

VISUALDSP++™ 3.0

Getting Started Guide

for SHARC® DSPs

Revision 4.0, January 2003

Part Number
82-001993-01

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



Copyright Information

©2003 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	xi
Online Documentation	xii
From VisualDSP++	xiii
From Windows	xiii
From the Web	xiv
Printed Manuals	xiv
VisualDSP++ Documentation Set	xiv
Hardware Manuals	xiv
Data Sheets	xv

CONTENTS

How to Contact DSP Publications	xv
Notation Conventions	xv

FEATURES AND TOOLS

VisualDSP++ Features	1-1
New VisualDSP++ Features in Release 3.0	1-5
Code Development Tools	1-8
VisualDSP++ Help System	1-10
Using the Help Window	1-10
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	1-23
Viewing Online Documents	1-24
Printing Online Documents	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: Set Up the Debug Session	2-10
Step 4: Run dotprodc	2-15
Exercise Two: Calling an Assembly Routine and Creating an LDF	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	2-33
Exercise Three: Plotting Data	2-34
Step 1: Load the Convolution Program	2-34
Step 2: Open a Plot Window	2-36

CONTENTS

Step 3: Run the Convolution Program and View the Data	2-41
Exercise Four: Linear Profiling	2-46
Step 1: Load the Convolution Program	2-46
Step 2: Open the Profiling Window	2-47
Step 3: Collect and Examine the Linear Profile Data	2-50
Exercise Five: Installing and Using a VCSE Component	2-54
Step 1: Start VisualDSP++ and Open the Project	2-55
Step 2: Install the EXAMPLES::CULawc Component	2-56
Step 3: Add the Component to Your Project	2-59
Step 4: Build and Run the Program	2-61

INDEX

PREFACE

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

Purpose of This Manual

The *VisualDSP++ 3.0 Getting Started Guide for SHARC 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 *Programming Reference* manuals that describe the 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 SHARC DSPs* includes procedures for:

- Using the Expert Linker 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 SHARC processors.

- ADSP-21020
- ADSP-21060
- ADSP-21060L
- ADSP-21061
- ADSP-21061L
- ADSP-21062

Product Information

- ADSP-21065L
- ADSP-21062L
- ADSP-21160M, ADSP-21160N
- ADSP-21161N

Product Information

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

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

MyAnalog.com

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

Registration:

Visit www.myanalog.com to sign up. Click **Register** to use MyAnalog.com. Registration takes about five minutes and serves as means 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 e-mail address.

DSP Product Information

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

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

- E-mail questions or requests for information to dsp.support@analog.com
- Fax questions or requests for information to
1-781-461-3010 (North America)
+49 (0) 089 76 903 157 (Europe)
- Access the Digital Signal Processor 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 SHARC DSPs

VisualDSP++ 3.0 Linker and Utilities Manual for SHARC DSPs

VisualDSP++ 3.0 Assembler and Preprocessor Manual for SHARC DSPs

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

VisualDSP++ 3.0 Product Bulletin for SHARC DSPs

VisualDSP++ 3.0 Kernel (VDK) User's Guide

VisualDSP++ Component Software Engineering User's Guide

Quick Installation Card

Product Information

For hardware information, refer to your DSP's *Hardware Reference*, *Programming Reference*, *Instruction Set Reference*, and data sheet.

All documentation is available online. Most documentation is available in printed form.

Online Documentation

Online documentation comprises Microsoft HTML Help (.CHM), Adobe Portable Documentation Format (.PDF), and HTML (.HTM and .HTML) files. A description of each file type is as follows.

File	Description
.CHM	VisualDSP++ online Help system files and VisualDSP++ manuals are provided in Microsoft HTML Help format. Installing VisualDSP++ automatically copies these files to the <code>VisualDSP\Help</code> folder. Online Help is ideal for searching the entire tools manual set. Invoke Help from the VisualDSP++ Help menu or via the Windows Start button.
.PDF	Manuals and data sheets in Portable Documentation Format are located in the installation CD's <code>Docs</code> folder. Viewing and printing a .PDF file requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher). Running <code>setup.exe</code> on the installation CD provides easy access to these documents. You can also copy .PDF files from the installation CD onto another disk.
.HTM or .HTML	Dinkum Abridged C++ library and FlexLM network license manager software documentation is located on the installation CD in the <code>Docs\Reference</code> folder. Viewing or printing these files requires a browser, such as Internet Explorer 4.0 (or higher). You can copy these files from the installation CD onto another disk.

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

From VisualDSP++

VisualDSP++ provides access to online Help. It does not provide access to .PDF files or the supplemental reference documentation (Dinkum Abridged C++ library and FlexLM network licence). Access Help by:

- Choosing **Contents**, **Search**, or **Index** from the VisualDSP++ **Help** menu
- Invoking context-sensitive Help on a user interface item (toolbar button, menu command, or window)

From Windows

In addition to shortcuts you may construct, Windows provides many ways to open VisualDSP++ online Help or the supplementary documentation.

Help system files (.CHM) are located in the `VisualDSP\Help` folder. Manuals and data sheets in PDF format are located in the `Docs` folder of the installation CD. The installation CD also contains the Dinkum Abridged C++ library and FlexLM network license manager software documentation in the `\Reference` folder.

To use Windows Explorer:

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

Product Information

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

Printed copies of 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

Printed copies of hardware manuals can be ordered through the Literature Center or downloaded from the Analog Devices Web site. The phone number is 1-800-ANALOGD (1-800-262-5643). The manuals can be ordered by title or by product number (located on the back cover of each manual).

Data Sheets

All data sheets can be downloaded from the Analog Devices Web site. As a general rule, printed copies of data sheets with a letter suffix (L, M, N, S) can be obtained from the Literature Center at **1-800-ANALOGD** (**1-800-262-5643**) or downloaded from the Web site. Data sheets without the suffix can be downloaded from the Web site only—no hard copies are available. You can ask for the data sheet by part name or by product number.

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

How to Contact DSP Publications

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

- E-mailing 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.



Code has been formatted to fit this manual's page width.

Notation Conventions

Example	Description
Close command (File menu) or OK	Text in bold style indicates the location of an item within the VisualDSP++ environment's menu system and user interface items.
{this that}	Alternative required items in syntax descriptions appear within curly brackets 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, code examples, and feature names are in text with <i>letter gothic</i> font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
 Note:	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.
 Caution:	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 1-8
- “VisualDSP++ Help System” on page 1-10

VisualDSP++ Features

VisualDSP++ provides the following features.

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

VisualDSP++ Features

- **Easy access to code development tools.** You can use the following code development tools: C/C++ compiler, VIDL compiler, assembler, linker, and loader. You 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. You can define options for a single file or for an entire project. You define these options once. You can then modify them as necessary.
- **Flexible project build options.** You can 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. You can view the status of your project build in progress. If the build reports an error, you double-click on the file name in the error message to open that source file. You can then correct the error, rebuild the file or project, and start a debug session.
- **VisualDSP++ Kernel (VDK) Support.** You can add VDK support to a project to structure and scale application development. The **Kernel** view in the **Project** window enables you to manipulate kernel objects such as events, event bits, priorities, semaphores, and thread types.
- **Flexible workspace management.** You can create multiple workspaces and quickly switch between them.
- **Easy movement between debug and build activities.** Once you start the debug session, you can move freely between editing, build, and debug activities.

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

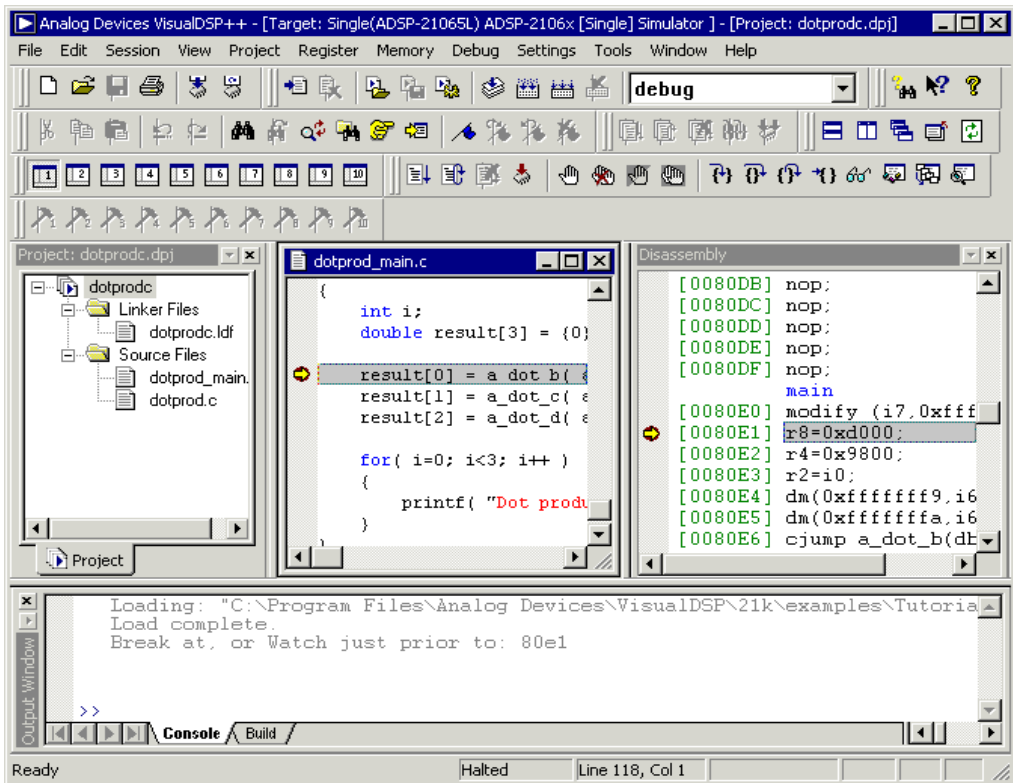


Figure 1-1. The VisualDSP++ IDDE

VisualDSP++ Features

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

- **Easy-to-use debugging activities.** You can debug with one common, easy-to-use interface for all processor simulators and emulators, or hardware evaluation and development boards. You can switch easily between these targets.
- **Multiple language support.** You can debug programs written in C, C++, 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.** You can set breakpoints on symbols and addresses and then step through the program's execution to find problems in coding logic. You can set watchpoints (conditional breakpoints) on registers, stacks, and memory locations to identify when they are accessed.
- **Tools for improving performance.** You can use the trace and linear and statistical profiles to identify bottlenecks in your DSP application and to identify program optimization needs. You can use plotting to view data arrays graphically. You can generate interrupts as well as inputs and outputs (data streams) to simulate real-world application conditions.
- **Multiprocessor debugging.** 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. 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 replaces the legacy profiler.

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).

- **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.

New VisualDSP++ Features in Release 3.0

- **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, and a right-click command that opens header files from the filename in an `#include` statement.
- **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 graphical interface to on-board flash memory devices. Use the Flash Programmer to view various flash memory information, fill or erase portions of flash memory, or load an Intel hex formatted file into flash memory. Analog Devices supplies flash drivers (algorithms) for many EZ-KIT Lite™ evaluation systems. Additionally, the Flash Programmer enables you to create custom flash drivers or modify existing drivers for use on custom boards with a variety of processor and flash device combinations.
- **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.

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.

Code Development Tools

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.

Code Development Tools

The SHARC code development tools include:

- C/C++ compiler
- VIDL compiler
- Runtime library with over 100 math, DSP, and C runtime library routines
- Assembler
- Linker
- Splitter
- Loader

- Simulator
- Emulator (must be purchased separately from VisualDSP++)

These tools support the ADSP-21xxx Family of DSPs and enable you to develop applications that take full advantage of the SHARC architecture. VisualDSP++ includes a linker that supports multiprocessing, shared memory, and memory overlays.

The code development tools provide the following key features.

- **Easy-to-program C, C++, and assembly languages.** You can program in C, C++, or 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.** You can define multiple types of executables for a single type of DSP in one Linker Description File (.LDF). You can specify input files, including objects, libraries, shared memory files, overlay files, and executables.
- **Support for overlays, multiprocessors, and shared memory executables.** 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. You can 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 SHARC DSPs*.

VisualDSP++ Help System

The VisualDSP++ Help system 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.

VisualDSP++ Help is a merge of several Help systems (.CHM files). Each is identified with a book icon  in your product installation's **Help** folder.

Most of the Help system comprises VisualDSP++ tools *manuals*, such as the Assembler and Preprocessor manuals. These manuals are also provided in PDF format (on installation disk) for printing and are available from Analog Devices as printed books.

Some .CHM files support *pop-up* messages for dialog box controls (buttons, fields, and so on). These messages, which appear in little yellow boxes, compose part of the context-sensitive Help in VisualDSP++.

The Help system describes the VisualDSP++ *user interface*. Help files include concepts, procedures, and reference information. Each toolbar button, menu-bar command, and debugging window in VisualDSP++ is linked to a topic in one of these files.

Using the Help Window

The Help window comprises three parts:

- The *Navigation pane* provides tabbed pages (**Contents**, **Index**, **Search**, and **Favorites**) that show different navigational views.
- The *Viewing pane* displays the selected object (topic, Web page, video, .PDF file, application).
- *Toolbar buttons* enable you to navigate or specify options.

Figure 1-2 shows the parts of the VisualDSP++ Help window.

