Copyright Information

©2007 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 icon bar and logo, the CROSSCORE logo, VisualDSP++, Blackfin, SHARC, TigerSHARC, and EZ-KIT Lite are registered trademarks of Analog Devices, Inc.

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

   MyAnalog.com ................................................................ xi

   Processor Product Information ........................................... xii

Related Documents ............................................................... xii

Online Technical Documentation ............................................ xiii

   Accessing Documentation From VisualDSP++ .................... xiv

   Accessing Documentation From Windows ........................... xiv

   Accessing Documentation From the Web .............................. xv

Printed Manuals ................................................................. xv

   Hardware Tools Manuals ................................................. xv

   Processor Manuals ......................................................... xv

   Data Sheets ..................................................................... xv
CONTENTS

Notation Conventions ................................................................. xvi

FEATURES AND TOOLS

VisualDSP++ Features ............................................................... 1-1
Code Development Tools ......................................................... 1-4
Connecting to a Debug Session .................................................. 1-6

BASIC TUTORIAL

Overview .................................................................................... 2-1
Exercise One: Building and Running a C Program .................. 2-3
   Step 1: Start VisualDSP++ and Open a Project .................. 2-3
   Step 2: Build the dotprodc Project .............................. 2-7
   Step 3: Run the Program .............................................. 2-9
   Step 4: Run dotprodc .................................................. 2-14
Exercise Two: Modifying a C Program to Call an Assembly Language Routine ........................................... 2-15
   Step 1: Create a New Project ....................................... 2-15
   Step 2: Add Source Files to dot_product_asm .................. 2-21
   Step 3: Modify the Project Source Files ......................... 2-22
   Step 4: Use the Expert Linker to Modify dot_prod_asm.ldf .... 2-25
   Step 5: Rebuild and Run dot_product_asm ...................... 2-29
Exercise Three: Plotting Data .................................................... 2-31
   Step 1: Load the FIR Program ...................................... 2-31
   Step 2: Open a Plot Window ......................................... 2-33
   Step 3: Run the FIR Program and View the Data .......... 2-37
Exercise Four: Linear Profiling .................................................. 2-45
CONTENTS

Step 1: Load the FIR Program ................................................ 2-45
Step 2: Open the Profiling Window ....................................... 2-46
Step 3: Collect and Examine the Linear Profile Data .............. 2-48

ADVANCED TUTORIAL

Overview ................................................................. 3-1
Exercise One: Using Profile-Guided Optimization ............... 3-2
  Step 1: Load the Project ............................................ 3-4
  Step 2: Configure a Data Set ..................................... 3-6
  Step 3: Attach an Input Stream .................................. 3-11
  Step 4: Configure Additional Data Sets ....................... 3-16
  Step 5: Create PGO Files and Optimize the Program ....... 3-18
  Step 6: Compare Execution Times ............................ 3-19
Exercise Two: Using Background Telemetry Channel .......... 3-23
  Running the BTC Assembly Demo ............................ 3-23
    Step 1: Load the BTC_AsmDemo Project ................. 3-24
    Step 2: Examine the BTC Commands .................... 3-25
    Step 3: Set Up the BTC Memory Window and View Data 3-28
  Running the BTC FFT Demo .................................. 3-36
    Step 1: Build the FFT Demo ............................... 3-37
    Step 2: Plot BTC Data ....................................... 3-38
    Step 3: Record and Analyze BTC Data ................. 3-43

INDEX
Thank you for purchasing Analog Devices, Inc. development software for signal processing applications.

**Purpose of This Manual**

The *VisualDSP++ Getting Started Guide* provides basic and advanced tutorials that highlight many VisualDSP++ features. By completing the step-by-step procedures, you will become familiar with the VisualDSP++ environment and learn how to use these features in your own digital signal processing (DSP) development projects.

**Intended Audience**

The primary audience for this manual is a programmer who is familiar with Analog Devices processors. This manual assumes that the audience has a working knowledge of the appropriate processor architecture and instruction set. Programmers who are unfamiliar with Analog Devices processors can use this manual but should supplement it with other texts (such as the *ADSP-BF533 Blackfin Processor Hardware Reference* that includes information about the ADSP-BF531 and ADSP-BF532 processors).

