symbolic names, such as hand-coded assembly or source files compiled without debugging information, this value is an address in the form of <code>PC[xxx]</code> , where <code>xxx</code> is the address of the instruction.

If the instructions are part of an assembly file, the execution unit is the assembly file followed by the line number in parentheses.

In [Figure 2-36 on page 2-48](#) the left pane shows that line 70 in the `fir.asm` file has consumed over 86% of the total execution time. If you double-click one of these lines, the right (source) pane displays `fir.asm` and jumps to that line. The source pane displays data for each line of executable code in the file for which linear profile data has been collected.

Exercise Five: Multiprocessor Debugging

Figure 2-37 shows the linear profile data for `fir.asm`.

%	Line No.	C:\Program Files\Analog Devices...
	62	<code>fir:</code>
0.07%	63	<code>MR = 0, MX0 = DM(I1,M1), MY...</code>
0.07%	64	<code>DM(I0,M3) = MX0; ...</code>
	65	
0.07%	66	<code>CNTR=Samps_per_Bin; ...</code>
0.07%	67	<code>DO mult_acc UNTIL CE;</code>
	68	<code>.repeat (Taps-1); ...</code>
	69	<code>MR=MR+MX0*MY0(SS), MX0=D...</code>
86.72%	70	<code>.end_repeat;</code>
2.89%	71	<code>MR=MR+MX0*MY0(SS), MX0=D...</code>
2.89%	72	<code>SR=ASHIFT MR2 (HI), MX...</code>
2.89%	73	<code>SR=SR OR LSHIFT MR1 (L...</code>

Figure 2-37. Linear Profile Data for `fir.asm`

The details of `fir.asm` show that line 70 spent 86.72% of the time in the inner loop of the FIR routine.

You are now ready to begin Exercise Five.

Exercise Five: Multiprocessor Debugging

In this exercise you will create a multiprocessor (MP) simulator session and explore the various features for debugging complex multiprocessor systems. You will use synchronous MP commands to control multiple targets, pin windows to specific processors, and use MP groups to control multiple subsets of processors in an MP system.

Step 1: Create a Multiprocessor Simulator Session

To create a multiprocessor simulator session:

1. If necessary, start VisualDSP++. (VisualDSP++ automatically connects to the last session that was open.)
2. From the Session menu, choose **New Session** to open the New Session dialog box, shown in [Figure 2-38](#).

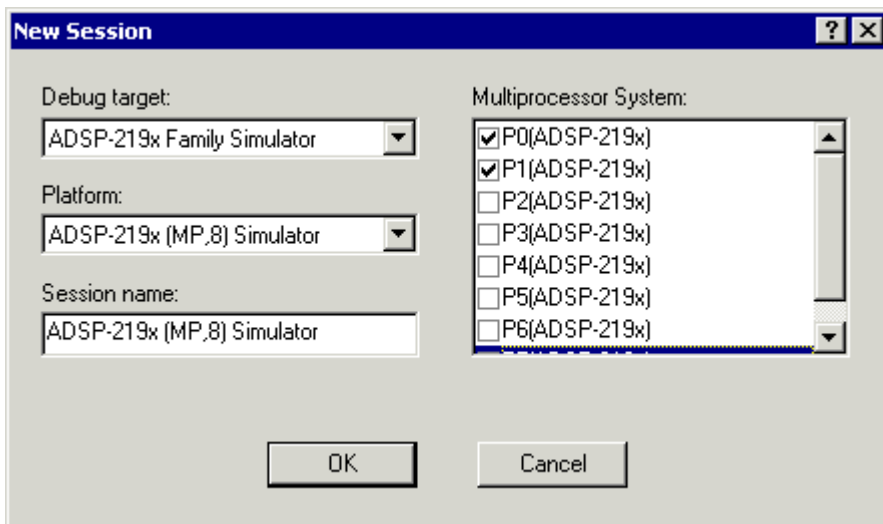


Figure 2-38. New Session Dialog Box

Exercise Five: Multiprocessor Debugging

Create a new multiprocessor simulator session by specifying the values listed in [Table 2-4](#).

Table 2-4. Setting up a Multiprocessor Simulator Session

Box	Value
Debug Target	ADSP-219x Family Simulator
Platform	ADSP-219x [MP;8] Simulator
Session Name	ADSP-219x [MP;8] Simulator

3. Under **Multiprocessor System**, select the check boxes next to processors **P0** and **P1**, as shown in [Figure 2-38 on page 2-51](#).

The check marks ($\Rightarrow$) indicate that these processors belong to the default multiprocessor group created when VisualDSP++ attaches to the session. Multiprocessor groups are described in greater detail later in this exercise.

4. Click **OK** to create the session.

VisualDSP++ closes the current session and attaches to the new multiprocessor session specified above.

Figure 2-39 shows the windows in the new multiprocessor session.