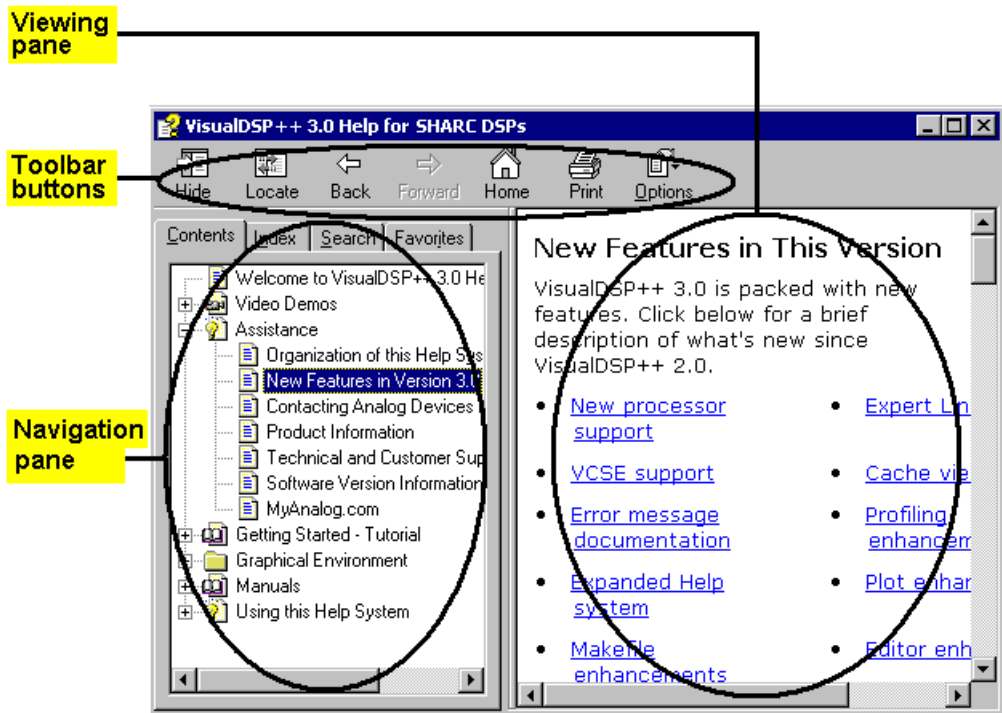


Figure 1-2. Parts of the VisualDSP++ Help Window

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 **C**ontents, **S**earch, or **I**ndex from the **H**elp 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.

By default, the VisualDSP++ software installation procedure places the complete set of Help files (except the *Getting Started Guide*) in the folder VisualDSP\Help.

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. Write down 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++.

Context-sensitive Help in VisualDSP++ 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](#).

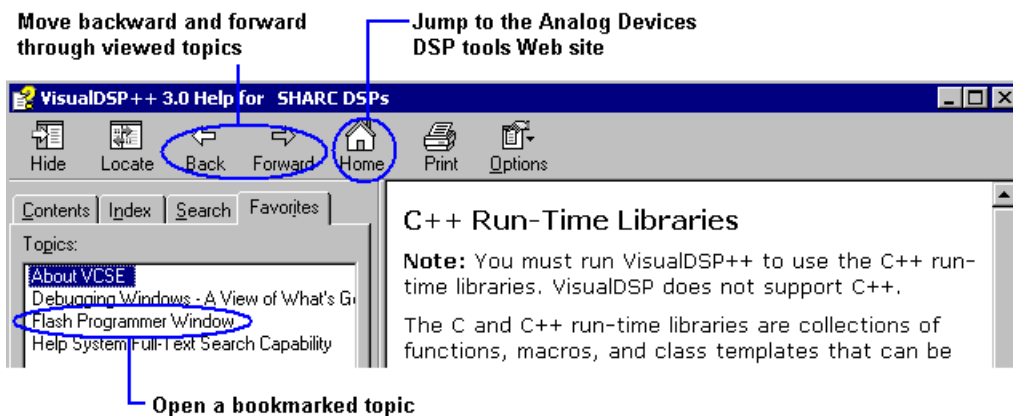


Figure 1-3. Help Window's Navigational Aids

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 on the Contents 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 is completed.

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 page: Right-click the topic  and choose Print .
Entire section of Help	On the Contents page: 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** page, 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 page ([Figure 1-4](#))

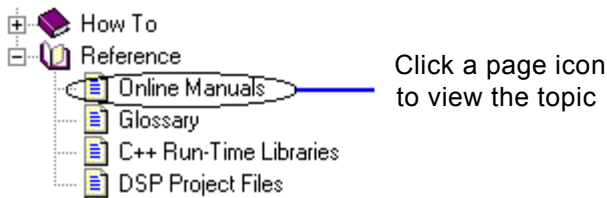


Figure 1-4. Contents Page – Online Manuals Topic

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

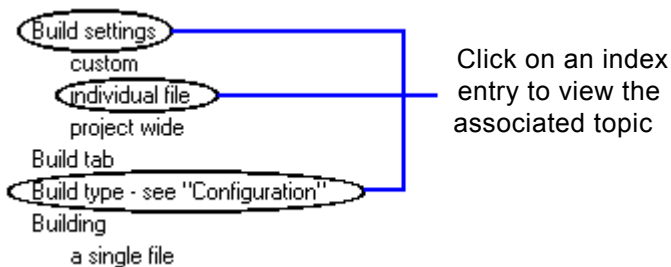


Figure 1-5. Index Entries on the Index Page

- 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 that uses the “AND” operator.

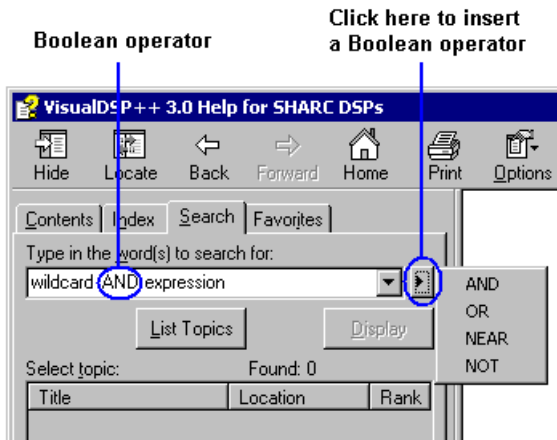


Figure 1-6. Boolean Search with the “AND” Operator

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.

- 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	<code>new AND plot</code>	Locates topics that contain both the words “new” and “plot”
Either term in a topic	<code>new OR plot</code>	Locates topics that contain either the word “new” or the word “plot” or both
The first term without the second term	<code>new NOT plot</code>	Locates topics that contain the word “new”, but not the word “plot”
Both terms in the same topic, close together	<code>new NEAR plot</code>	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 Documents

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

Table 1-5. Types of User Documentation

Files	Purpose
.CHM	VisualDSP++ online Help system files and VisualDSP++ manuals are provided in Microsoft HTML Help format. Installing VisualDSP++ automatically copies these files to the <code>VisualDSP\Help</code> folder. Online Help is ideal for searching the entire tools manual set. Invoke Help from the VisualDSP++ Help menu or via the Windows Start button. The .CHM files require Internet Explorer 4.0 (or higher) or the installation of a component that provides a .CHM file viewer.
.PDF	Manuals and data sheets in Portable Documentation Format are located in the installation CD's <code>Docs</code> folder. Viewing and printing a .PDF file requires a PDF reader, such as Adobe Acrobat Reader (4.0 or higher). Running <code>setup.exe</code> on the installation CD provides easy access to these documents. You can also copy PDF files from the installation CD onto another disk.
.HTM or .HTML	Dinkum Abridged C++ library and FlexLM network license manager software documentation is located on the installation CD in the <code>Docs\Reference</code> folder. Viewing or printing these files requires a browser, such as Internet Explorer 4.0 (or higher). You can copy these files from the installation CD onto another disk.



The VisualDSP++ software installation procedure does not copy PDF versions of books and data sheets or supplemental reference documentation to the VisualDSP directory.

Printing Online Documents

You can print documents from the VisualDSP++ Tools Installation CD-ROM.

To print online documents:

1. Insert the VisualDSP++ Tools Installation CD-ROM in your CD-ROM drive.
2. Open the **Docs** folder by using one of these options:
 - From the **VisualDSP++ Tools Installation** main menu, click **View Documentation**. (If the main menu does not appear, run `setup.exe`.)
 - In Windows Explorer, select your CD-ROM drive (for example, **d:**) and open the **Docs** folder.

3. Open the folder where the document is located.

The `Data Sheets` folder contains copies of DSP data sheets.

The `Hardware Manuals` folder contains copies of hardware manuals.

The `Reference` folder includes the HTML files that comprise the Dinkum Abridged C++ library and the FlexLM network license documentation.

The `Tools Manuals` folder contains copies of VisualDSP++ tools manuals.

4. Double-click the document that you want to print. Selecting a PDF file opens Adobe Acrobat Reader and displays the document. Selecting an HTML file opens a browser and displays the document.
5. From the **File** menu, choose **Print** and 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: Calling an Assembly Routine and Creating an LDF” on page 2-16](#)
- [“Exercise Three: Plotting Data” on page 2-34](#)
- [“Exercise Four: Linear Profiling” on page 2-46](#)
- [“Exercise Five: Installing and Using a VCSE Component” on page 2-54](#)

Overview

This tutorial demonstrates some of the 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-21xxx DSPs. For these exercises, you will use the ADSP-2106x simulator for the ADSP-21065L target.

You can use a different ADSP-21xxx processor with only minor changes to the Linker Description File (.LDF) included with each project.

Overview

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

```
Analog Devices\VisualDSP\21k\ldf
```

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

The tutorial contains five exercises.

- In **Exercise One**, you will start up VisualDSP++, build a project containing C source code, set up a debug session, and run the program.
- 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, and rebuild the project.
- In **Exercise Three**, you will apply a simple convolution algorithm to a buffer of data. You will use the VisualDSP++ plotting engine to view the different data arrays graphically.
- In **Exercise Four**, you will use linear profiling to examine the efficiency of the convolution 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 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.

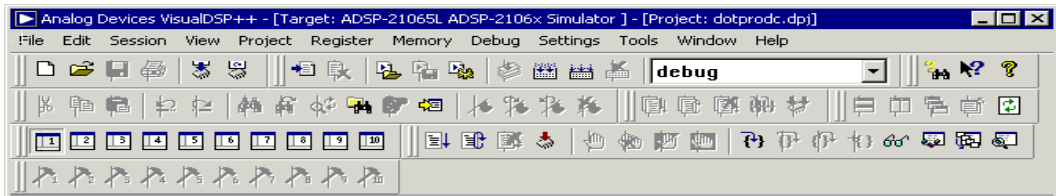


Figure 2-1. VisualDSP++ Toolbar Buttons

Exercise One: Building and Running a C Program

In this exercise, you will:

- Start up the VisualDSP++ environment
- Open and build an existing project
- Set up the debug session and examine windows and dialog boxes
- Run the program