Programmers who are unfamiliar with VisualDSP++ should refer to the VisualDSP++ online Help and user’s or getting started guides. For the locations of these documents, see “Related Documents”.
This manual consists of:

- **Chapter 1, “Features and Tools”**
  Provides an overview of VisualDSP++ features and code development tools

- **Chapter 2, “Basic 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.

- **Chapter 3, “Advanced Tutorial”**
  Provides step-by-step instructions for using profile-guided optimization (PGO) and background telemetry channel (BTC)

**What’s New in This Manual**

This manual contains updated example screens and procedures for the VisualDSP++ 5.0 integrated development and debugging environment (IDDE).
**Technical or Customer Support**

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


- E-mail tools questions to processor.tools.support@analog.com

- E-mail processor questions to
  - processor.support@analog.com (World wide support)
  - processor.europe@analog.com (Europe support)
  - processor.china@analog.com (China support)

- Phone questions to 1-800-ANALOGD

- Contact your Analog Devices, Inc. local sales office or authorized distributor

- Send questions by mail to:
  Analog Devices, Inc.
  One Technology Way
  P.O. Box 9106
  Norwood, MA 02062-9106
  USA

**Supported Processors**

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

**TigerSHARC (ADSP-TSxxx) Processors**

The name “TigerSHARC” refers to a family of floating-point and fixed-point [8-bit, 16-bit, and 32-bit] processors. VisualDSP++ currently supports the following TigerSHARC processors:

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

**SHARC (ADSP-21xxx) Processors**

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

- ADSP-21020
- ADSP-21060
- ADSP-21061
- ADSP-21062
- ADSP-21065L
- ADSP-21160
- ADSP-21161
- ADSP-21162
- ADSP-21261
- ADSP-21262
- ADSP-21263
- ADSP-21264
- ADSP-21265
- ADSP-21266
- ADSP-21267
- ADSP-21268
- ADSP-21269
- ADSP-21271
- ADSP-21362
- ADSP-21363
- ADSP-21364
- ADSP-21365
- ADSP-21366
- ADSP-21367
- ADSP-21368
- ADSP-21369
- ADSP-21371
- ADSP-21375

**Blackfin (ADSP-BFxxx) Processors**

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

- ADSP-BF531
- ADSP-BF532
- ADSP-BF533
- ADSP-BF534
- ADSP-BF535
- ADSP-BF536
- ADSP-BF537
- ADSP-BF538
- ADSP-BF539
- ADSP-BF522
- ADSP-BF525
Product Information

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

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

MyAnalog.com

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

Registration:

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

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

Processor Product Information

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

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

- E-mail questions or requests for information to
  processor.support@analog.com (World wide support)
  processor.europe@analog.com (Europe support)
  processor.china@analog.com (China support)

- Fax questions or requests for information to
  1-781-461-3010 (North America)
  +49-89-76903-157 (Europe)

Related Documents

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

- VisualDSP++ User’s Guide
- VisualDSP++ Assembler and Preprocessor Manual
- VisualDSP++ Run-Time Library Manual for SHARC Processors
- VisualDSP++ Licensing Guide
- VisualDSP++ C/C++ Compiler Manual for SHARC Processors
- VisualDSP++ C/C++ Compiler and Library Manual for TigerSHARC Processors
Preface

- *VisualDSP++ C/C++ Compiler and Library Manual for Blackfin Processors*
- *VisualDSP++ Linker and Utilities Manual*
- *VisualDSP++ Loader Manual*
- *Device Drivers and System Services Manual for Blackfin Processors*
- *VisualDSP++ Product Release Bulletin*
- *VisualDSP++ Quick Installation Reference Card*

For hardware information, refer to your processor’s hardware reference, programming reference, or data sheet. All documentation is available online. Most documentation is available in printed form.