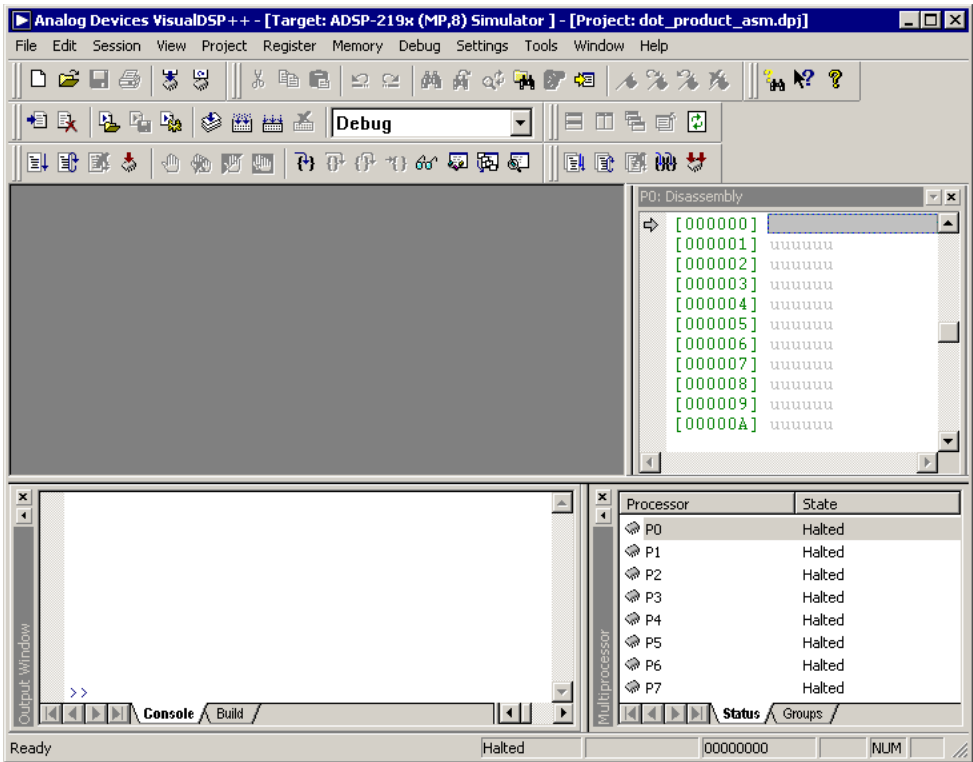


Figure 2-39. VisualDSP++ Windows in a Multiprocessor Session

Exercise Five: Multiprocessor Debugging

5. Take a moment to examine the session windows. Except for a few minor changes, notice that a multiprocessor session looks nearly identical to a single-processor session. The main differences are the:
 - **Multiprocessor** window, with the tabs **Status** and **Groups**
 - Name of the processor (such as **P0**) in the title bar of each debug window displaying processor information
 - Multiprocessor toolbar buttons to run five MP commands

The **Multiprocessor** window appears in the lower-right corner of the VisualDSP++ main window, as shown in [Figure 2-40](#).

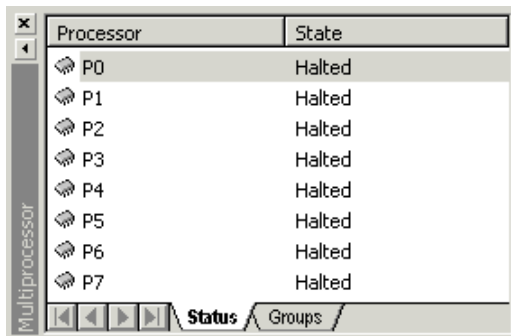


Figure 2-40. Multiprocessor Window: Status Page

The **Status** page lists each processor in the session and its current status: **Running**, **Halted**, **Stepping**, or **Unknown**.

The title bar of each debug window (such as **Disassembly**, **Memory**, or **Register**) includes the name of the processor from which information has been collected.

For example, the title bar of the **Disassembly** window shown in [Figure 2-41](#) on [page 2-55](#) indicates that information was collected from processor **P0**.

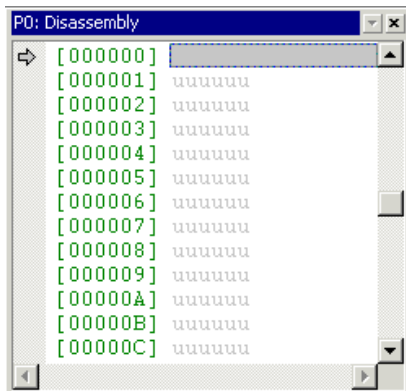


Figure 2-41. Disassembled Instructions in Processor P0

The **Multiprocessor** toolbar, shown in [Figure 2-42](#), provides access to common multiprocessor commands.

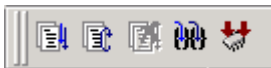


Figure 2-42. Multiprocessor Toolbar

The buttons on this toolbar provide easy access to the following commands.

-  **Multiprocessor Run**
-  **Multiprocessor Restart**
-  **Multiprocessor Halt**
-  **Multiprocessor Step**
-  **Multiprocessor Reset**

You can also access these commands from the **Debug** menu by choosing **Multiprocessor**.

Exercise Five: Multiprocessor Debugging

You are now ready to change focus and pin windows.

Step 2: Changing Focus and Pinning Windows

You can easily view data from any processor in the current session. Displaying data from a different processor in the same window is called *changing focus*. On the **Multiprocessor** window's **Status** page ([Figure 2-40 on page 2-54](#)), the first processor listed, **P0**, is highlighted and currently has focus by default.

During a debug session, you may want a debugger window to display the data from only one particular processor and to maintain that focus as new processors are selected. This feature is known as *pinning a window*.

To change focus and pin windows:

1. On the **Multiprocessor** window's **Status** page ([Figure 2-40 on page 2-54](#)), click **P1**.

Notice that the title bar of the **Disassembly** window changes to “**P1: Disassembly**” to indicate that this window is now focused on processor P1. The window displays P1's disassembled instructions.

2. Click on the text field to the right of address 000007 in the **Disassembly** window to highlight the field in gray.
3. Type the instruction `ax0=0x1234` and press **Enter**. The instruction is written into P1's memory, and the **Disassembly** window is updated as shown in [Figure 2-43 on page 2-57](#) to reflect this new instruction.

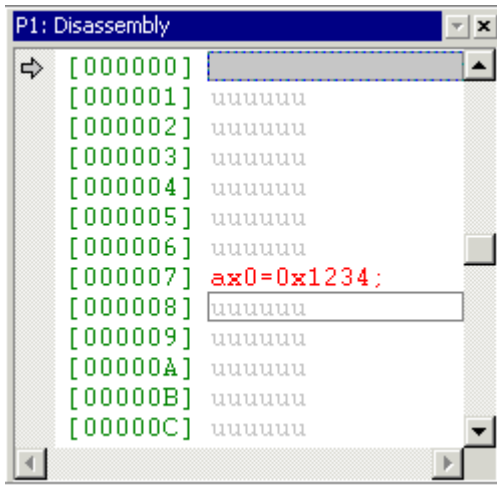


Figure 2-43. Disassembly Window: Patching a New Instruction into Processor P1's Memory

Open a second **Disassembly** window from the **View** menu by choosing **Debug Windows** and **Disassembly**.

Two **Disassembly** windows focus on processor P1, as shown in [Figure 2-44 on page 2-58](#).