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

```
Program Files\Analog Devices\VisualDSP\21k\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.

- a. Select the values shown in [Table 2-1](#).

Table 2-1. Session Specification

Box	Value
Debug Target	ADSP-2106x Family Simulator
Platform	ADSP-2106x Simulator
Session Name	ADSP-21065L ADSP-2106x Simulator
Processor	ADSP-21065L

- b. Click **OK**. The VisualDSP++ main window appears.

If you have already run VisualDSP++ and the **Reload last project at startup** option is selected on the **Project** page under **Settings and Preferences**, 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\21k\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](#).

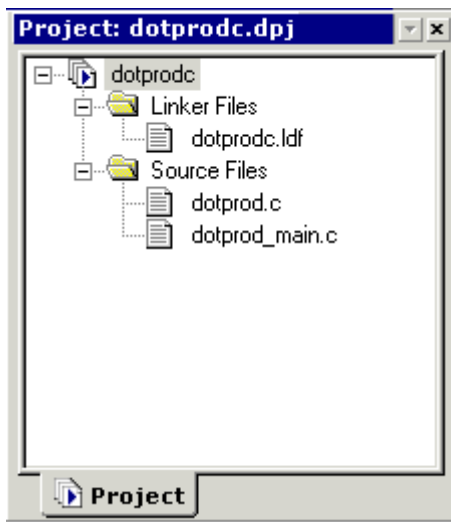


Figure 2-2. Project Loaded in the Project Window

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

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](#).

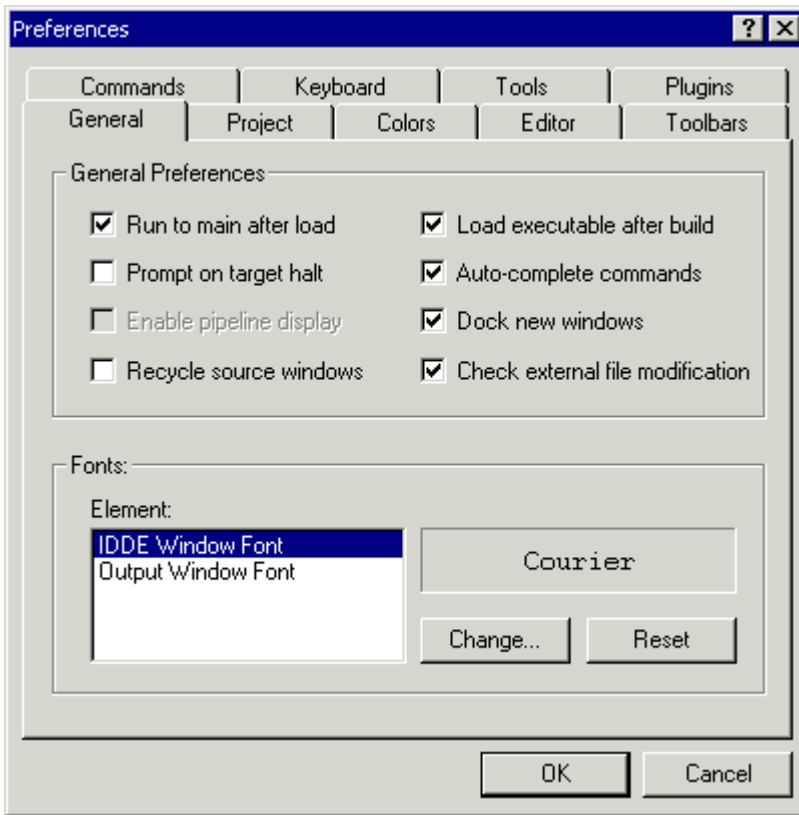


Figure 2-3. Preferences Dialog Box

- 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.

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.

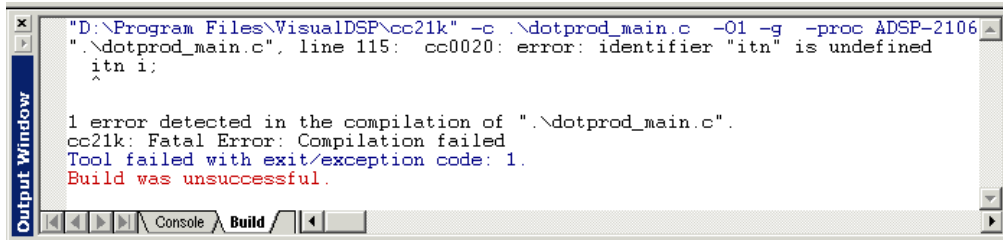


Figure 2-4. Example of Error Message

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`.

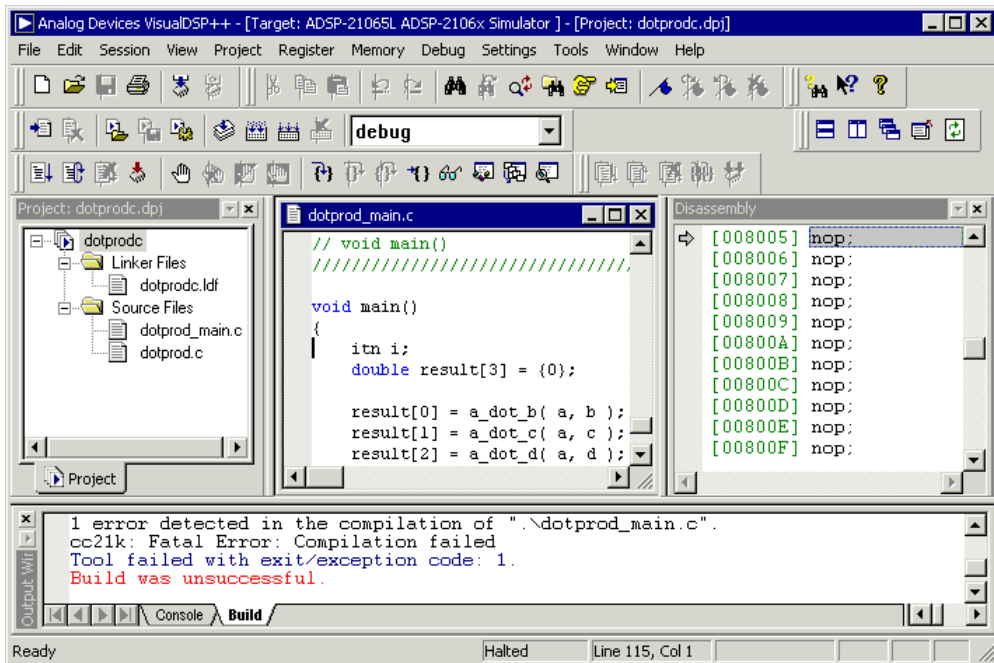


Figure 2-5. Output Window and Editor Window

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 in the **Build** view in the **Output** window.

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

Step 3: Set Up the Debug Session

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 selected processor in the debug session 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.

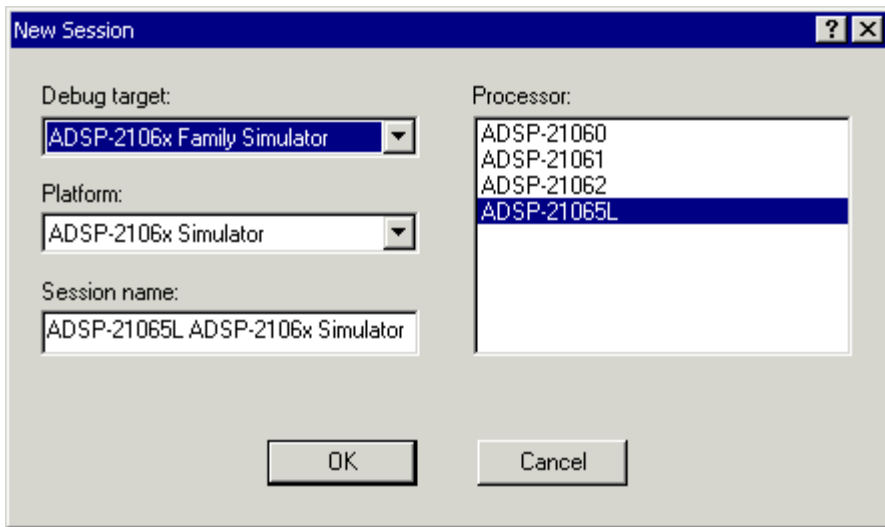


Figure 2-6. New Session Dialog Box

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

Table 2-2. Session Specification

Box	Value
Debug Target	ADSP-2106x Family Simulator
Platform	ADSP-2106x Simulator
Session Name	ADSP-21065L ADSP-2106x Simulator
Processor	ADSP-21065L

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](#)).

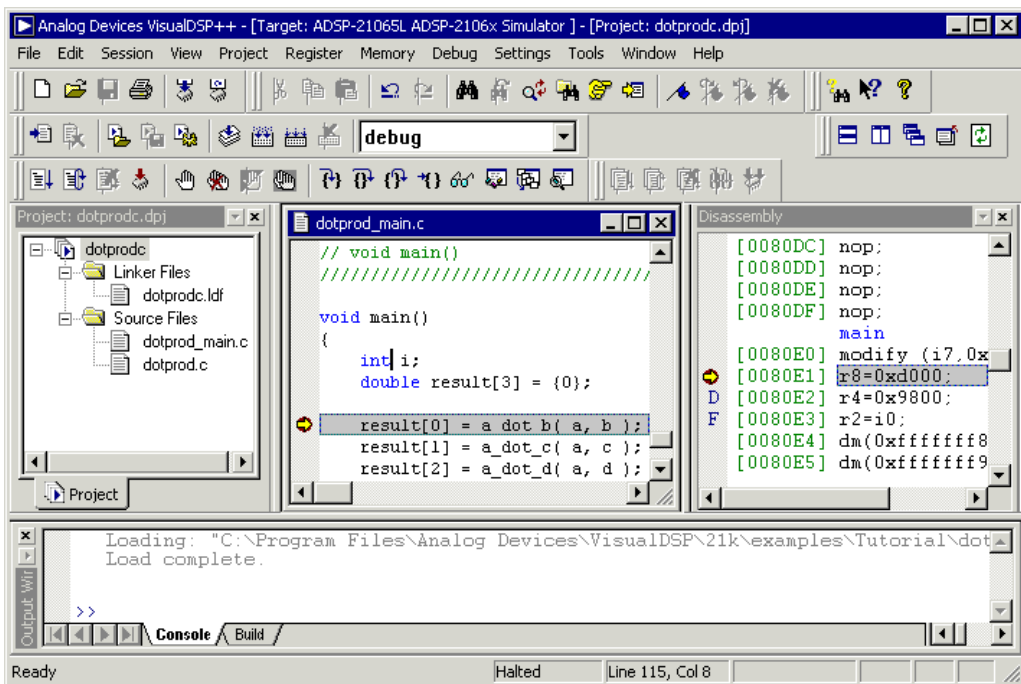


Figure 2-7. Loading dotprodc.dxe

5. Look at the information in the open windows.

The **Output** window's **Console** view 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  and a yellow arrow  appear at the start of the program labeled “main”.

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](#).

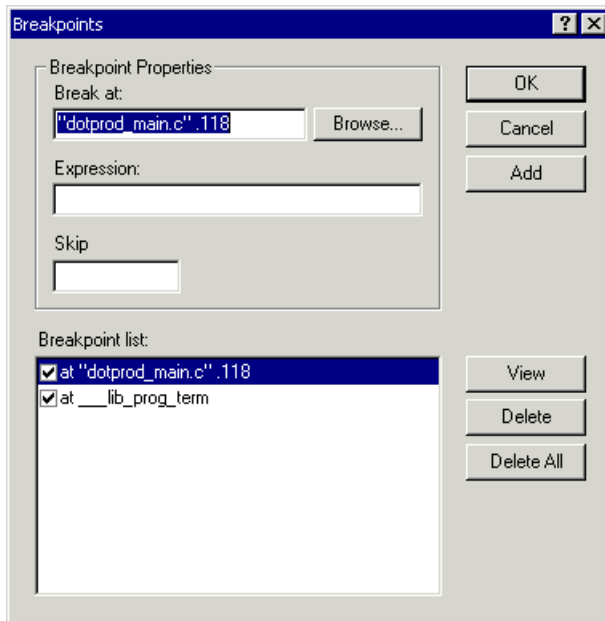


Figure 2-8. Breakpoints Dialog Box

The breakpoints are set at these C program labels:

- "dotprod_main.c" 118
- __lib_prog_term

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 in the **Console** view ([Figure 2-9](#)) in the **Output** window.