Visit the Technical Library Web site to access all processor and tools manuals and data sheets:

**Online Technical Documentation**

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

A description of each documentation file type is as follows.
Product Information

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>.chm</td>
<td>Help system files and VisualDSP++ tools manuals.</td>
</tr>
<tr>
<td>.html</td>
<td>Dinkum Abridged C++ library and FlexLM network license manager software doc-</td>
</tr>
<tr>
<td></td>
<td>umentation. Viewing and printing the .html files require a browser, such as</td>
</tr>
<tr>
<td></td>
<td>Internet Explorer 6.0 (or higher).</td>
</tr>
<tr>
<td>.pdf</td>
<td>VisualDSP++ tools manuals in Portable Documentation Format, one .pdf file for</td>
</tr>
<tr>
<td></td>
<td>each manual. Viewing and printing the .pdf files require a PDF reader, such</td>
</tr>
<tr>
<td></td>
<td>as Adobe Acrobat Reader (4.5 or higher).</td>
</tr>
</tbody>
</table>

If documentation is not installed on your system as part of the software installation, you can add it from the VisualDSP++ CD-ROM at any time by running the VisualDSP++ installation. Access the online documentation from the VisualDSP++ environment, Windows® Explorer, or the Analog Devices Web site.

Accessing Documentation From VisualDSP++

From the VisualDSP++ environment:

- Access VisualDSP++ online Help from the Help menu’s Contents, Search, and Index commands.

- Open online Help from context-sensitive user interface items (toolbar buttons, menu commands, and windows).

Accessing Documentation From Windows

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

Help system files (.chm) are located in the Help folder, and .pdf files are located in the Docs folder of your VisualDSP++ installation CD-ROM.
Using Windows Explorer

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

Accessing Documentation From the Web

Download manuals at the following Web site:

```
```

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

Printed Manuals

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

Hardware Tools Manuals

To purchase EZ-KIT Lite® and in-circuit emulator (ICE) manuals, call 1-603-883-2430. The manuals may be ordered by title or by product number located on the back cover of each manual.

Processor Manuals

Hardware reference and instruction set reference manuals may be ordered through the Literature Center at 1-800-ANALOGD (1-800-262-5643), or downloaded from the Analog Devices Web site. Manuals may be ordered by title or by product number located on the back cover of each manual.
Notation Conventions

Data Sheets

All data sheets (preliminary and production) may be downloaded from the Analog Devices Web site. Only production (final) data sheets (Rev. 0, A, B, C, and so on) can be obtained from the Literature Center at 1-800-ANALOGD (1-800-262-5643); they also can be downloaded from the Web site.

To have a data sheet faxed to you, call the Analog Devices Faxback System at 1-800-446-6212. Follow the prompts and a list of data sheet code numbers will be faxed to you. If the data sheet you want is not listed, check for it on the Web site.

Notation Conventions

Text conventions used in this manual are identified and described as follows.

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close command (File menu)</td>
<td>Titles in reference sections indicate the location of an item within the VisualAudio environment's menu system (for example, the Close command appears on the File menu).</td>
</tr>
<tr>
<td>{this</td>
<td>that}</td>
</tr>
<tr>
<td>[this</td>
<td>that]</td>
</tr>
<tr>
<td>[this,..]</td>
<td>Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipse; read the example as an optional comma-separated list of this.</td>
</tr>
<tr>
<td>.SECTION</td>
<td>Commands, directives, keywords, and feature names are in text with letter gothic font.</td>
</tr>
<tr>
<td>filename</td>
<td>Non-keyword placeholders appear in text with italic style format.</td>
</tr>
</tbody>
</table>
Additional conventions, which apply only to specific chapters, may appear throughout this document.
Notation Conventions
1 FEATURES AND TOOLS

This chapter contains the following topics.

- “VisualDSP++ Features”
- “Code Development Tools” on page 1-4
- “Connecting to a Debug Session” on page 1-6

VisualDSP++ Features

VisualDSP++ provides these features:

- **Extensive editing capabilities.** Create and modify source files by using multiple language syntax highlighting, drag-and-drop, bookmarks, and other standard editing operations. View files generated by the code development tools.

- **Flexible project management.** Specify a project definition that identifies the files, dependencies, and tools that you use to build projects. Create this project definition once or modify it to meet changing development needs.
VisualDSP++ Features

• **Easy access to code development tools.** Analog Devices provides these code development tools: C/C++ compiler, assembler, linker, splitter, and loader. Specify options for these tools by using dialog boxes instead of complicated command-line scripts. Options that control how the tools process inputs and generate outputs have a one-to-one correspondence to command-line switches. Define options for a single file or for an entire project. Define these options once or modify them as necessary.