Exercise Five: Multiprocessor Debugging

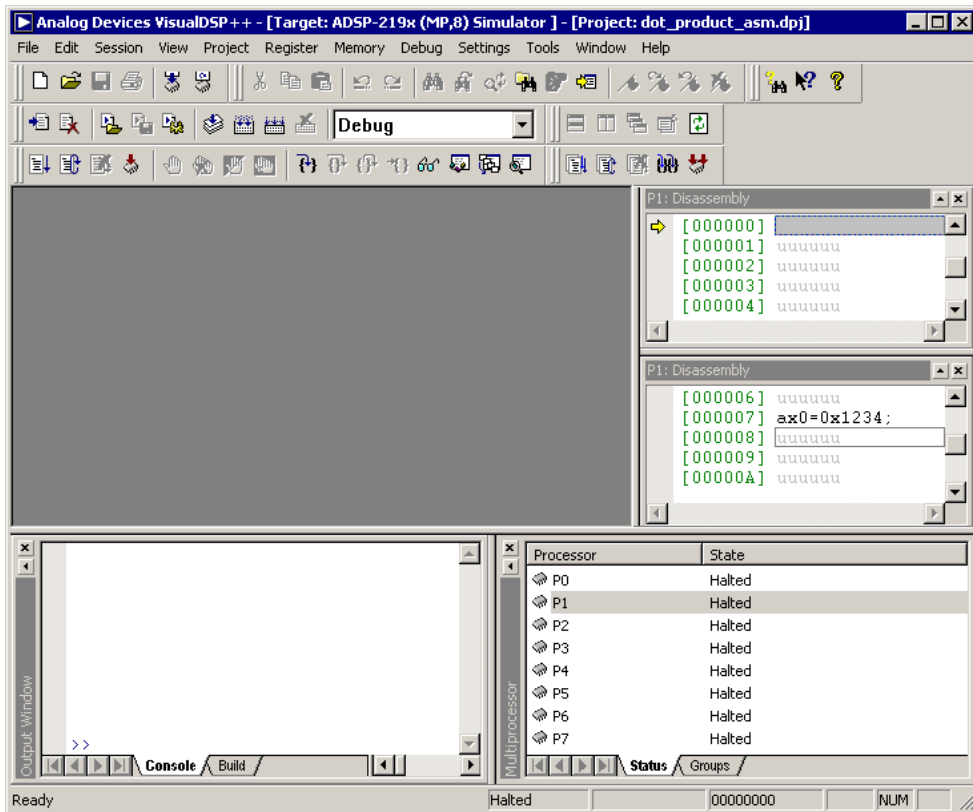


Figure 2-44. Two Open Disassembly Windows Focused on Processor P1

4. Select different processors from the **Multiprocessor** window's **Status** page (Figure 2-40 on page 2-54) and notice that both **Disassembly** windows change focus to the currently selected processor. Also note that the `ax0=0x1234;` instruction is present only when the **Disassembly** windows are focused on P1.
5. Select processor **P0**.

- Right-click the mouse in the top **Disassembly** window and choose **Pin to Processor** from the popup menu, shown in [Figure 2-45](#).

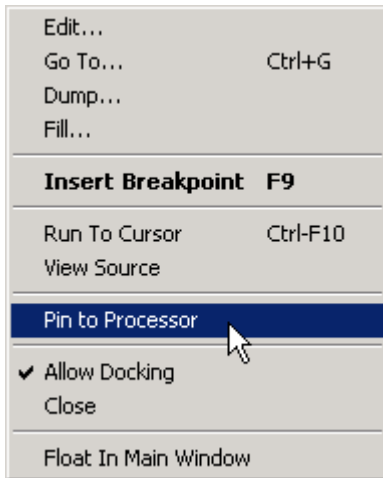


Figure 2-45. Selecting Pin to Processor

A small pin icon  appears in the title bar of the **Disassembly** window. This icon indicates that the window is now “locked” onto processor P0, and will always display data from P0, regardless of the currently focused processor.

- On the **Multiprocessor** window’s **Status** page ([Figure 2-40 on page 2-54](#)), select processor **P1**. Notice that the bottom **Disassembly** window changes focus to processor P1 and that the top **Disassembly** window stays focused on processor P0.

[Figure 2-46 on page 2-60](#) shows the top (P0) and bottom (P1) **Disassembly** windows.

Exercise Five: Multiprocessor Debugging

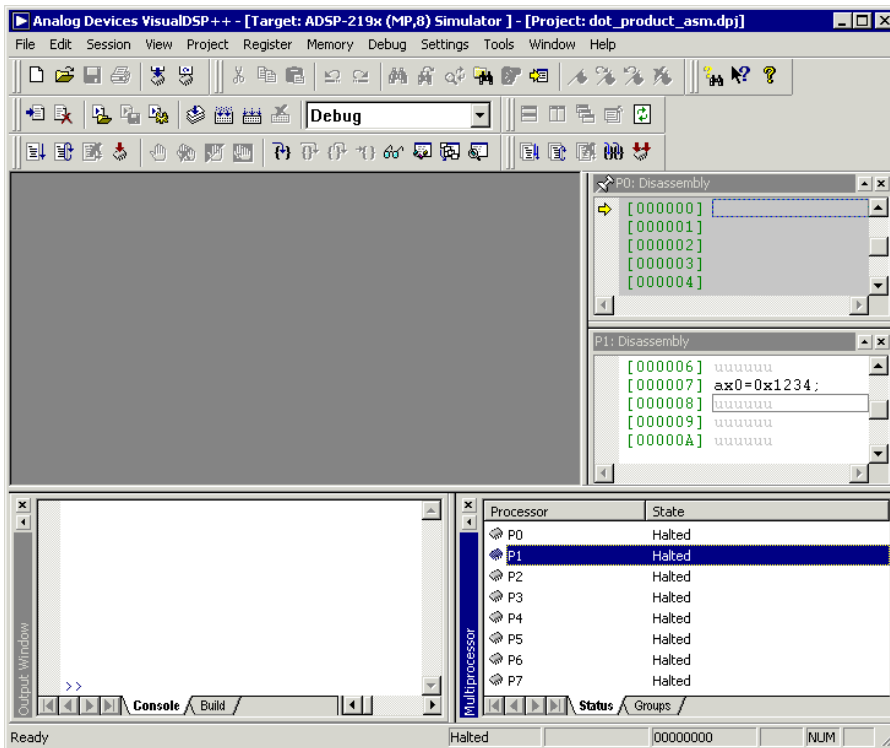


Figure 2-46. Pinning a Disassembly Window

The top **Disassembly** window is shaded gray to indicate that it is not displaying data from the currently focused processor. When many windows are open simultaneously during a debug session, shading helps you see which windows are focused on the current processor.

8. Right-click in the bottom **Disassembly** window and choose **Pin to Processor** from the popup menu.

The bottom **Disassembly** window is now pinned to P1.

You are now ready to load programs in a multiprocessor session.

Step 3: Loading Programs in a Multiprocessor Session

To load programs to all the processors in a multiprocessor session:

1. On the **Multiprocessor** window's **Status** page ([Figure 2-40 on page 2-54](#)), select P0 to set focus to processor P0.
2. Click the **Load Program** button , or from the **File** menu, choose **Load Program**. The **Open a Processor Program** dialog box appears.
3. Open your **Analog Devices** folder and double-click the following sub-folders in succession.