```
Dot product [0] = 0.000000  
Dot product [1] = 0.707107  
Dot product [2] = -0.500000
```

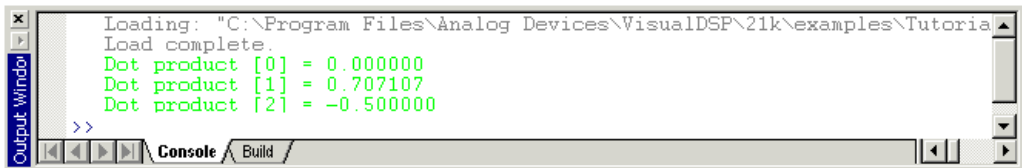


Figure 2-9. Results of the dotprodc Program

You are now ready to begin Exercise Two.

Exercise Two: Calling an Assembly Routine and Creating an LDF

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](#).

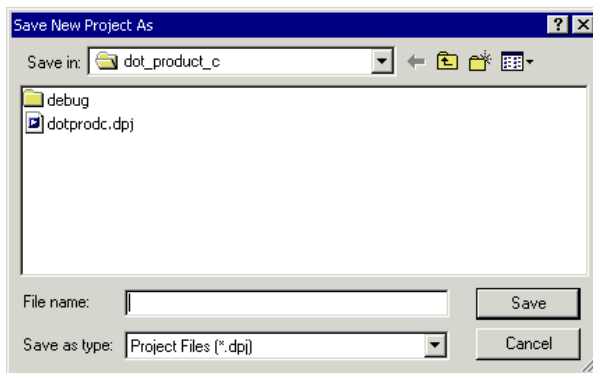


Figure 2-10. Save New Project As Dialog Box

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.

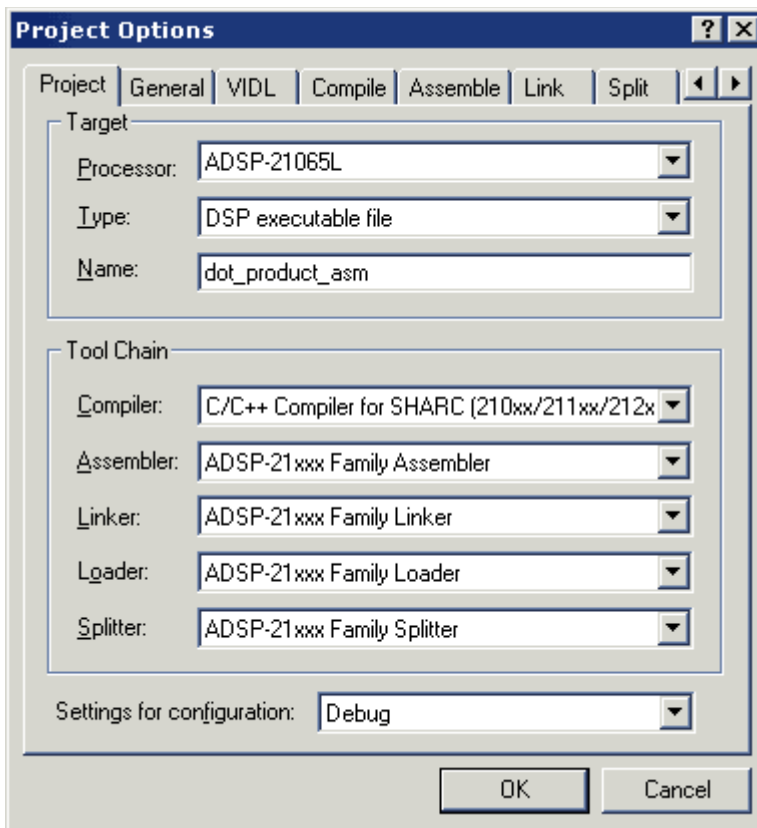


Figure 2-11. Project Options Dialog Box: Project Page

This dialog box enables you to specify project build information.

Exercise Two: Calling an Assembly Routine and Creating an LDF

5. Take a moment to examine the tabbed pages in the **Project Options** window: **Project**, **General**, **VIDL**, **Compile**, **Assemble**, **Link**, **Split**, **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 2-3. Completing the Project Page

Box	Value
Processor	ADSP-21065L
Type	DSP executable file
Name	dot_product_asm
Settings for configuration	Debug

These settings specify information for building an executable file for the ADSP-21065L DSP. 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](#).

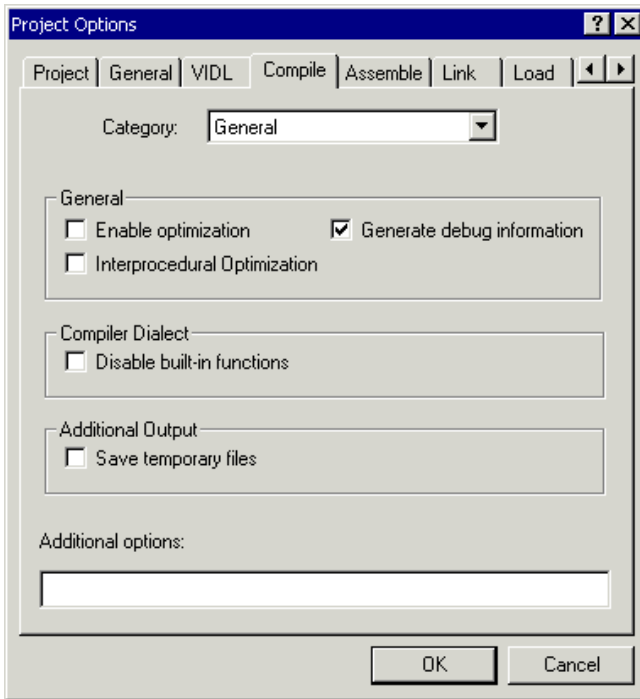


Figure 2-12. Project Options Dialog Box: Compile Page

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**. Once added, kernel support cannot be removed.

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

Exercise Two: Calling an Assembly Routine and Creating an LDF

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.

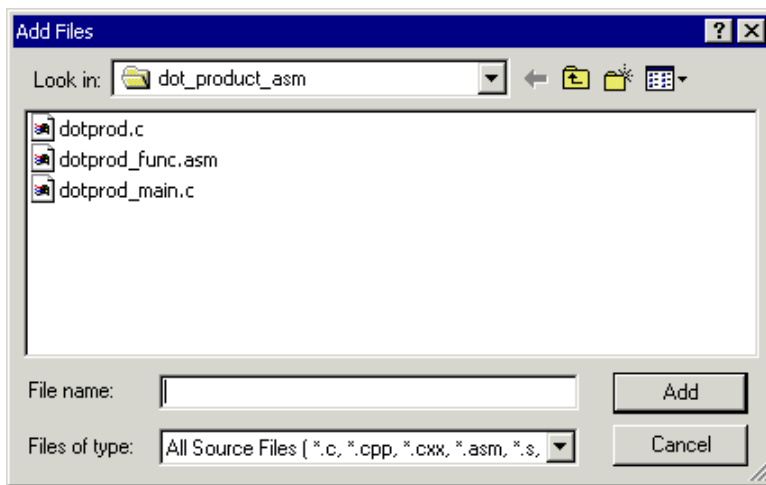


Figure 2-13. Add Files Dialog Box: Adding Source Files to the Project

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](#).

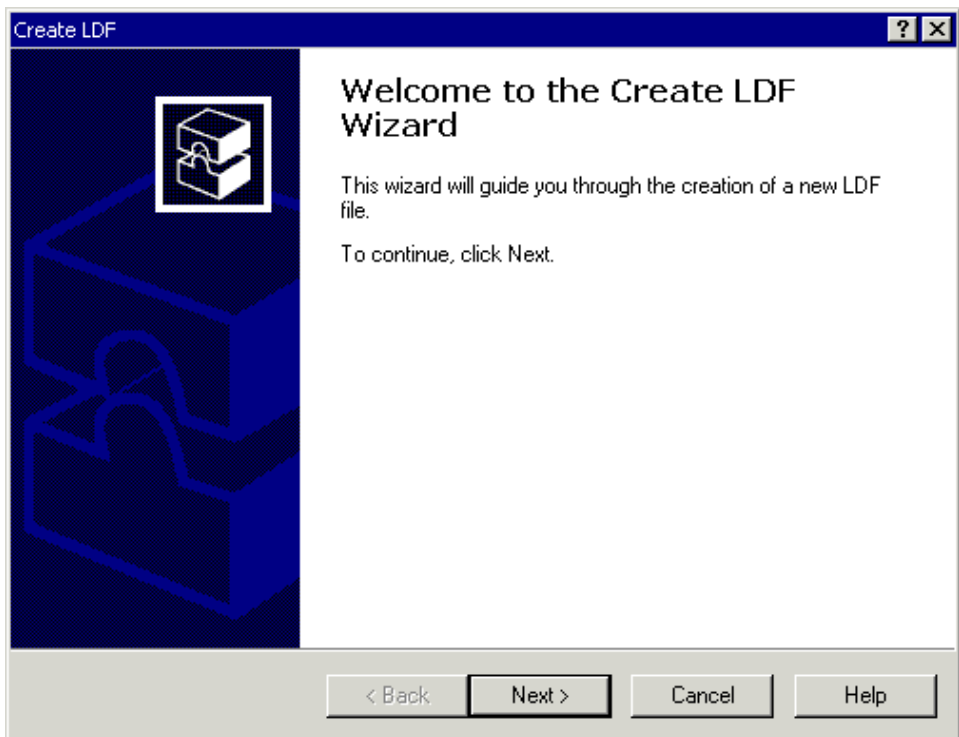


Figure 2-14. Create LDF Wizard

Exercise Two: Calling an Assembly Routine and Creating an LDF

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

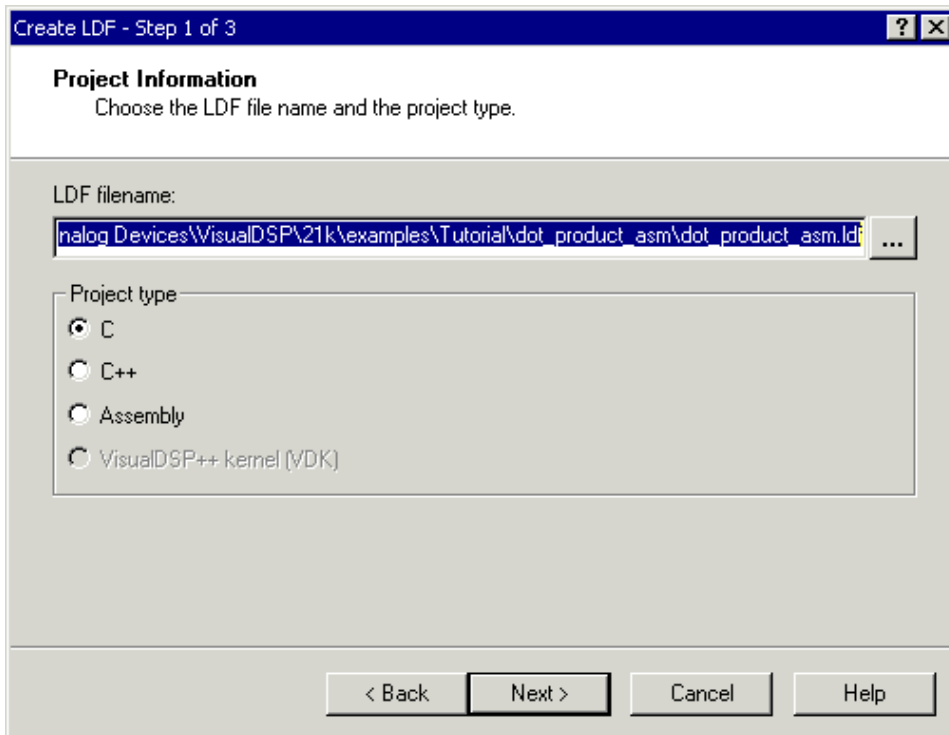


Figure 2-15. Create LDF – Step 1 of 3 Page

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](#).

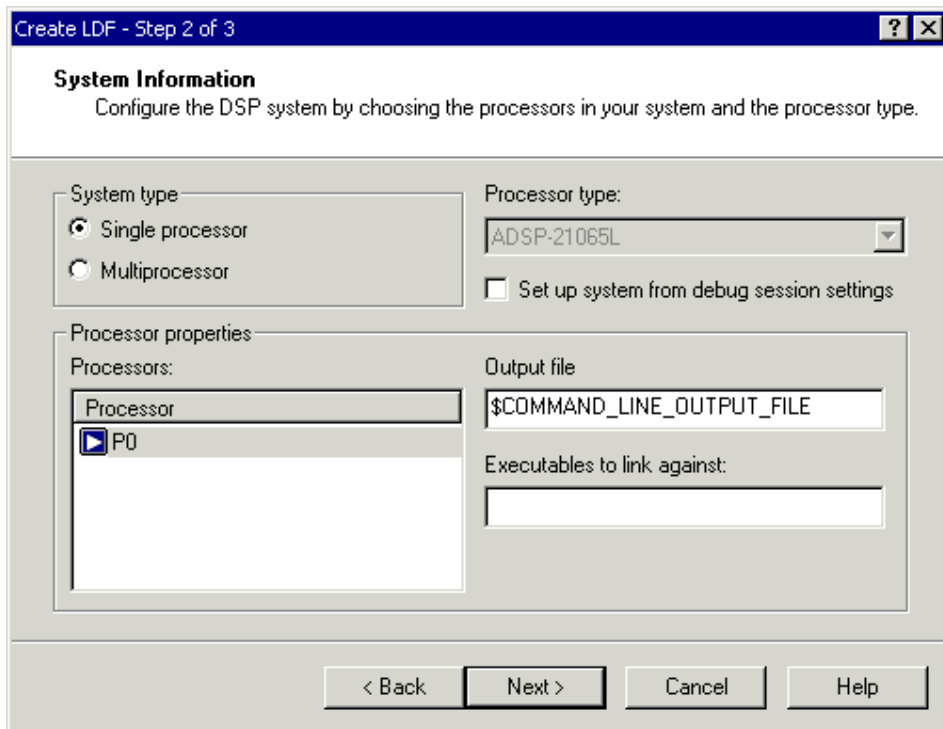


Figure 2-16. Create LDF – Step 2 of 3 Page

This page enables you to set the **System type** (defaulted to **Single processor**), the **Processor type** (defaulted to **ADSP-21065L** 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 next page (**Create LDF – Step 3 of 3**), shown in [Figure 2-17 on page 2-24](#).

Exercise Two: Calling an Assembly Routine and Creating an LDF

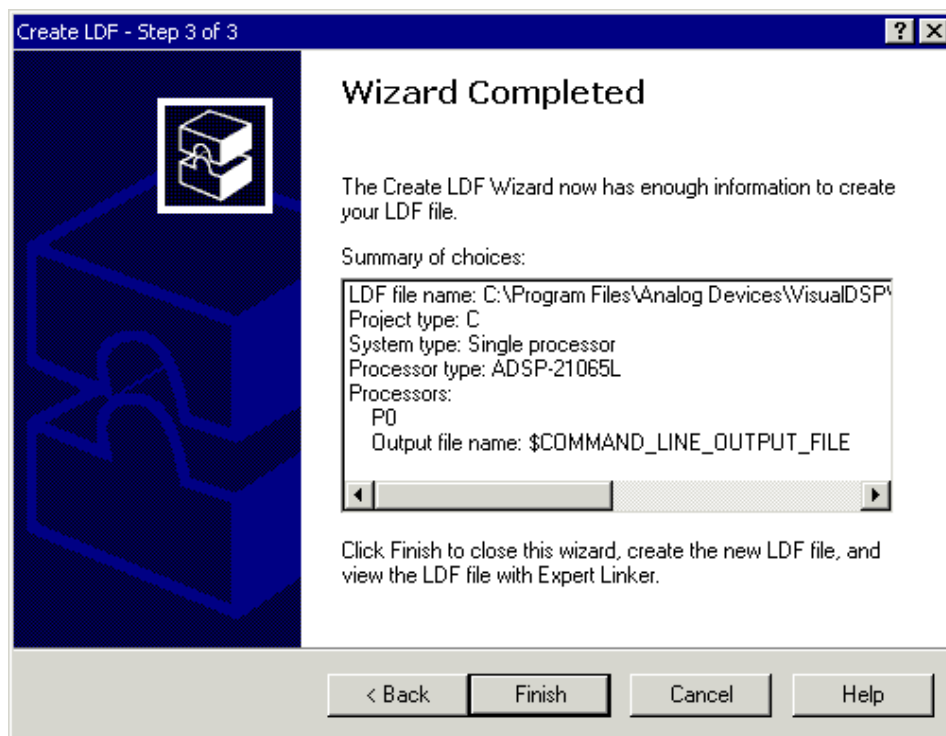


Figure 2-17. Create LDF – Step 3 of 3 Page

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](#).

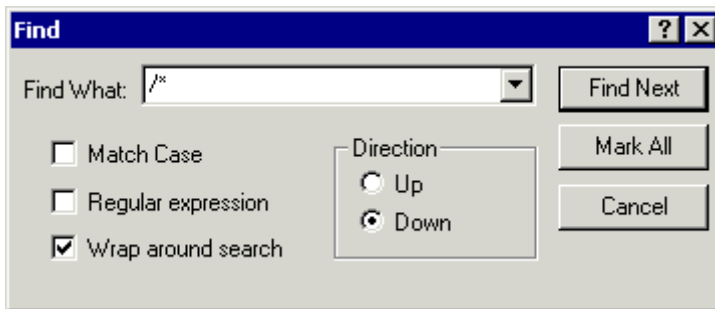


Figure 2-18. Find Dialog Box: Locating Occurrences of `/*`

Exercise Two: Calling an Assembly Routine and Creating an LDF

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 double 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.

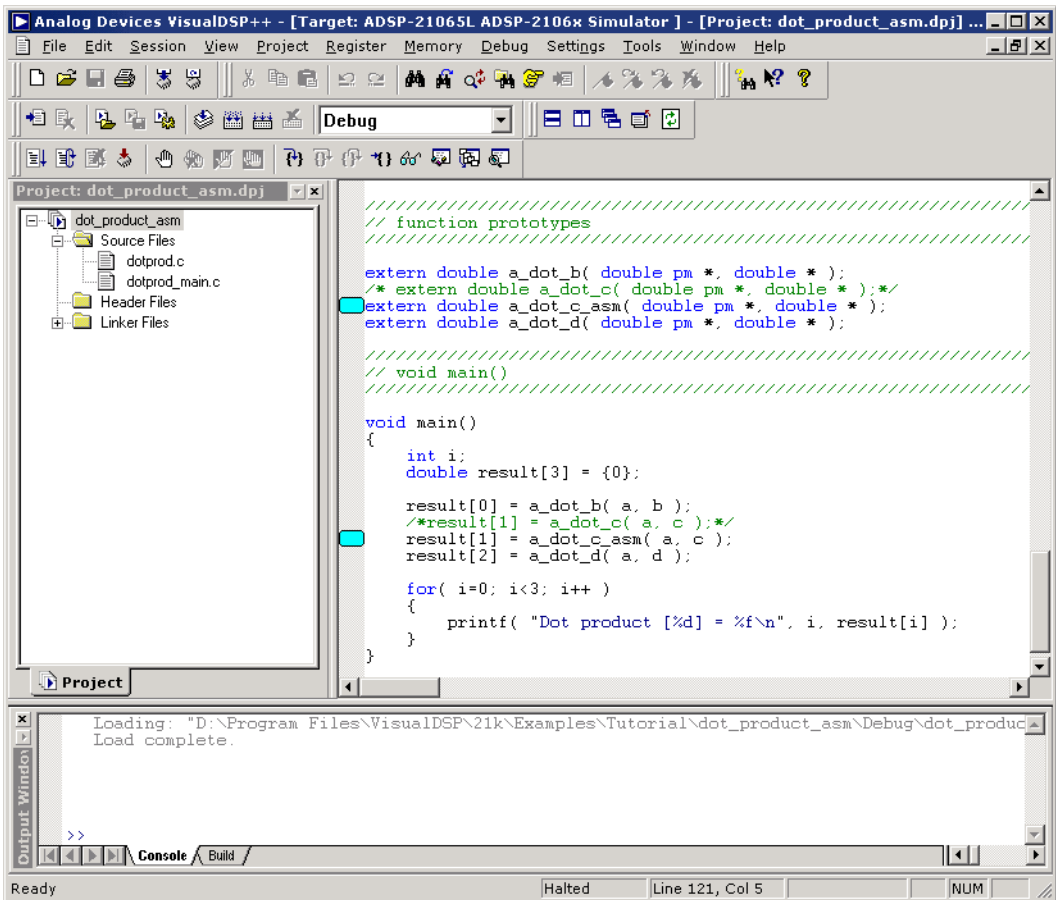


Figure 2-19. Editor Window: Modifying dotprod_main.c to Call a_dot_c_asm

- From the **File** menu, choose **Save** to save the changes to the file.
- 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: Calling an Assembly Routine and Creating an LDF

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](#).