• **Flexible project build options.** Control builds at the file or project level. VisualDSP++ enables you to build files or projects selectively, update project dependencies, or incrementally build only the files that have changed since the previous build. View the status of your project build in progress. If the build reports an error, double-click on the file name in the error message to open that source file. Then correct the error, rebuild the file or project, and start a debug session.

• **VisualDSP++ Kernel (VDK) support.** Add VDK support to a project to structure and scale application development. The Kernel page of the Project window enables you to manipulate events, event bits, priorities, semaphores, and thread types.

• **Flexible workspace management.** Create up to ten workspaces and quickly switch between them. Assigning a different project to each workspace enables you to build and debug multiple projects in a single session.

• **Easy movement between debug and build activities.** Start the debug session and move freely between editing, build, and debug activities.
VisualDSP++ reduces debugging time by providing these key features:

- **Easy-to-use debugging activities.** Debug with one common, easy-to-use interface for all processor simulators and emulators, or hardware evaluation and development boards. Switch easily between these targets.

- **Multiple language support.** Debug programs written in C, C++, 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.
Code Development Tools

C/C++ and assembly, and display the values of local variables or evaluate expressions (global and local) based on the current context.

- **Effective debug control.** Set breakpoints on symbols and addresses and then step through the program's execution to find problems in coding logic. Set watchpoints (conditional breakpoints) on registers, stacks, and memory locations to identify when they are accessed.

- **Tools for improving performance.** Use the trace, profile, and linear and statistical profiles to identify bottlenecks in your DSP application and to identify program optimization needs. Use plotting to view data arrays graphically. Generate interrupts, outputs, and inputs to simulate real-world application conditions.

Code Development Tools

Code development tools include:

- C/C++ compiler
- Run-time library with over 100 math, DSP, and C run-time library routines
- Assembler
- Linker
- Splitter
- Loader
- Simulator
- Emulator (must be purchased separately from VisualDSP++)
Features and Tools

These tools enable you to develop applications that take full advantage of your processor’s architecture.

The VisualDSP++ linker supports multiprocessing, shared memory, and memory overlays.

The code development tools provide these key features:

- **Easy-to-program C, C++, and assembly languages.** Program in C/C++, assembly, or a mix of C/C++ and assembly in one source. The assembly language is based on an algebraic syntax that is easy to learn, program, and debug.

- **Flexible system definition.** Define multiple types of executables for a single type of processor in one Linker Description File (.ldf). Specify input files, including objects, libraries, shared memory files, overlay files, and executables.

- **Support for overlays, multiprocessors, and shared memory executables.** The linker places code and resolves symbols in multiprocessor memory space for use by multiprocessor systems. The loader enables you to configure multiple processors with less code and faster boot time. Create host, link port, and PROM boot images.

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

For details about assembly syntax, refer to the *VisualDSP++ 5.0 Assembler and Preprocessor Manual.*
Connecting to a Debug Session

From the Windows Start menu, navigate to the VisualDSP++ environment via the Programs menu. After a second or two, the main VisualDSP++ window appears on the screen. When VisualDSP++ launches for the first time, it does not connect to any session (Figure 1-2).

Figure 1-2. VisualDSP++ Main Window

VisualDSP++ is able to connect to a number of different debug sessions, where each session has its own application and benefits. The session types available with VisualDSP++ are¹:

- **EZ-KIT Lite.** This is the dedicated USB connection between the PC and EZ-KIT Lite board. An EZ-KIT connection is simple to manage and is part of the EZ-KIT Lite. However, the connection is available with the kit only. Once your custom hardware board is available for development, you use an emulator session (described below) to connect to the custom hardware.

¹ Third-party software may add additional session types.
Features and Tools

- **Simulator.** This is a software model of the processor. Simulators offer unique advantages; the first is that no external hardware is required, a great benefit when using VisualDSP++ on the road. Furthermore, simulators offer a unique insight to the internal workings of the processor (pipelines, caches, and more), which is not possible with hardware-based sessions. The downside is that a simulator is several orders of magnitude slower than actual hardware. The software model simulates only the processor, making it difficult to accurately simulate a complex system that involves more than the processor.

VisualDSP++ includes two types of Blackfin simulators: a cycle-accurate interpreted simulator and a functional compiled simulator. A cycle-accurate simulator is a completely accurate model of the Blackfin processor and allows you to fully visualize the inner-workings of the processor. The compiled simulator sacrifices the detailed view but allows you to simulate much more quickly, millions of cycles per second, depending on the speed of your PC.