VisualDSP\219x\examples\tutorial\dot_product_c\debug

4. Double-click `dotprodc.dxe` to open the **Load Multiprocessor Confirmation** dialog box, shown in [Figure 2-47](#).

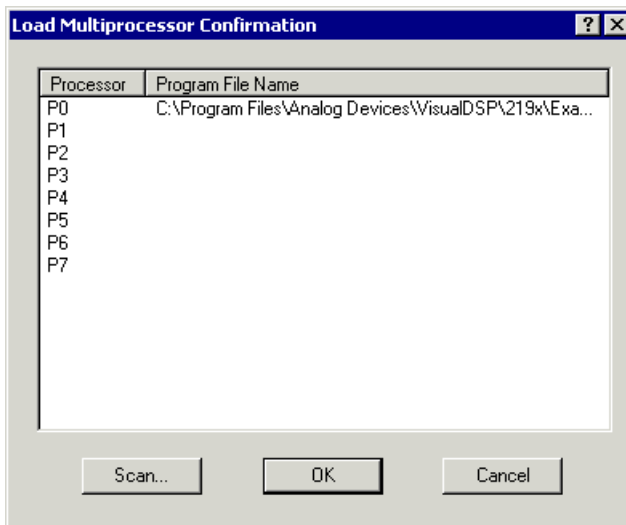


Figure 2-47. Specifying Load `dotprodc.dxe` into Processor P0

Exercise Five: Multiprocessor Debugging

Since P0 is the currently selected processor, the path to `dotprodc.dxe` is added to the P0 row.

5. Click the left mouse button in the **Program File Name** column next to P1.

An edit box appears to enable you to enter the name of the program to load into processor P1.

6. Load the `dot_product_asm.dxe` program as follows.
 - a. Click the **Browse** button  to open the **Select a Processor Program** dialog box, which enables you to browse for the program to load.
 - b. Click the up-one-level button  until you locate the `dot_product_asm` folder. Then double-click the `dot_product_asm` and `debug` folders in succession.
 - c. Double-click `dot_product_asm.dxe` to place the path to this program into the P1 row of the **Load Multiprocessor Confirmation** dialog box (see [Figure 2-48 on page 2-63](#)).

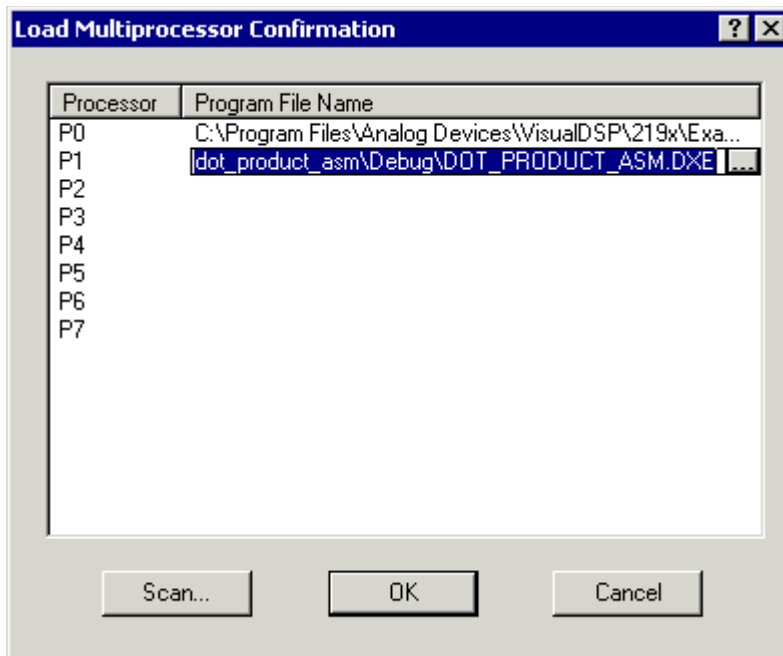


Figure 2-48. Specifying Load dot_product_asm.dxe into Processor P1

- d. Click **OK** to load the programs. When you see the warning about loading a 2191 program into a 219x session, click **Yes** to continue the loading process.

After you load each program into its respective processor, both processors run to the first executable line in `main()` and open the source files for each program.

7. Perform one of the following actions to rearrange the editor windows for easier viewing.
 - From the **Window** menu, choose **Tile Vertically**.
 - Click the **Tile Vertically** button .

Exercise Five: Multiprocessor Debugging

Figure 2-49 shows the editor windows tiled vertically.

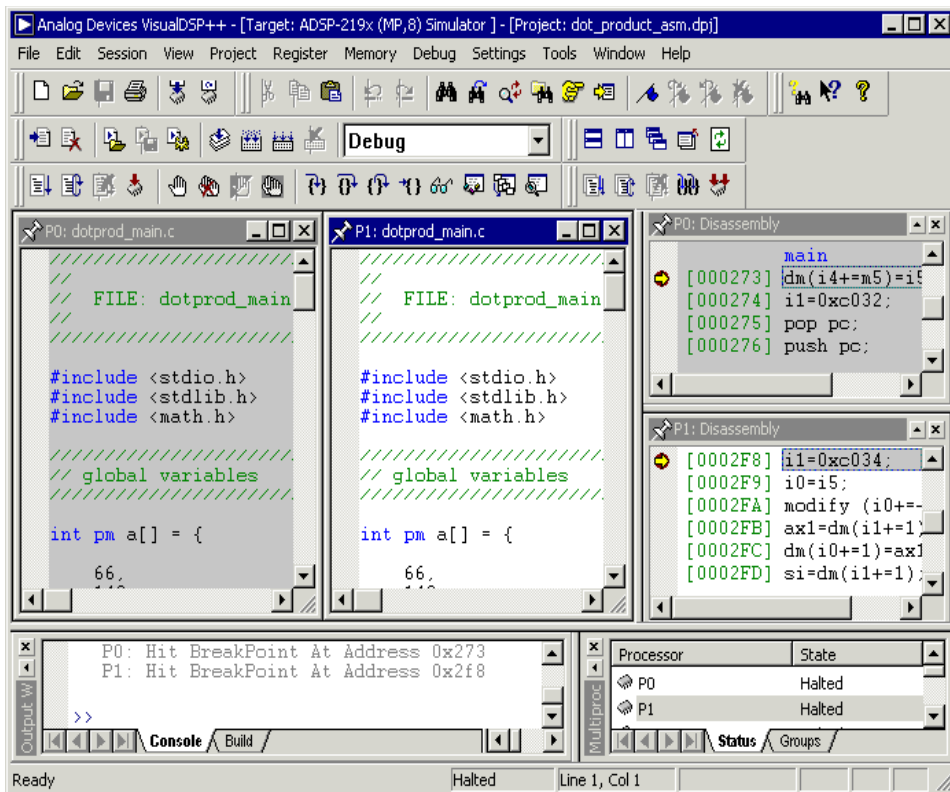


Figure 2-49. Two Programs Loaded onto Two Separate Processors

Notice that the editor windows are pinned to their respective processors.