```

"D:\Program Files\VisualDSP\eam21k.exe" -proc ADSP-21065L -o .\Debug\dotprod_func.doj -g .\d
"D:\Program Files\VisualDSP\cc21k" -c .\dotprod_main.c -g -proc ADSP-21065L -o .\Debug\dotp
"D:\Program Files\VisualDSP\cc21k.exe" .\Debug\dotprod.doj .\Debug\dotprod_func.doj .\Debug\dc
[Warning li4000]
The following input section(s) that contain program code
and/or data have not been placed into the executable for
processor P0
as there are no relevant commands specified in the linker
description file.

Filename                Input Section
.\Debug\dotprod_func.doj  pm_code2

[Error li1113] The symbol '_a_dot_c_asm' referenced in file '.\Debug\dotprod_main.doj' in the
[Error li1113] The symbol 'pm_code2' referenced in file '.\Debug\dotprod_func.doj' in the pro
[Error li1113] The symbol '_a_dot_c_asm' referenced in file '.\Debug\dotprod_func.doj' in the
[Error li1113] The symbol 'dploop' referenced in file '.\Debug\dotprod_func.doj' in the proje
[Error li1113] The symbol '_a_dot_c_asm_end' referenced in file '.\Debug\dotprod_func.doj' in

Linker finished with 5 error(s) 1 warning(s)
cc21k: Fatal Error: Link failed
Tool failed with exit/exception code: 1.
Build was unsuccessful.
  
```

Figure 2-20. Output Window: Linker Error

Exercise Two: Calling an Assembly Routine and Creating an LDF

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

 You might have to resize the **Expert Linker** window and scroll to see both panes (**Input Sections** and **Memory Map**).

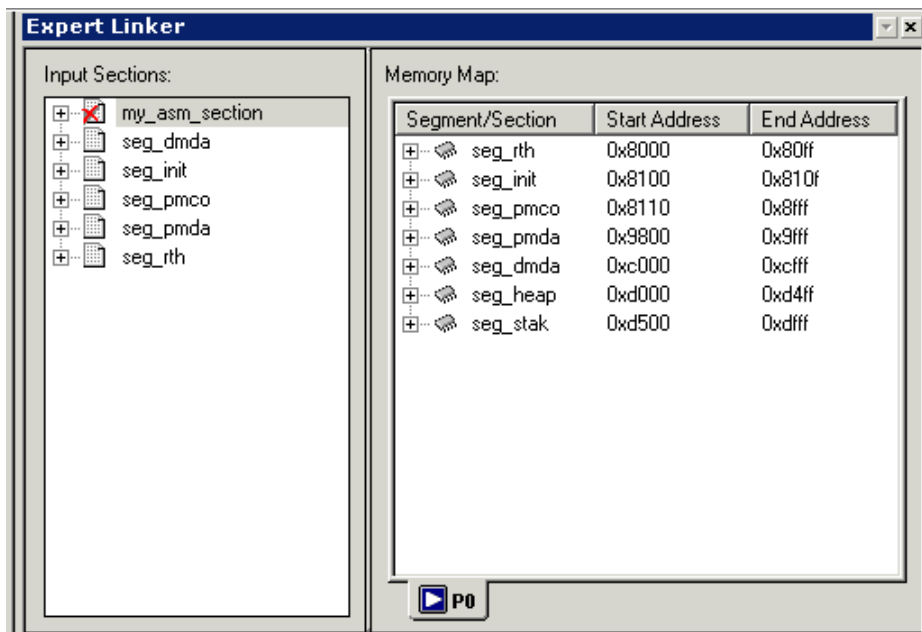


Figure 2-21. Expert Linker Window

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.

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 on page 2-30](#).

5. Map the section `my_asm_section` into the memory segment named `seg_pmco` 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 `seg_pmco`. As shown in [Figure 2-22 on page 2-32](#), the red X should no longer appear because the section `my_asm_section` has been mapped.

Exercise Two: Calling an Assembly Routine and Creating an LDF

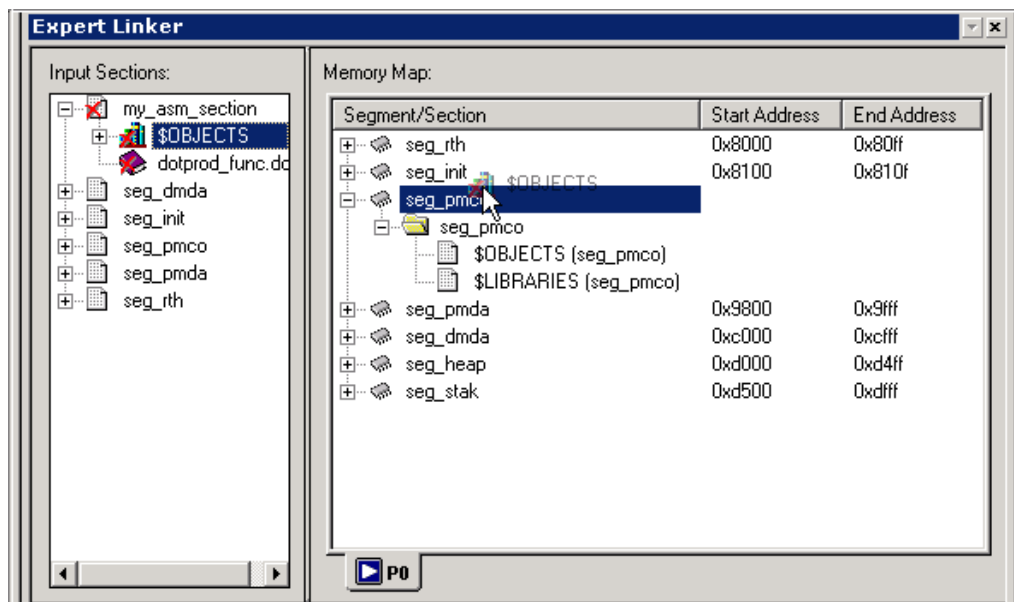


Figure 2-22. Dragging \$OBJECTS onto seg_pmco

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.

Step 6: Rebuild and Run dot_product_asm

To run dot_product_asm:

1. Build the project by clicking the **Build Project** button  or by choosing **Build Project** from the **Project** menu.

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

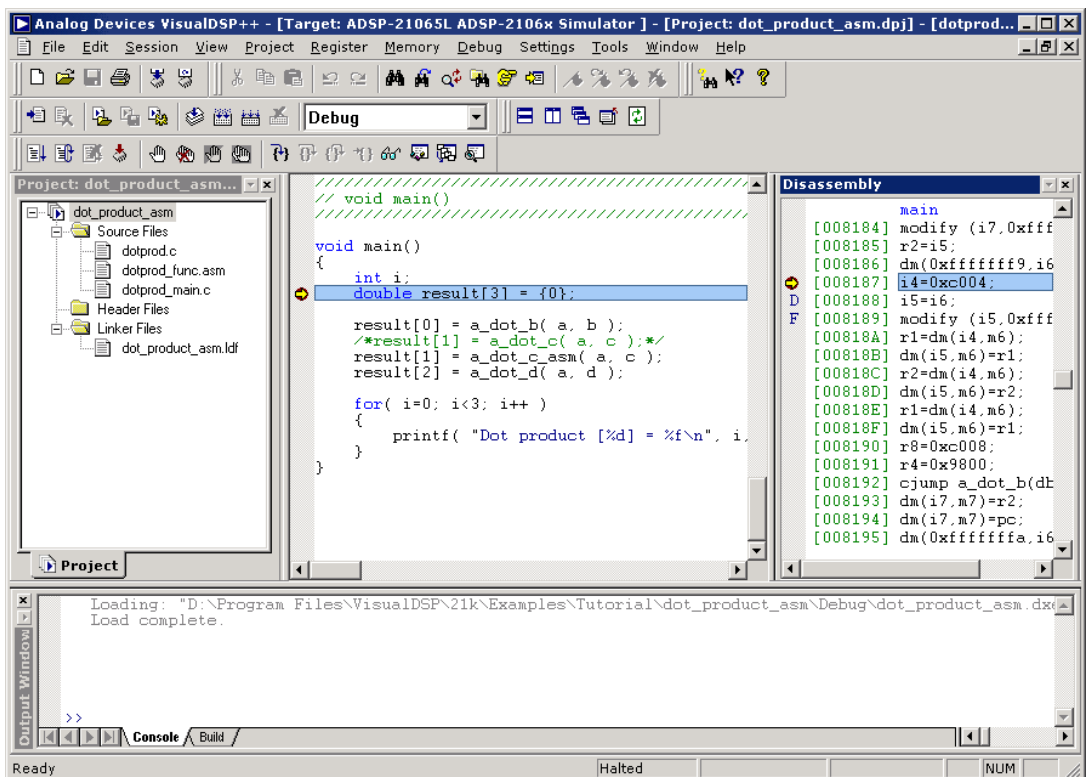


Figure 2-23. dot_product_asm Successfully Built and Loaded

Exercise Three: Plotting Data

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

The program calculates the three dot products and displays the results in the **Console** view 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] = 0.000000
Dot product [1] = 0.707107
Dot product [2] = -0.500000
```

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 convolution algorithm to a buffer of data. You will use the VisualDSP++ plotting engine to view the different data arrays graphically, both before and after running the program.

Step 1: Load the Convolution Program

To load the Convolution program:

1. Close the `dot_product_asm` project, but keep the **Disassembly** window and **Output** window (in the **Console** view) open.
2. From the **File** menu, choose **Load Program** or click . The **Open a Processor Program** dialog box appears.
3. Select the `convolution.dxe` program to load as follows.
 - a. Open your **Analog Devices** folder and double-click the `VisualDSP\21k\Examples\tutorial\convolution\Debug` subfolder.

- b. Double-click `convolution.dxe` to load the program. in an editor window.
- c. If you are prompted to look for `convolution.cpp`, click **Yes** to open the **Find** dialog box and proceed to step d. If VisualDSP++ opens an editor window, proceed to [step 4 on page 2-36](#).
- d. Click the up-one-level button  to access the `convolution` folder.
- e. Double-click `convolution.cpp` to display the file in an editor window, as shown in [Figure 2-24](#).