- **Emulator.** This is a JTAG emulator, the ideal device for connecting to hardware, giving the best performance and maximum flexibility. A separate module from the PC and EZ-KIT Lite, an emulator provides a high-bandwidth connection between the PC and device being debugged. Currently, Analog Devices offers USB- and PCI- based emulators. An emulator is required to connect to any non-EZ-KIT Lite hardware.

- **Legacy target.** This is a target created in VisualDSP++ 4.0 or a prior version.
Connecting to a Debug Session
2 BASIC TUTORIAL

This chapter contains the following topics.

- “Overview” on page 2-1
- “Exercise One: Building and Running a C Program” on page 2-3
- “Exercise Two: Modifying a C Program to Call an Assembly Language Routine” on page 2-15
- “Exercise Three: Plotting Data” on page 2-31
- “Exercise Four: Linear Profiling” on page 2-45

Overview

This basic tutorial demonstrates key features and capabilities of the VisualDSP++ integrated development and debugging environment (IDDE). The exercises use sample programs written in C and assembly for Blackfin processors.

You can use different Blackfin processors with only minor changes to the Linker Description File (.ldf) included with each project. VisualDSP++ includes basic Linker Description Files for each processor type in the ldf folder. For Blackfin processors, the folder’s default installation path is:

Program Files\Analog Devices\VisualDSP 5.0\Blackfin\ldf

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

The tutorial contains four exercises:

- **In Exercise One**, you start up VisualDSP++, build a project containing C source code, and profile the performance of a C function.

- **In Exercise Two**, you create a new project, create a Linker Description File to link with the assembly routine, rebuild the project, and profile the performance of the assembly language routine.

- **In Exercise Three**, you plot the various waveforms produced by a Finite Impulse Response (FIR) algorithm.

- **In Exercise Four**, you use linear profiling to examine the efficiency of the FIR algorithm used in Exercise Three. Using the collected linear profile data, you pinpoint the most time-consuming areas of the algorithm, which are likely to require hand tuning in the assembly language.

The ADSP-BF5xx Family Simulator and ADSP-BF533 processor are used for all exercises.

**Tip**: Become familiar with the VisualDSP++ toolbar buttons, shown in Figure 2-1. They are shortcuts for menu commands such as **Open** a file and **Run** a program. Toolbar buttons and menu commands that are not available for tasks are disabled and display in gray.

![Figure 2-1. VisualDSP++ Toolbar Buttons](image)

Tip: Become familiar with the VisualDSP++ toolbar buttons, shown in Figure 2-1. They are shortcuts for menu commands such as Open a file and Run a program. Toolbar buttons and menu commands that are not available for tasks are disabled and display in gray.
VisualDSP++ is a licensed software product. To run the software, you must have a valid license installed on your system. If you try to run VisualDSP++ and a license is not installed, a message window opens to let you add a license. For details about license management, see the VisualDSP++ 5.0 User’s Guide or VisualDSP++ online Help.

**Exercise One: Building and Running a C Program**

In this exercise, you:

- Start up the VisualDSP++ environment
- Open and build an existing project
- 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:

```
C:\Program Files\Analog Devices\VisualDSP 5.0\Blackfin\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, Analog Devices, VisualDSP++ 5.0, and VisualDSP++ Environment.

   If you are running VisualDSP++ for the first time, you will not be connected to a debug target. In VisualDSP++ 5.0, it is possible to edit and build your code without being connected to a debug target.
Exercise One: Building and Running a C Program

through a debug session. When you are ready to run and debug your program, you can quickly connect to a target and disconnect when you are finished. Doing so eliminates the overhead associated with the target connection, resulting in a smoother and more responsive experience.

- When you need to connect to a debug session, click the Connect to Target toolbar button ( ) or choose from the available sessions listed under Select Session in the Session menu. To create a debug session, select New Session from the Session menu. This will launch the Session Wizard, which is covered in more detail later. See Figure 2-7 on page 2-10.

If you have already run VisualDSP++ and the Reload last project at startup option is selected on the Project page of the Preferences dialog box (Settings -> Preferences), VisualDSP++ opens the last project that you worked on. To close this project, choose Close and then Project from the File menu, and then click No when prompted to save the project.

2. From the File menu, choose Open and then Project.

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 5.0\Blackfin\Examples\Tutorial\dot_product_c

   This path is based on the default installation.
4. Double-click the `dotprod` project (.dpj) file.

VisualDSP++ loads the project in the Project window, as shown in Figure 2-2. The environment displays messages in the Output window as it processes the project settings and file dependencies.

Figure 2-2. Project Loaded in the Project Window

The `dotprod` 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

5. From the Settings menu, choose Preferences to open the Preferences dialog box, shown in Figure 2-3.

![Preferences Dialog Box](image)

Figure 2-3. Preferences Dialog Box

6. On the General page, under General Preferences, ensure that the following options are selected.

   - Run to main after load
   - Load executable after build

7. Click OK to close the Preferences dialog box.

   The VisualDSP++ main window appears. You are now ready to build the project.
Step 2: Build the dotprodc Project

To build the dotprodc project:

1. From the Project menu, choose Build Project.

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

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

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

In this example (Figure 2-4) notice that the compiler detects an undefined identifier and issues the following error in the Build view of the Output window.