8. Click the left mouse button in the **Disassembly** window pinned to P0. Then click the left mouse button in the bottom **Disassembly** window pinned to P1.

Notice that for all pinnable windows, including editor windows, the focus follows the currently active window. This feature facilitates navigation between processors.

Step 4: Stepping in a Multiprocessor Session

When debugging a multiprocessor session, you can still use all the standard single-processor commands such as **Run**, **Step**, **Halt**, **Reset**, and **Restart**. Each command operates only on the currently focused processor. All other processors are unaffected.

The multiprocessor commands **MP Run**, **MP Step**, and **MP Halt** operate *synchronously*. All three operations execute on exactly the same clock cycle in all processors. This feature is helpful when critical multiprocessor interaction issues arise during the development process. **MP Load**, **MP Reset**, and **MP Restart** are *not* synchronous operations. They are executed in order, one after the other.

Exercise Five: Multiprocessor Debugging

To step instructions in a multiprocessor session:

1. Set the focus to processor **P0**.
2. Step P0 two assembly instructions as follows.
 - a. Click the left mouse button in the **Disassembly** window pinned to P0.
 - b. Click the **Step Into** button  twice.

Notice that the **P0 Disassembly** window is now at address `0x275`, and the **P1 Disassembly** window is still at address `0x2F8`. You can issue multiprocessor commands to both processors at the same time.

3. Click the **Multiprocessor Step** button  twice.

Notice that both processors P0 and P1 have stepped instructions. P0 is now at address `0x277`, and P1 is at address `0x2FA`.

Step 5: Configuring and Using Multiprocessor Groups

Often, in large multiprocessor systems, individual processors are grouped into a “cluster” of processors that work together to perform a related task. Using multiprocessor groups is helpful for debugging complex systems. For example, you can send multiprocessor commands to all the processors at one time. By changing the active group, you can send debugging commands to only the processors in one particular group.

When you create a multiprocessor session, a group named “**Default**” is created. The processors that you select in the **New Session** dialog box are added to the **Default** group. (See “[Step 1: Create a Multiprocessor Simulator Session](#)” on page 2-51.) Up to this point, you have used only two of the eight processors in the multiprocessor session. The remaining processors, P2–P7, have not been used.

In this step you will complete the following tasks.

- Add processors P2–P7 to the **Default** group and issue an **MP Reset** command to all eight processors
- Create new groups and add processors to them
- Make different groups active

Adding Processors to the Default Group and Issuing MP Reset

To add processors to the **Default** group and issue an **MP Reset** command:

1. In the **Multiprocessor** window, click the **Groups** tab to display the **Groups** page, shown in [Figure 2-50](#).

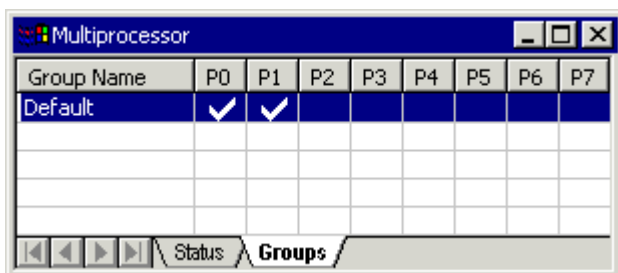


Figure 2-50. Multiprocessor Window: Groups Page

Notice that processors P0 and P1 are checked to show that they are in the **Default** group, and processors P2–P7 are not. All the MP commands (**MP Run**, **MP Step**, **MP Halt**, **MP Reset**, and **MP Restart**) are sent only to the processors in this **Default** group. The remaining processors are unaffected by these commands.

2. Right-click in the **Multiprocessor Group** window.

Exercise Five: Multiprocessor Debugging

3. Choose **Select All Processors** from the popup menu, shown in [Figure 2-51](#).

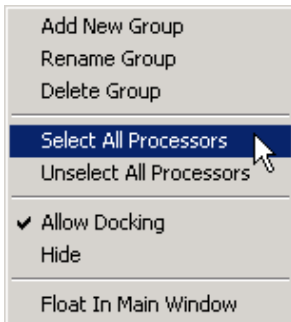


Figure 2-51. Moving All Processors into the Default Group

In the **Multiprocessor Group** window, a check mark indicates processors in the **Default** group (see [Figure 2-52](#)).

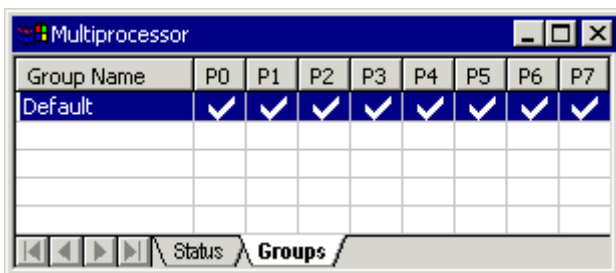


Figure 2-52. All Processors Moved into the Default group

4. Issue an **MP Reset** command to all eight processors by performing one of these actions:
 - Click the **Multiprocessor Reset** button .
 - From the **Debug** menu, choose **Multiprocessor** and then

choose **Reset**.

Because all eight processors are in the selected group, the reset is applied to all of them.

Creating and Configuring New Multiprocessor Groups

To create new groups and add processors to them:

1. Right-click in the **Multiprocessor Group** window.
2. Choose **Add New Group** from the popup menu.

This command creates an empty group (**Group 1**), as shown in [Figure 2-53](#). You can change the group's name. For this exercise, however, you will use the default.

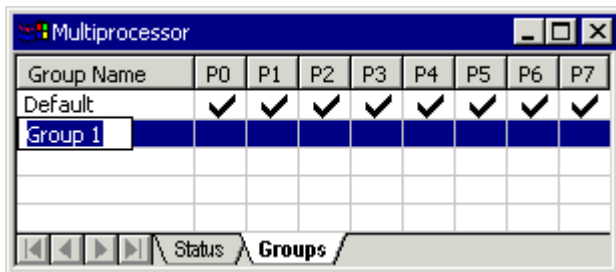


Figure 2-53. Creating a New Group

3. Press **Enter** to accept the default name, **Group 1**.
4. Place processor P0 into this new group by clicking the left mouse button in the **P0** column of the **Group 1** row.
5. Create a second new group as explained above and accept the default name **Group 2**.

Exercise Five: Multiprocessor Debugging

- Place processor P1 into this new group by clicking the left mouse button in the **P1** column of the **Group 2** row.

Figure 2-54 shows the two new multiprocessor groups.