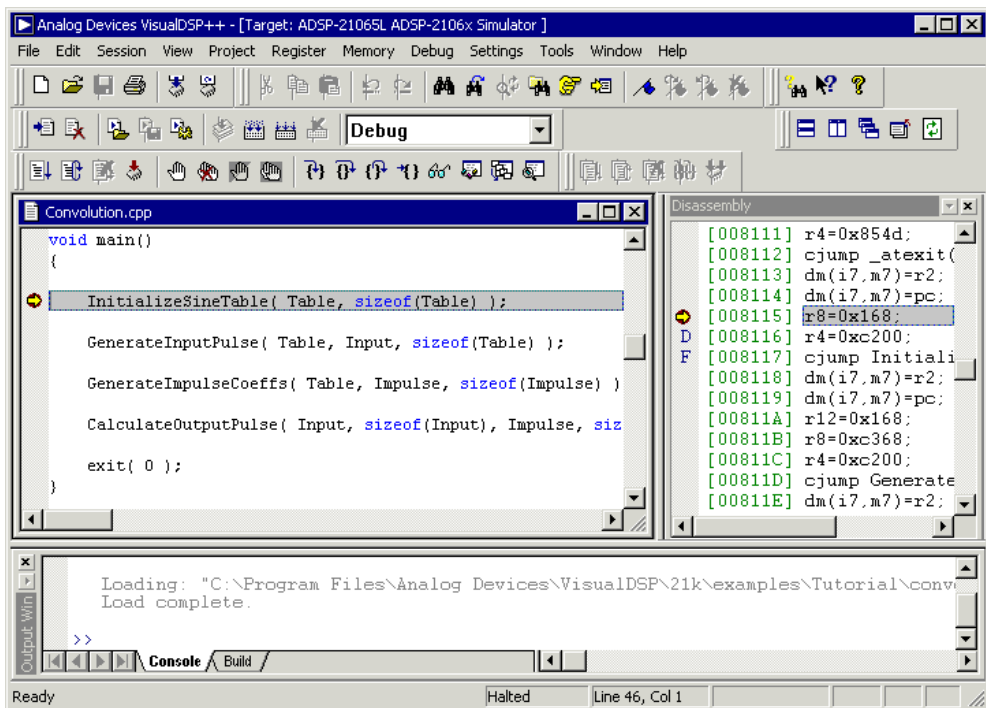


Figure 2-24. Loading the Convolution Program

Exercise Three: Plotting Data

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

You can see four global data arrays:

Table

Input

Output

Impulse

You can also see four functions that operate on these arrays:

InitializeSineTable()

GenerateInputPulse()

GenerateImpulseCoeffs()

CalculateOutputPulse()

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 2-37](#).

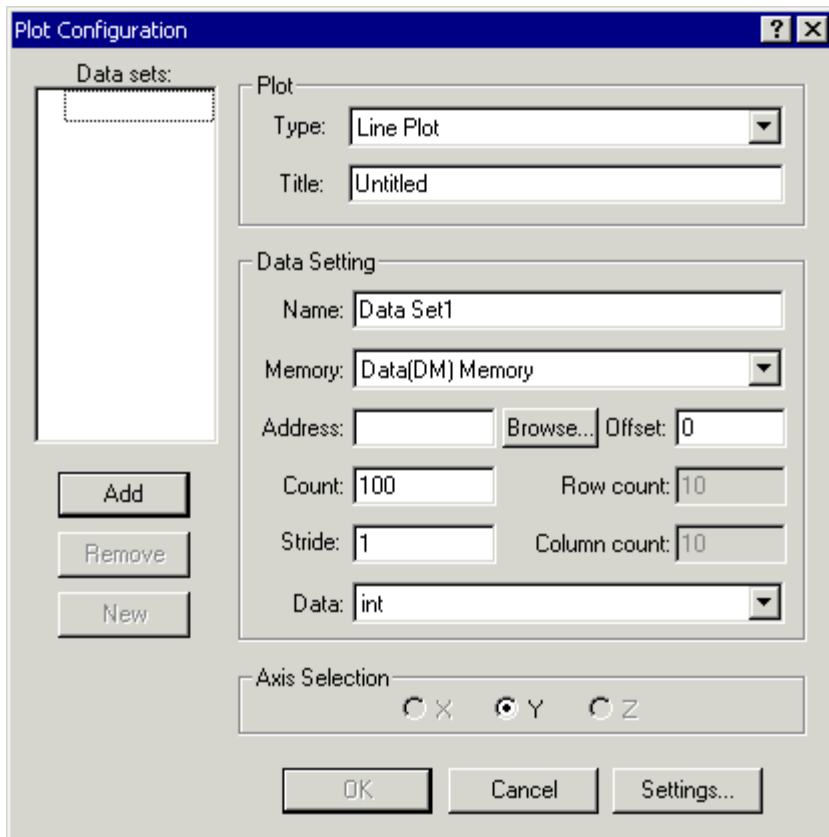


Figure 2-25. Plot Configuration Dialog Box

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 **convolution**.

Exercise Three: Plotting Data

3. Enter three data sets to plot by using the values in [Table 2-4](#).

Table 2-4. Three Data Sets: Table, Input, and Output

Data Setting Field	Table Data Set	Input Data Set	Output Data Set	Description
Name	Table	Input	Output	Data set
Memory	Data(DM) Memory	Data(DM) Memory	Data(DM) Memory	Data memory
Address	Table	Input	Output	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	360	360	396	The arrays are 360 and 396 elements long.
Stride	1	1	1	The data is contiguous in memory.
Data	float	float	float	Input and Output are arrays of float values.
Offset	0	0	0	Use zero, the default value.

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 2-39](#).

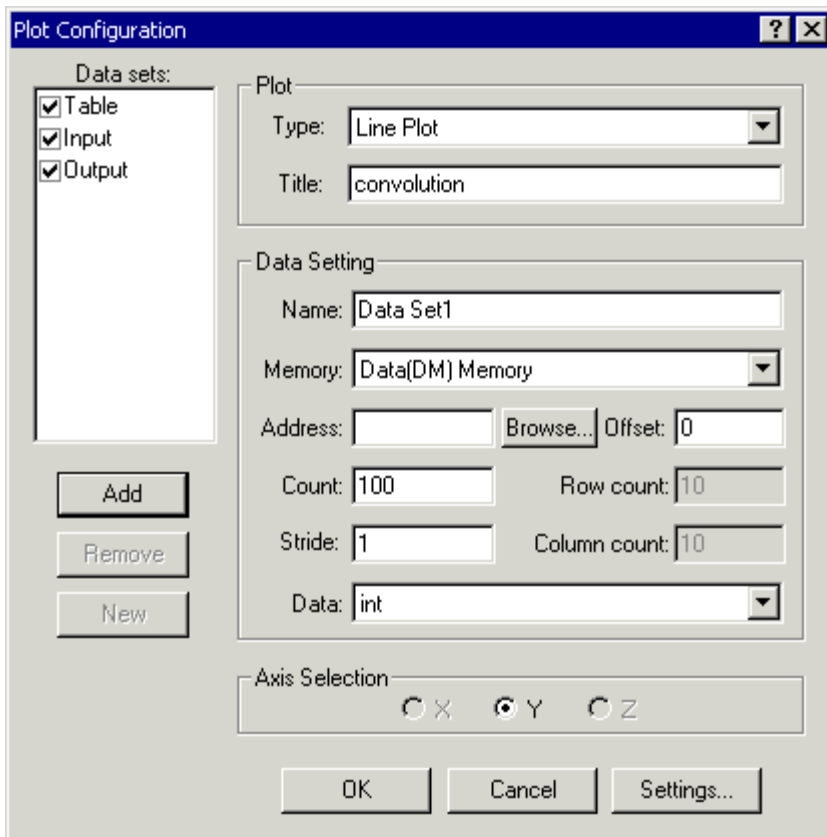


Figure 2-26. Plot Configuration Dialog Box with Table/Input/Output Data Sets

Exercise Three: Plotting Data

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

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

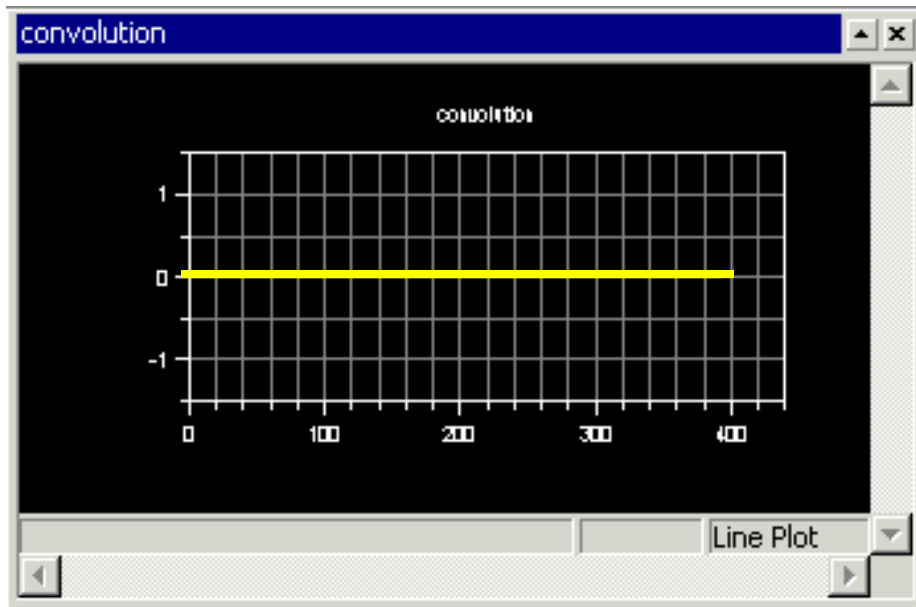


Figure 2-27. Plot Window: Before Running the Convolution Program

To display the legend box in the plot window, right-click in the plot window and choose **Modify Settings**. Then, on the **General** page, select **Legend** in the **Options** group box.



The legend box is not shown in the plot windows shown in this tutorial.

Step 3: Run the Convolution Program and View the Data

To run the Convolution program and view the data:

1. Press **F10** or click the **Step Over** button  to step over the first line in main that calls the `InitializeSineTable()` function.

Stepping over each function enables you to see the data being calculated in a plot window.

2. Step over the call to `GenerateInputPulse()` by using the **Step Over** command as you did in the previous step. The plot window now displays the data for both the Input array and the Table array.

Once you finish stepping over the function, the word “Halted” appears in the status bar at the bottom of the screen. The plot window should now show the sine wave data in the Table array.

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

Exercise Three: Plotting Data

When the program halts, you see the results of the `convolution` algorithm in the Output array. All three data sets are now visible in the plot window, as shown in [Figure 2-28](#).

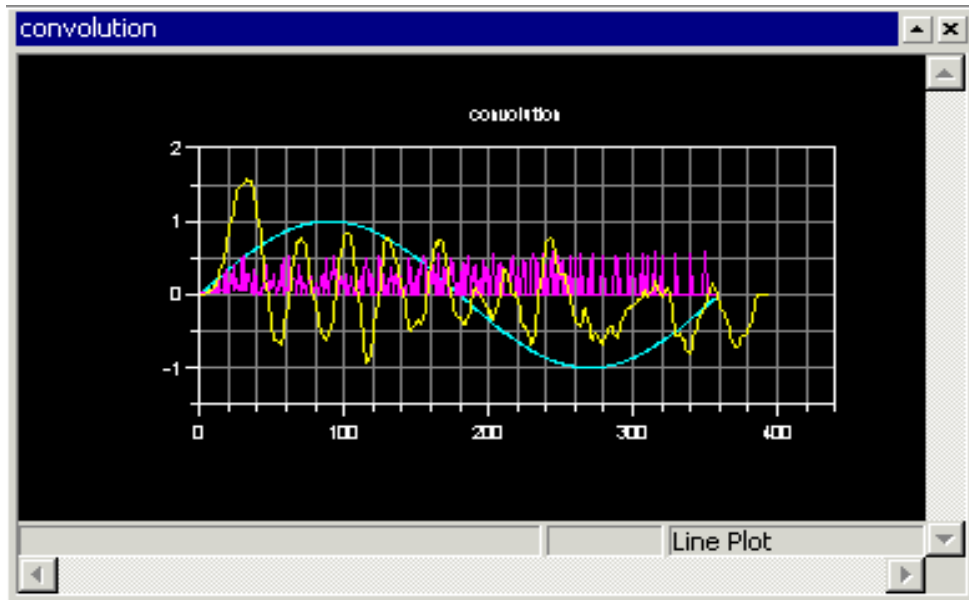


Figure 2-28. Plot Window After Running the Convolution Program to Completion

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

4. 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.

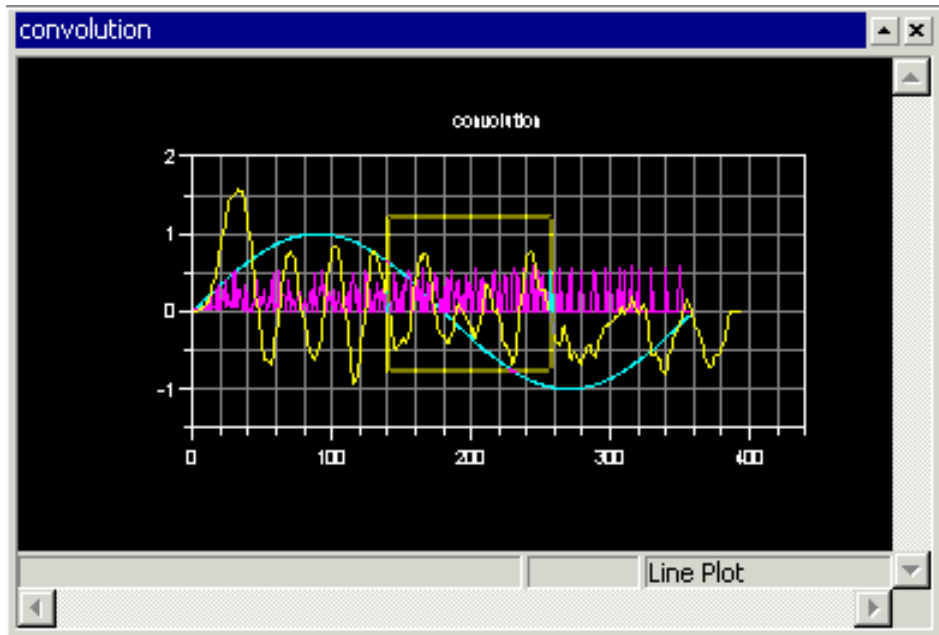


Figure 2-29. Plot Window: Selecting a Region to Magnify

Figure 2-30 on page 2-44 shows the magnified results.