```
‘dotprodc_main.c’, line 115: cc0020: error: identifier “itn” is undefined
1 error detected in the compilation of ‘dotprodc_main.c’.
cc1009: fatal error: Compilation failed
Tool failed with exit/exception code: 1.
Build was unsuccessful
```

Figure 2-4. Example of Error Message
Exercise One: Building and Running a C Program

2. Double-click the error message 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**).

![Figure 2-5. Output Window and Editor Window](image)

The editor window in **Figure 2-5** shows that the integer variable declaration `int` has been misspelled as `itn`.

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 **File dotprod_main.c** from the **File -> Save** 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 of the **Output** window.

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

**Step 3: Run the Program**

In this procedure, you:

- 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 of the **Preferences** dialog box, the executable file `dotprodc.dxe` is downloaded to the target automatically. If you are not connected to a debug target, VisualDSP++ will prompt you to connect to one using an existing debug session (or to create a new debug session). Choose the **Select a session or create a new session** option as shown in **Figure 2-6**.

![No Session Selected](image.png)

**Figure 2-6. No Session Selected**
Exercise One: Building and Running a C Program

Click OK to create a new session. This will launch the Session Wizard, shown in Figure 2-7.

![Figure 2-7. Session Wizard](image)

The Session Wizard guides you through the process of specifying your debug session, including the processor, connection type, and platform.
Basic Tutorial

To set up the debug session:

1. On the Select Processor page, select Blackfin as the Processor family and select ADSP-BF533 under Choose a target processor. Click Next to continue.

2. On the Select Connection Type page, select Simulator, and click Next to continue.

3. On the Select Platform page, select ADSP-BF5xx Single Processor Simulator. In Session name, use the default session name or enter a more meaningful name of your choosing. Click Next to review your choices, and then click Finish.

VisualDSP++ closes the Session Wizard, automatically loads your project’s executable file (dotprodc.dxe), and advances to the main function of your code (see Figure 2-8).

4. Look at the information in the open windows.

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

   The Disassembly window displays the assembly 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 sets several breakpoints automatically. Most of the breakpoints set are used as part of advanced features of VisualDSP++. There are two breakpoints of
Exercise One: Building and Running a C Program

interest for this tutorial, one at the beginning and one at the end of code execution. Your breakpoint locations may differ slightly from those shown in the examples in this book.

5. From the Settings menu, choose Breakpoints to view the breakpoints set in your program. VisualDSP++ displays the Breakpoints dialog box, shown in Figure 2-9.
The two breakpoints of interest are set at these C program locations:

- at __lib_prog_term
- at main + 0x06

The Active page of the Breakpoints dialog box enables you to view, add, and delete breakpoints and browse for symbols. The Automatic page allows you to choose which breakpoints to set automatically each time your program is loaded. In the Disassem-

Figure 2-9. Breakpoints Dialog Box
Exercise One: Building and Running a C Program

bly and editor windows, double-clicking on a line of code toggles (adds or deletes) breakpoints. In the editor window, however, you must place the mouse pointer in the gutter before double-clicking.

These toolbar buttons set or clear breakpoints:

- Toggles a breakpoint for the current line
- Clears all breakpoints

6. 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 2-10) of the Output window.

Dot product [0] = 13273595
Dot product [1] = -49956078
Dot product [2] = 35872518

Figure 2-10. Results of the dotprodc Program

You are now ready to begin Exercise Two.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

In Exercise One, you built and ran a C program. In Exercise Two, you:

- Modify the C program to call an assembly language routine
- Create a Linker Description File to link with the assembly language routine
- Rebuild the project

The project files are almost 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 File menu, choose Close and then Project dotprodc.dpj to close the dotprodc project.

   Click Yes when prompted to close all open source windows.

   If you have modified your project during this session, you are prompted to save the project. Click No.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

2. From the File menu, choose New and then Project to open the Project Wizard, shown in Figure 2-11.

![Project Wizard](image)

Figure 2-11. Project Wizard – Select Type Page

3. Ensure that Standard application is selected in the Project types box.

4. In the Name field, type dot_product_asm.
5. Click the browse button \[\ldots\] to the right of the **Directory** field to open the **Browse For Folder** dialog box. Locate the *dot_product_asm* tutorial folder and click **OK**. By default this directory is in the following location.