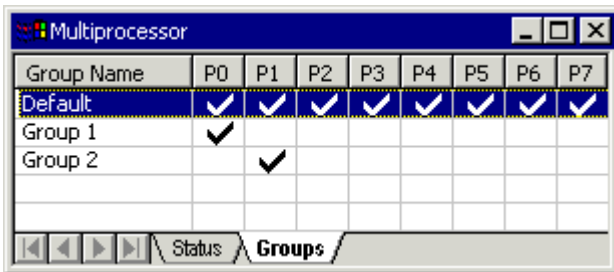


Figure 2-54. Creating Two New Multiprocessor Groups

Activating New Multiprocessor Groups

To make the new groups active:

- Make **Group 1** active by clicking the left mouse button on the **Group 1** label.
- Click the **Multiprocessor Step** button  a few times and notice that only processor P0 advances.

Because P0 is the only processor in the active group, only the P0 processor is affected by multiprocessor commands.

- Make **Group 2** active by clicking the left mouse button on the **Group 2** label.
- Click the **Multiprocessor Step** button  a few times and notice that only processor P1 advances.

You have now ready to begin Exercise Six.

Exercise Six: Installing and Using a VCSE Component

In this exercise, you will complete the following tasks.

- Start up the VisualDSP++ environment
- Open an existing project
- Install a VCSE component on your system
- Add the component to the project
- Build and run the program with the component

The sources for the exercise are in the `vcse_component` folder. The default installation path is:

```
Program files\Analog Devices\VisualDSP\219x\Examples\  
Tutorial\vcse_component
```

Step 1: Start VisualDSP++ and Open the Project

To start VisualDSP++ and open the project:

1. Click the Windows **Start** button and select **Programs, VisualDSP, and VisualDSP++ Environment**.

The VisualDSP++ main window appears.

If you have already run VisualDSP++ and the **Reload last project at startup** option is selected in the **Project Options** dialog box, VisualDSP++ opens the last project that you worked on.

To close this project, choose **Close** from the **Project** menu and then click **No** when prompted to save the project. Since you have made no changes to the project, you do not have to save it.

Exercise Six: Installing and Using a VCSE Component

2. From the **Project** menu, choose **Open**.

VisualDSP++ displays the **Open Project** dialog box.

3. In the **Look in** box, open the Program Files\Analog Devices folder and double click the following sub-folders in succession.

VisualDSP\219x\Examples\Tutorial\vcse_component

Note: This path is based on the default installation.

4. Double-click the useg711.dpj project file.

VisualDSP++ loads the project and displays messages in the **Output** window as it processes the project settings.

Note: The first time that you open projects installed from the software kit, VisualDSP++ may detect that files, folders, or both have moved. If you receive a “Project has been moved” message, click **OK** to continue.

The useg711 project contains a single C language source file useg711.c, which contains the code needed to create an instance of the CULawc component and to invoke the methods of the IG711 interface.

Step 2: Install the EXAMPLES::CULawc Component on Your System

The EXAMPLES::CULawc component is distributed as part of VisualDSP++ and is ready to be installed on your system.

1. From the **Tools** menu, select the **VCSE** submenu and then choose **Manage Components**.
2. In the **Display** field, select **Downloaded component package...** from the drop-down list.

The **Open** dialog box is displayed.

3. In the **Look in** box, open the `Program Files\Analog Devices` folder and double-click the following subfolders in succession.

`VisualDSP\219x\Examples\Tutorial\vcse_component`

Note: This path is based on the default installation.

4. Double-click the `examples_culawc_2191.vcp` file.

VisualDSP++ opens the file, extracts the information about the component, and shows it as a downloaded component in the **Component Manager** dialog box ([Figure 2-55 on page 2-74](#)).

Exercise Six: Installing and Using a VCSE Component

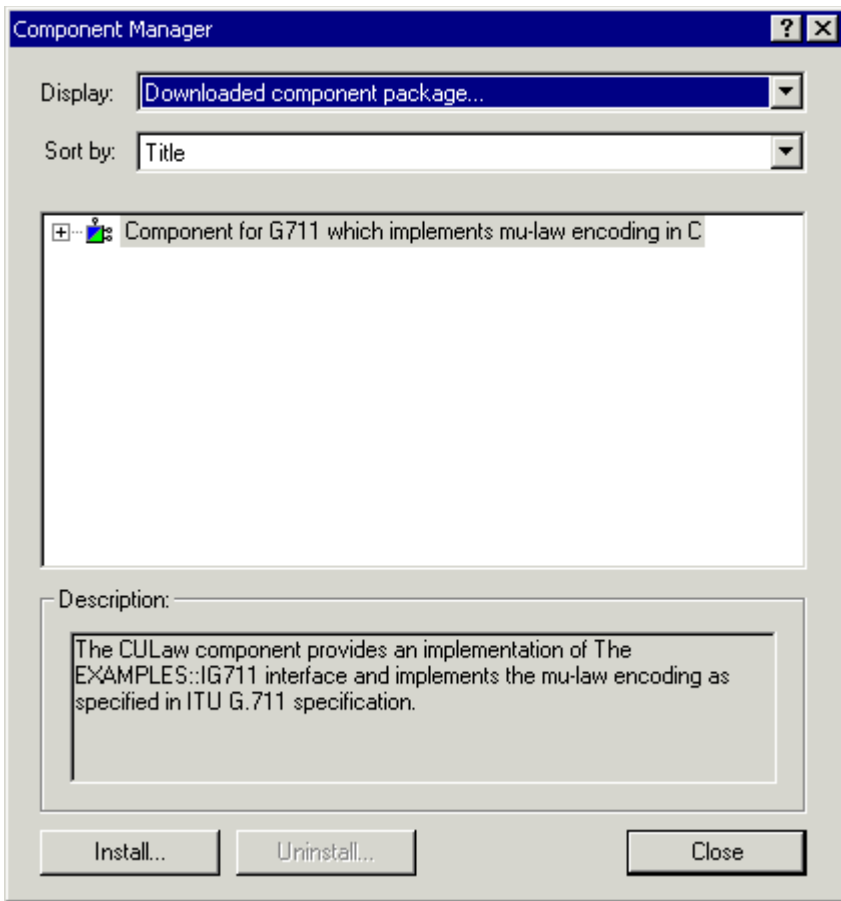


Figure 2-55. Component Manager Dialog Box – Downloaded Component

5. Click the **Install...** button to install the component on your system. Once the component is installed, click **OK**.

6. In the **Display** field, select **Locally installed components** from the drop-down list, and in the **Sort by** field, select **Title**.

Select **Component for G711 which implements the mu-law encoding in C**. Component Manager displays the dialog box shown in [Figure 2-56](#).