Exercise Three: Plotting Data

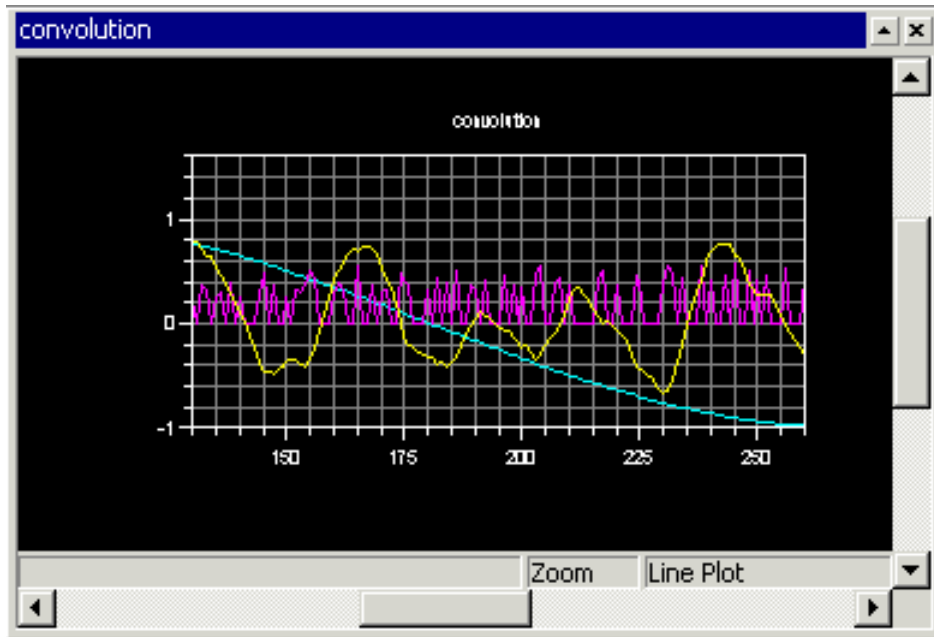


Figure 2-30. Plot Window: Magnified Result

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.

5. 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 (←) and Right (→) 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 on page 2-45](#).



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

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

To disable the data cursor, right-click in the plot window and choose (de-select) **Data Cursor**.

To return to the previous view (before magnification), right-click in the plot window and choose **Reset Zoom** from the popup menu.

You are now ready to begin Exercise Four.

Exercise Four: Linear Profiling

In this exercise, you will load and debug the Convolution program from the previous exercise. You will use linear profiling, however, 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 Convolution Program

To load the Convolution 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\21k\Examples\tutorial\convolution\Debug` subfolder.
 - b. Double-click `convolution.dxe` to load and run the Convolution program.

- c. If you are prompted to look for `convolution.cpp`, click **Yes** to open the **Find** dialog box and proceed to step d. If VisualDSP++ opens an editor window, proceed to “[Step 2: Open the Profiling Window.](#)”
- d. Click the up-one-level button  to access the `convolution` folder.
- e. Double-click `convolution.cpp` to display the file in an editor window.

You are now ready to set up linear profiling.

Step 2: Open the Profiling Window

To open the **Linear Profiling Results** window:

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

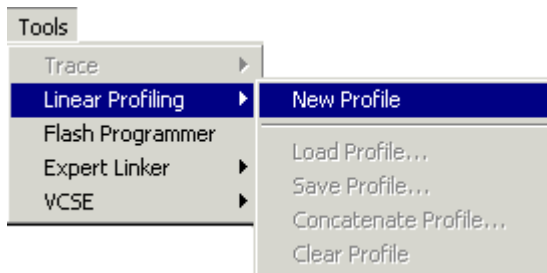


Figure 2-32. Setting Up Linear Profiling for the Convolution Program

The **Linear Profiling Results** window opens.

Exercise Four: Linear Profiling

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 2-33](#).

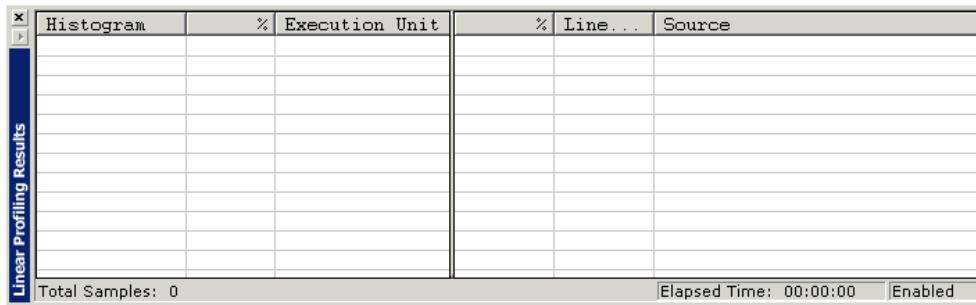
The image shows a screenshot of the 'Linear Profiling Results' window. The window has a title bar with standard OS controls. Below the title bar is a vertical sidebar on the left with the text 'Linear Profiling Results'. The main area of the window contains a table with six columns: 'Histogram', '%', 'Execution Unit', '%', 'Line...', and 'Source'. The table is currently empty, showing only the header rows. At the bottom of the window, there is a status bar with three fields: 'Total Samples: 0', 'Elapsed Time: 00:00:00', and 'Enabled'.

Figure 2-33. Linear Profiling Results Window (Empty)

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

Since we are interested only in high level source code at this point, we will filter out any samples that do not map directly to our source code.

3. Right-click in the **Linear Profiling Results** window and choose **Properties** to display the **Profile Window Properties** dialog box.

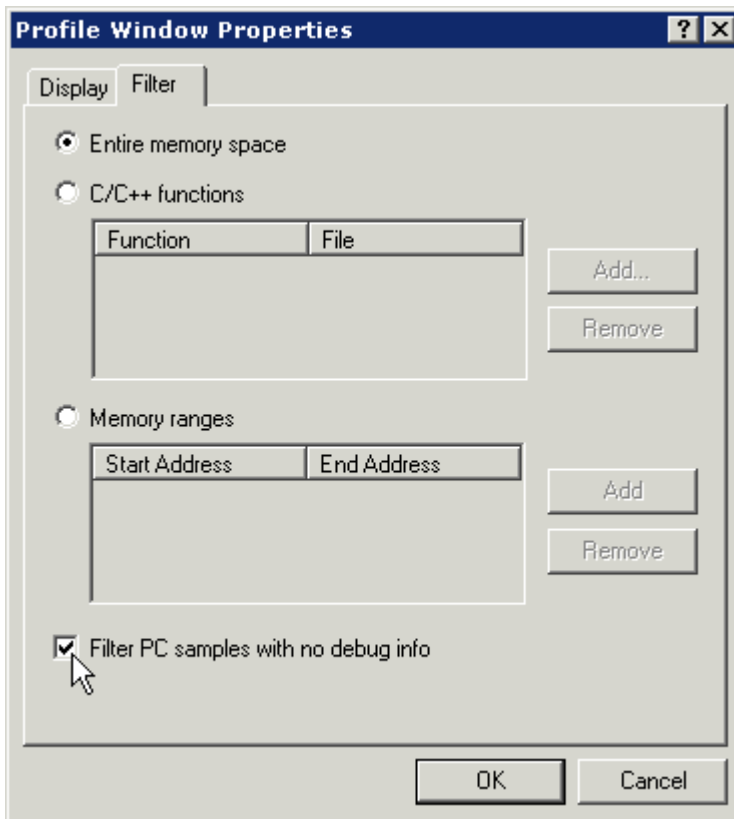


Figure 2-34. Filtering Samples with No Debug Information

4. Select the **Filter** tab (shown in [Figure 2-34](#)) and then click in the **Filter PC samples with no debug info** check box to enable the filter. Click **OK** to close the dialog box.

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 displays the results of the profile data, as shown in [Figure 2-35](#).

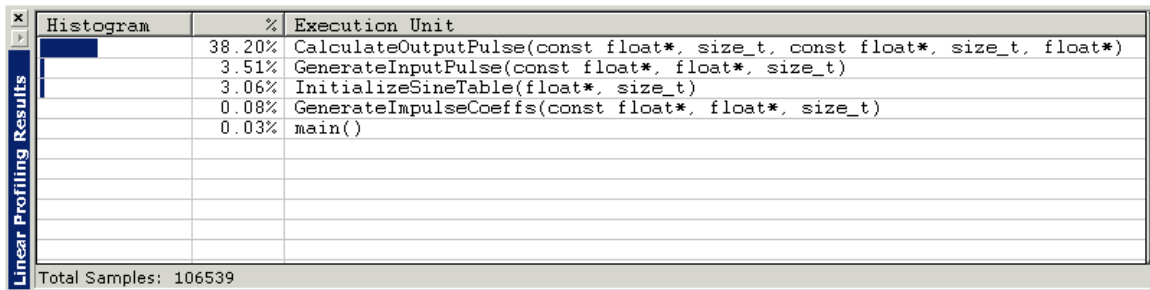


Figure 2-35. Linear Profiling Results of Analyzing the Performance of the Convolution Program – Left Pane

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](#).

If you are prompted to look for `convolution.cpp`, complete these steps:

- Click **Yes** to open the Find dialog box.
- Click the up-one-level button  to access the `convolution` folder.
- Double-click `convolution.cpp` to display the file in an editor window.

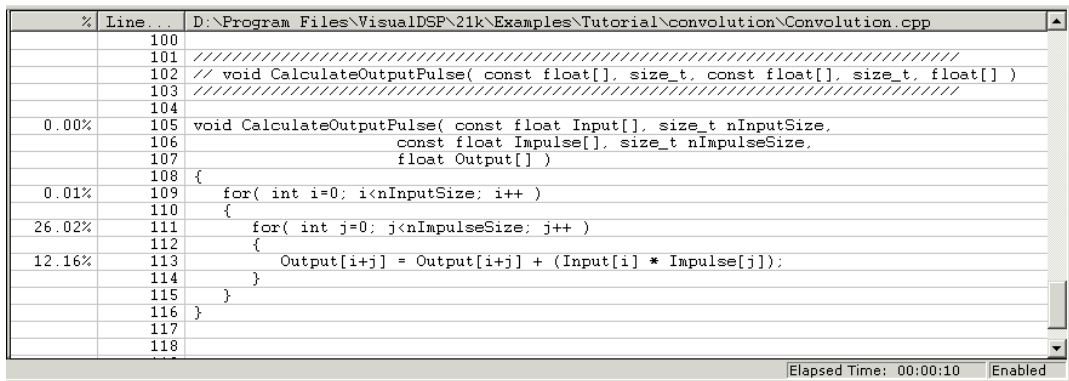


Figure 2-36. Linear Profiling Results of Analyzing the Performance of the Convolution Program – Right Pane

The field values in the left pane are defined on the next page.

Exercise Four: Linear Profiling

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 2-35 on page 2-50](#) the left pane shows that the function `CalculateOutputPulse()` has consumed over 38% of the total execution time. Double-clicking one of these lines displays the source file, `convolution.cpp`, in the right (source) pane. The source pane displays data for each line of executable code in the file for which linear profile data has been collected.

Double-clicking the line with the `CalculateOutputPulse()` function in the left pane displays the linear profile data shown in [Figure 2-37](#) in the right pane.

%	Line	D:\Program Files\VisualDSP\21k\Examples\Tutorial\convolution\Convolution.cpp
	100	
	101	////////////////////////////////////
	102	// void CalculateOutputPulse(const float[], size_t, const float[], size_t, float[])
	103	////////////////////////////////////
	104	
0.00%	105	void CalculateOutputPulse(const float Input[], size_t nInputSize,
	106	const float Impulse[], size_t nImpulseSize,
	107	float Output[])
	108	{
0.01%	109	for(int i=0; i<nInputSize; i++)
	110	{
26.02%	111	for(int j=0; j<nImpulseSize; j++)
	112	{
12.16%	113	Output[i+j] = Output[i+j] + (Input[i] * Impulse[j]);
	114	}
	115	}
	116	}
	117	
	118	

Elapsed Time: 00:00:10 Enabled

Figure 2-37. Linear Profile Data for Convolution.cpp

The details of the `CalculateOutputPulse()` function show that 26.02% of the time spent running the entire Convolution program is spent inside the nested `for` loop, calculating the convolution.

The data suggests that you should rewrite this function in hand-tuned assembly language to decrease the total running time of the algorithm and improve performance.

You are now ready to begin Exercise Five.

Exercise Five: Installing and Using a VCSE Component

In this exercise, you will complete the following tasks.

- Start up the VisualDSP++ environment and select a new session
- 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\21k\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 on the **Project** page under **Settings and Preferences**, 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 **Sessions** menu, choose **New Session**. The **New Session** dialog box appears.
3. From the **Processor** list, choose the **ADSP-21060** processor and click **OK**.
4. From the **Project** menu, choose **Open**.