```
C:\Program Files\Analog Devices\VisualDSP 5.0\Blackfin\Examples\Tutorial\dot_product_asm
```

6. Click **Next** to display the **Select Processor** page.

7. Verify that the **Processor type** is **ADSP-BF533** and the **Silicon Revision** is **Automatic**. Click **Next** to display the **Application Settings** page.

8. In **Select the project output type for your application**, verify that the **Executable (.dxe)** is selected. Uncheck the **Add template source code to the application** check box.

9. Click **Next** to display the **Add Startup Code/LDF** page.

10. Read the displayed text, and select the **Add an LDF and startup code** option. When this project is created, startup code that initializes and configures the processor will be added to the project, as will a Linker Description File that defines the target memory map and the placement of program sections within processor memory. The options available to configure the startup code and .ldf file are beyond the scope of this tutorial. Click **Finish**. The new project is created and is shown in the **Project** window.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

11. From the Project menu click Project Options to display the Project Options dialog box (Figure 2-12).

![Project Options Dialog Box – Project Page](image)

This dialog box enables you to specify project build information.

12. Take a moment to view the various pages in the Project Options dialog box by selecting them from the tree on the left (Project, General, Compile, Assemble, Link, Load, Pre-Build, and Post-Build). Each page provides options used to build the project.
13. On the **Project** page (Figure 2-12), verify that the values shown in Table 2-1 are specified.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>ADSP-BF533</td>
</tr>
<tr>
<td>Revision</td>
<td>Automatic</td>
</tr>
<tr>
<td>Type</td>
<td>Executable file</td>
</tr>
<tr>
<td>Name</td>
<td>dot_product_asm</td>
</tr>
<tr>
<td>Settings for configuration</td>
<td>Debug</td>
</tr>
</tbody>
</table>

These settings specify options for building an executable file for the ADSP-BF533 processor. The executable contains debug information, so you can examine program execution.

14. Click the **Compile** page to display the **General** sub-page, shown in Figure 2-13.

15. Specify these settings in the **Code Generation** group box:

   a. Select the **Enable optimization** check box to enable optimization.

   b. Select the **Generate debug information** check box, if it is not already selected, to enable debug information for the C source.

   c. Clear the **Generate assembly code annotations** check box.

These settings direct the C compiler to optimize code for the ADSP-BF533 processor. Because the optimization takes advantage of DSP architecture and assembly language features, some of the C debug information is not saved. Therefore, debugging is performed through debug information at the assembly language level.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

16. Click OK to apply changes and to close the Project Options dialog box.

You are now ready to add the source files to the project.
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 2-14) appears.

   Figure 2-14. 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 from the drop-down list.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

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.

5. Click the Rebuild All 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 3: Modify the Project Source Files

In this procedure, you:

- 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 2-15.

![Find Dialog Box](image.png)

Figure 2-15. Find Dialog Box: Locating Occurrences of /*
3. In the **Find what** box, type /*, and then click **Mark All**.

The editor bookmarks all lines containing /* and positions the cursor at the first instance of /* in the `extern int a_dot_c_asm` declaration. Bookmarks are indicated by a blue icon in the left margin (gutter).

4. Select the comment characters /* and use the **Ctrl+X** key combination to delete 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 changes 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, moving them to the end of the `a_dot_c` declaration. The `a_dot_c` declaration is now fully commented out, and the `a_dot_c_asm` declaration is no longer commented.

5. Press **F2** 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. Select the /* and press **Ctrl+X** to delete 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 is now calling the `a_dot_c_asm` routine instead of the `a_dot_c` function (previously called in Exercise One).