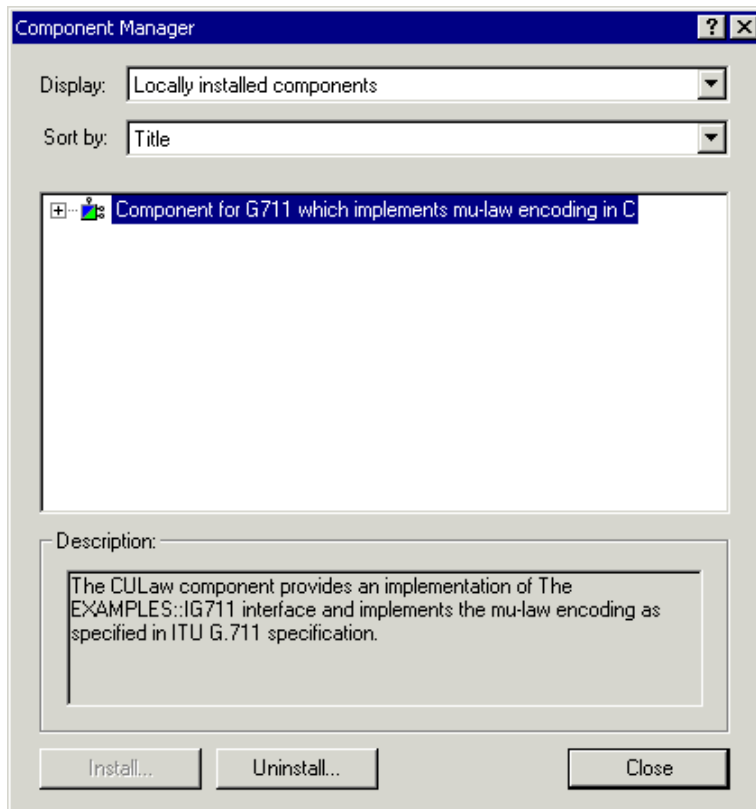


Figure 2-56. Component Manager Dialog Box – Selected Component

7. Click **Close** to close Component Manager.

Exercise Six: Installing and Using a VCSE Component

Step 3: Add the Component to Your Project

To add the newly installed component to the project:

1. From the **Tools** menu, select the VCSE submenu and then choose **Add Component**.
2. Click **Component for G711 which implements the mu-law encoding in C** to select it.
3. Click **Add** to indicate that you want to add the component to the project.

Component Manager displays the dialog box shown in [Figure 2-57](#).

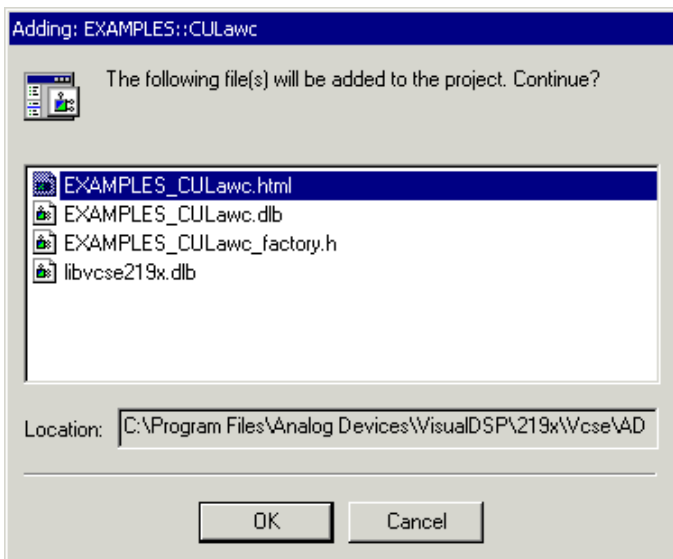


Figure 2-57. Adding Files to the Project

4. Click **OK** to add the component files to the project.

Step 4: Build and Run the Program

To build and run the program:

1. From the **Project** menu, choose **Build Project**.

VisualDSP++ displays the message shown in [Figure 2-58](#).

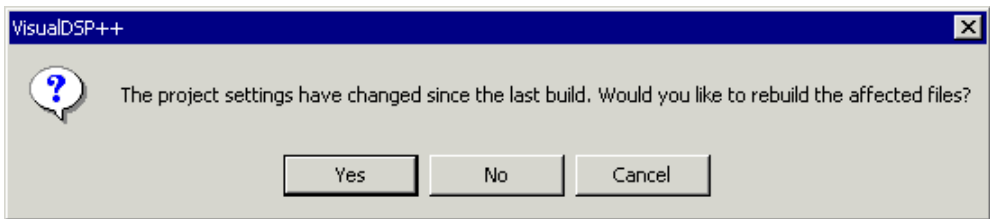


Figure 2-58. Rebuilding Files Affected by Changes to Project Settings

2. Click **Yes**. VisualDSP++ compiles the source files and creates the program.
3. From the **Debug** menu, choose **Run** to execute the program. The program generates the following output.

```
Harness to test component code generated by
EXAMPLES_CULawc.idl
Testing EXAMPLES::IG711

Test Completed result = MR_OK
```

You have now completed this tutorial. You can begin building your own project, or you can look at some of the other example programs in:

VisualDSP\219x\Examples\Simulator Peripheral Examples

These examples are documented in the appendix “Simulation of ADSP-21xx Peripherals” in the *VisualDSP++ 3.0 User’s Guide for ADSP-21xx DSPs*.

Exercise Six: Installing and Using a VCSE Component

I INDEX

Symbols

% of Histogram data, defined, [2-49](#)
/*, [2-26](#)

A

Add Files dialog box, [2-20](#)
adding project source files, [2-20](#)

B

bookmarks, adding to source files,
[2-26](#)
breakpoint symbols
 red circle, [2-13](#)
 yellow arrow, [2-13](#)
Build Project command, [2-7](#)

C

C programs
 building and running, [2-3](#)
 modifying to call as assembly
 routine, [2-16](#)
changing focus, defined, [2-56](#)
code development tools
 features, [1-9](#)
 overview, [1-2](#)

commands

 Build Project, [2-7](#)
 Rebuild All, [2-7](#)
comment characters, moving in
 source files, [2-26](#)
Compile page, [2-18](#)
configuring and using
 multiprocessor groups, [2-66](#)
Console page, [2-15](#)
conventions used in this manual, [xvi](#)
creating
 debug sessions, [2-10](#)
 new projects, [2-16](#)
customer support, [ix](#)

D

Data Cursor command, [2-42](#)
data sets
 plotting, [2-35](#)
debug features, [1-4](#)
debug sessions
 setting up new sessions, [2-10](#)
 setting up subsequent sessions,
 [2-10](#)
Default multiprocessor group, [2-66](#)
dialog boxes
 Add Files, [2-20](#)