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

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

`VisualDSP\21k\Examples\tutorial\vcse_component`

Note: This path is based on the default installation.

6. Double-click the `useg711.dpj` project file.

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

Exercise Five: Installing and Using a VCSE Component

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

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\21k\Examples\tutorial\vcse_component`

Note: This path is based on the default installation.

4. Double-click the `examples_culawc_21K.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 2-38 on page 2-57](#)).

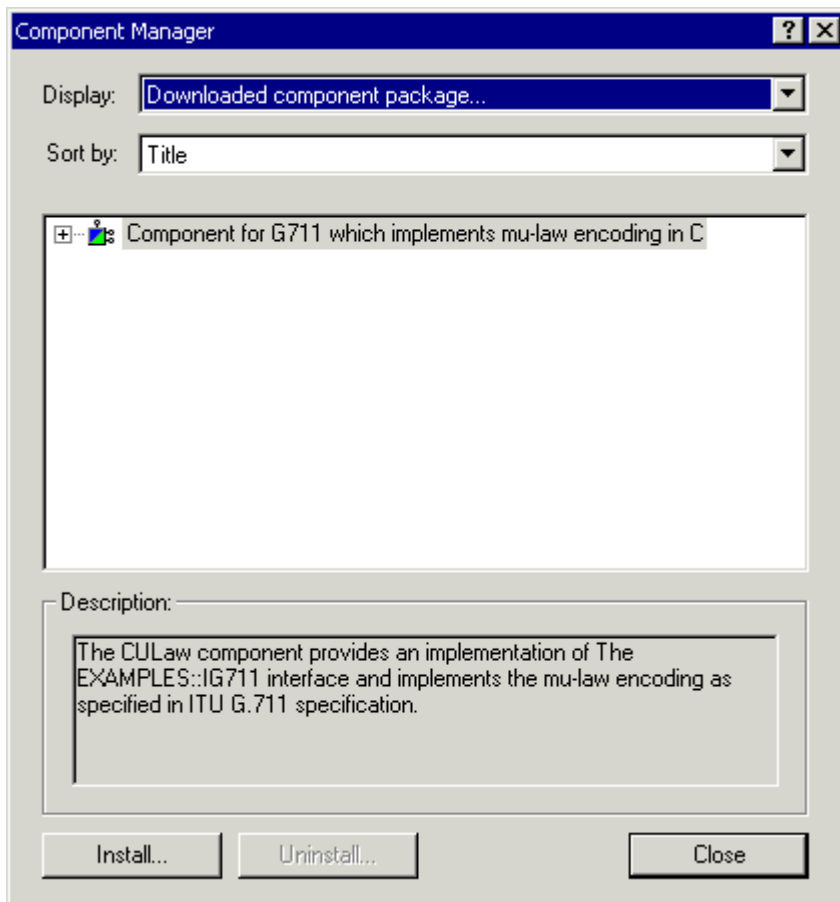


Figure 2-38. Component Manager Dialog Box – Downloaded Component

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

Exercise Five: Installing and Using a VCSE Component

5. 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 2-39](#).

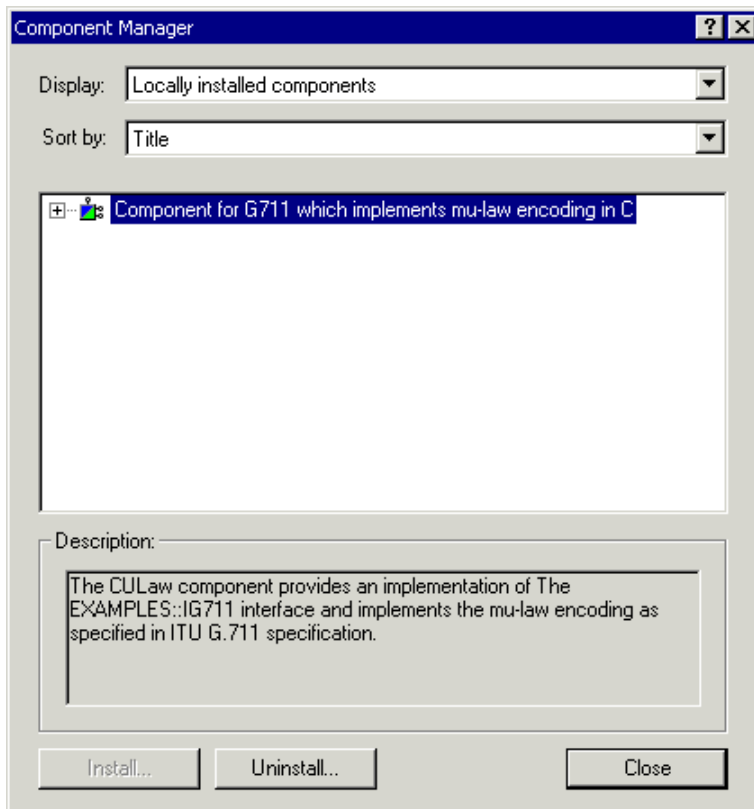


Figure 2-39. Component Manager Dialog Box – Selected Component

Click **Close** to close Component Manager.

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.

If you have multiple components on your system and you are not sure which one to add, click the expand button  to display the component information, as shown in [Figure 2-40](#).

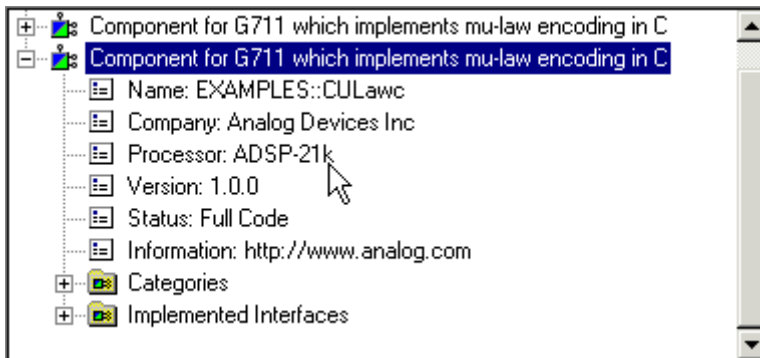


Figure 2-40. Expanded View of Component Information

Make sure that **Processor: ADSP-21k** is listed for the component that you are adding to your project.

Exercise Five: Installing and Using a VCSE Component

3. Click **Add** to indicate that you want to add the component to the project. Component Manager displays the dialog box shown in [Figure 2-41](#).

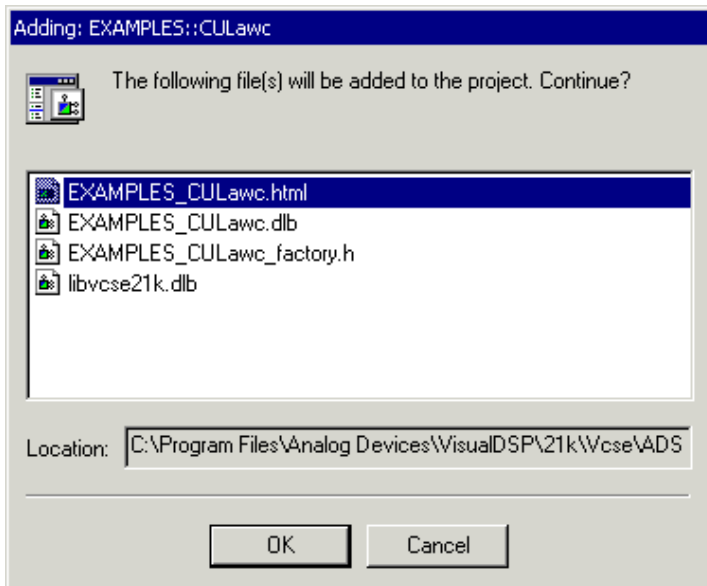


Figure 2-41. Adding Files to the Project

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 2-42](#).

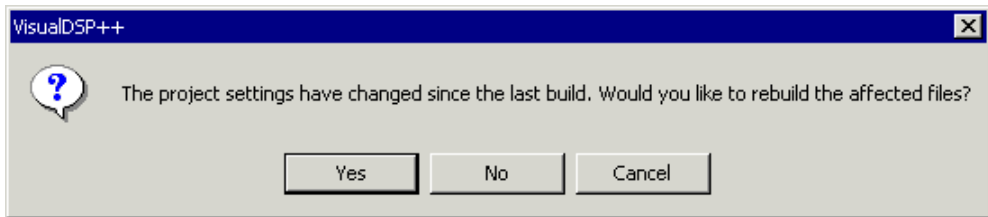


Figure 2-42. Rebuilding Files Affected by Changes to Project Settings

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 exercise and the tutorial.

Exercise Five: Installing and Using a VCSE Component

I INDEX

Symbols

% of Histogram data, defined, [2-52](#)
/*, [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 assembly routines,
 [2-16](#)
code development tools
 features, [1-8](#)
 overview, [1-2](#)
commands
 Build Project, [2-7](#)
 Rebuild All, [2-7](#)

comment characters, moving in source
 files, [2-26](#)

Compile page, [2-18](#)

Console page, [2-15](#)

conventions, manual, [xv](#)

Convolution program

 data arrays, [2-36](#)

 global data arrays, [2-36](#)

 loading, [2-34](#)

 running, [2-41](#)

 viewing the results, [2-42](#)

convolution.cpp source file, [2-35](#), [2-51](#),
 [2-52](#)

convolution.dxe

 loading, [2-34](#)

See also Convolution program

creating

 debug sessions, [2-10](#)

 new projects, [2-16](#)

customer support, [ix](#)

D

Data Cursor command, [2-44](#), [2-45](#)

data sets

 input and output, [2-38](#)

 plotting, [2-36](#)

debug features, [1-4](#)

INDEX

- debug sessions
 - setting up new sessions, [2-10](#)
 - setting up subsequent sessions, [2-10](#)
 - dialog boxes
 - Add Files, [2-20](#)
 - Adding (VCSE files), [2-60](#)
 - Breakpoints, [2-14](#)
 - Component Manager (VCSE), [2-58](#)
 - Find, [2-25](#)
 - New Session, [2-10](#)
 - Plot Configuration, [2-36](#)
 - Profile Window Properties, [2-49](#)
 - Project Options, [2-17](#), [2-18](#)
 - Save New Project As, [2-16](#)
 - Disassembly window
 - adding and deleting breakpoints, [2-14](#)
 - information displayed, [2-13](#)
 - dot_product
 - rebuilding, [2-33](#)
 - running, [2-34](#)
 - dot_product_asm, building the project, [2-17](#)
 - dot_product_c, folder location, [2-5](#)
 - 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, [2-52](#)
 - exercises
 - building and running C programs, [2-3](#)
 - installing and using a VCSE component, [2-54](#)
 - linear profiling, [2-46](#)
 - modifying a C program to call an assembly routine, [2-16](#)
 - plotting data, [2-34](#)
 - Expert Linker
 - using to create a Linker Description File, [2-21](#)
 - window, [2-30](#), [2-32](#)
- F
- Find dialog box, [2-22](#), [2-25](#)
- G
- General page, [2-6](#)
 - Generate debug information check box, [2-19](#)
- H
- histogram, defined, [2-52](#)
- I
- input data sets, entering, [2-38](#)

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

J

JTAG emulators, [2-46](#)

L

linear profiling, [2-46](#)

collecting and examining profile data,
[2-50](#)

Linear Profiling Results window, [2-50](#)

loading the Convolution program,
[2-46](#)

opening the Profiling window, [2-47](#)

results of analyzing the Convolution
program, [2-50](#)

viewing profile data for
convolution.cpp, [2-53](#)

Linker Description File (LDF)

creating, [2-21](#)

folder, [2-2](#)

linker errors, viewing in Output
window, [2-29](#)

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

M

magnifying selected regions, [2-43](#)

messages

Output window, [2-13](#)

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

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

N

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

O

opening projects, [2-4](#)

output data sets, entering, [2-38](#)

Output window, [2-8](#)

Console view, [2-15](#)

information displayed, [2-13](#)

viewing linker errors, [2-29](#)

P

Plot Configuration dialog box, [2-36](#),
[2-38](#)

plot windows

after running the Convolution
program, [2-42](#)

before running the Convolution
program, [2-40](#)

magnified result, [2-43](#)

magnifying data points, [2-44](#)

opening, [2-36](#)

selecting a region to magnify, [2-43](#)

viewing data points, [2-45](#)

zooming in on a region, [2-42](#)

plotting

data, [2-34](#)

specifying data sets, [2-36](#)

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

Profile Window Properties dialog box,
[2-49](#)

INDEX

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, [2-33](#)
 building dotprodc, [2-7](#)
 creating an .LDF file, [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, [2-44](#), [2-45](#)

S

Save New Project As dialog box, [2-16](#)
source files
 adding to projects, [2-20](#)
 modifying, [2-25](#)
statistical profiling, [2-46](#)

T

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

V

VCSE

 Adding (files) dialog box, [2-60](#)
 adding a VCSE component to a project, [2-59](#)
 building and running the program, [2-61](#)
 Component Manager dialog boxes, [2-57](#), [2-58](#)
 installing a VCSE component, [2-56](#)
View Sample Count command, [2-52](#)

VisualDSP++

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

W

windows

 Disassembly, [2-13](#), [2-14](#)
 editor, [2-9](#), [2-26](#)
 Expert Linker, [2-30](#), [2-32](#)
 Linear Profiling Results, [2-48](#), [2-50](#), [2-51](#), [2-53](#)
 Output, [2-8](#), [2-9](#), [2-13](#)
 plot, [2-40](#), [2-42](#), [2-43](#), [2-45](#)