INDEX

- Adding (VCSE files), [2-76](#)
- Breakpoints, [2-14](#)
- Component Manager (VCSE),
[2-75](#)
- Find, [2-25](#)
- New Session, [2-10](#)
- New Session (multiprocessor),
[2-52](#)
- Plot Configuration, [2-35](#)
- Project Options, [2-17](#), [2-18](#)
- Save New Project As, [2-16](#)
- Disassembly window
 - adding and deleting breakpoints,
[2-14](#)
 - information displayed, [2-13](#)
 - pinning to a processor, [2-59](#)
- dot_product
 - rebuilding, [2-32](#)
 - running, [2-33](#)
- dot_product_asm, building the
project, [2-17](#)
- dot_product_c, folder location, [2-4](#)
- dotprod_main.c, [2-8](#)
 - modifying to call a_dot_c_asm,
[2-25](#)
 - opening, [2-8](#)
- dotprodasm.ldf
 - modifying, [2-28](#)
 - viewing, [2-28](#)
- dotprodc, running, [2-15](#)
- dotprodc.dxe, automatically
loading, [2-12](#)

- E
 - editing project source files, [2-25](#)
 - editor windows, [2-9](#), [2-26](#)
 - execution units, definition of, [2-49](#)
 - exercises
 - building and running C programs,
[2-3](#)
 - installing and using a VCSE
component, [2-71](#)
 - linear profiling, [2-45](#)
 - modifying a C program to call an
assembly routine, [2-16](#)
 - multiprocessor debugging, [2-50](#)
 - plotting data, [2-33](#)
- Expert Linker, using to create a
Linker Description File, [2-21](#)

- F
 - Fast Fourier Transform (FFT), [2-44](#)
 - FFT Magnitude command, [2-43](#)
 - Find dialog box, [2-22](#), [2-25](#)
 - Finite Impulse Response (FIR)
filter, [2-33](#)
 - FIR filter, viewing the results, [2-40](#)
 - FIR program
 - global data arrays, [2-35](#)
 - running, [2-40](#)

- G
 - General page, [2-6](#)
 - Generate debug information check
box, [2-19](#)
 - Groups page (Multiprocessor
window), [2-67](#)

H

histogram, defined, [2-49](#)

I

input data sets, entering, [2-37](#)

Integrated Development and
Debugging Environment
(IDDE), [2-1](#)

J

JTAG emulators, [2-45](#)

L

linear profiling

collecting and examining data,
[2-48](#)

enabling, [2-46](#)

results of analyzing the FIR
program, [2-48](#)

viewing profile data for the FIR
function, [2-50](#)

Linear Profiling Results window
(empty), [2-47](#)

Linker Definition File (LDF)
folder, [2-2](#)

Linker Description File (LDF)
creating, [2-21](#)

linker errors

viewing in the Output window,
[2-28](#)

Load executable after build
command, [2-10](#)

M

magnifying selected regions, [2-41](#)

messages

Output window, [2-13](#)

project has been moved, [2-5](#)

project is up to date, build
completed successfully, [2-7](#)

modifying C programs to call
assembly routines, [2-16](#)

multiprocessor commands, [2-65](#)

multiprocessor debug session

changing focus, [2-56](#)

patching a new instruction, [2-57](#)

pinning Disassembly windows to
a processor, [2-59](#)

multiprocessor debugging, [2-50](#)

multiprocessor groups

activating new groups, [2-70](#)

adding processors to the Default
group, [2-67](#)

configuring and using, [2-66](#)

creating and configuring, [2-69](#)

Multiprocessor Reset command,
[2-68](#)

multiprocessor simulator sessions

creating, [2-51](#)

differences from single processor
sessions, [2-54](#)

multiprocessor toolbar buttons,
[2-55](#)

Multiprocessor window

Groups page, [2-67](#)

Status page, [2-54](#)

MyAnalog.com, [x](#)

INDEX

N

New Session dialog box, [2-10](#), [2-51](#)

O

opening

 projects, [2-4](#)

output data sets

 entering, [2-37](#)

Output window, [2-8](#)

 Console page, [2-15](#)

 information displayed, [2-13](#)

 viewing a linker error, [2-28](#)

P

Pin to Processor command, [2-59](#)

pinning windows, defined, [2-56](#)

Plot Configuration dialog box,

[2-35](#), [2-37](#)

plot windows

 after running the FIR program,

[2-40](#)

 before running the FIR program,

[2-39](#)

 magnified result, [2-41](#)

 magnifying data points, [2-41](#)

 opening, [2-35](#)

 selecting a region to magnify, [2-41](#)

 viewing data points, [2-42](#)

 viewing signals in the frequency

 domain, [2-44](#)

 zooming in on a region, [2-40](#)

plotting

 data, [2-33](#)

 specifying data sets, [2-35](#)

 preferences, selecting, [2-6](#)

 programs, running in a debug

 session, [2-10](#)

 Project Options dialog box, [2-17](#),

[2-18](#)

 Project page, [2-17](#)

 projects

 adding files to dot_product_asm,

[2-20](#)

 building dot_product, [2-32](#)

 building dotprodc, [2-7](#)

 creating a LDF, [2-21](#)

 creating new projects, [2-16](#)

 dot_product_asm, building, [2-17](#)

 dotprodc files, [2-5](#)

 managing overview, [1-1](#)

 modifying source files, [2-25](#)

 opening, [2-4](#)

 options, [2-18](#)

R

Rebuild All command, [2-7](#)

Reset Zoom command, [2-41](#)

S

Save New Project As dialog box,

[2-16](#)

Select All Processors command,

[2-68](#)

source files

 adding to projects, [2-20](#)

 modifying, [2-25](#)

statistical profiling, [2-45](#)

Status page (Multiprocessor window), [2-58](#), [2-59](#)
Stepping in a multiprocessor debug session, [2-65](#)

T

technical support, [ix](#)
toolbar buttons, [2-3](#)
tutorial overview, [2-1](#)

V

VCSE

Adding (files) dialog box, [2-76](#)
adding a VCSE component to a project, [2-76](#)
building and running the program, [2-77](#)

installing a VCSE component, [2-73](#)

View Sample Count command, [2-49](#)

VisualDSP++

features, [1-1](#)
starting, [2-4](#)
tool buttons, [2-3](#)

W

windows

Disassembly, [2-13](#), [2-14](#)
editor, [2-9](#), [2-26](#)
Linear Profiling Results, [2-46](#)
Multiprocessor, Groups page, [2-67](#)
Multiprocessor, Status page, [2-54](#)
Output, [2-8](#), [2-9](#), [2-13](#)
plot, [2-39](#), [2-40](#), [2-41](#), [2-42](#)