**Figure 2-16** shows the changes made in step 6.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

From the **File** menu, choose **Save** and then **File dotprod_main.c** to save the changes.

Place the cursor in the editor window. Then, from the **File** menu, choose **Close** and then **File dotprod_main.c** to close the **dotprod_main.c** file.

You are now ready to modify **dot_prod_asm.ldf**.

Figure 2-16. Modifying dotprod_main.c to Call a_dot_c_asm
Basic Tutorial

Step 4: Use the Expert Linker to Modify dot_prod_asm.ldf

In this procedure you:

- View the Expert Linker representation of the .ldf file that you created
- Modify the .ldf file to map 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. Build the project by performing one of these actions:
   - Click the Build Project button.
   - From the Project menu, choose Build Project.
4. Notice the error in the Output window (Figure 2-17).

![Figure 2-17. Output Window: Linker Error](image)

[Error li1060] The following symbols are referenced, but not mapped:
_a_dot_c_asm' referenced from Debug\dotprod_main.doj прогул

Linker finished with 1 error and 1 warning
cc0389: fatal error: Link failed
Tool failed with exit/exception code: 1.
Build was unsuccessful.
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

5. In the Project window, double-click the dot_prod_asm.ldf file. The Expert Linker window (Figure 2-18) opens with a graphical representation of your file.

6. Resize the window to expand the view and change the view mode. To display the tree view shown in Figure 2-18, right-click in the right pane, choose View Mode, and then choose Memory Map Tree.

The left pane (Input Sections) 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 Expert Linker has determined that the section is not mapped by the .ldf file.
The right pane (Memory Map) contains a representation of the memory segments that Expert Linker defined when it created the .ldf file.

Figure 2-18. Expert Linker Window

7. Map my_asm_section into the memory segment named MEM_PROGRAM as follows.

   In the Input Sections pane, open my_asm_section by clicking on the plus sign (+) in front of it. The input section expands to show that the linker macros $COMMAND_LINE_OBJECTS and $OBJECTS and the object file dotprod_func.doj have a section that has not been
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

mapped. In the Memory Map pane, expand MEM_L1_CODE and drag the icon from the Input Sections pane in front of $OBJECTS onto the L1_code output section under MEM_L1_CODE.

As shown in Figure 2-19, the red X should no longer appear because the my_asm_section section has been mapped.

![Figure 2-19. Dragging $OBJECTS onto the L1_code Output Section](image)

8. From the Tools menu, choose Expert Linker and Save to save the modified .ldf 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 save automatically it.

You are now ready to rebuild and run the modified project.
Step 5: Rebuild and Run dot_product_asm

To run dot_product:

1. Build the project by performing one of these actions:
   - Click the Build Project button.
   - From the Project menu, choose Build Project.

At the end of the build, the Output window displays this message in the Build view:

“Build completed successfully.”

VisualDSP++ loads the program, runs to main, and displays the Output, Disassembly, and editor windows (shown in Figure 2-20).
Exercise Two: Modifying a C Program to Call an Assembly Language Routine

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 of the Output window. When the program stops running, the message “Halted” appears in the status bar at the bottom of the VisualDSP++ main window. The results, shown below, are identical to the results obtained in Exercise One.

- Dot product [0] = 13273595
- Dot product [1] = -49956078
- Dot product [2] = 35872518

You are now ready to begin Exercise Three.
Exercise Three: Plotting Data

In this exercise, you:

- Load and debug a prebuilt program that applies a simple Finite Impulse Response (FIR) filter to a buffer of data
- Use the VisualDSP++ plotting engine to view the different data arrays graphically, both before and after running the program

Step 1: Load the FIR Program

To load the FIR program:

1. Keep the Disassembly window and Console page (of the Output window) open, but close all other windows.
2. From the File menu, choose Load Program or click . The Open a Processor Program dialog box appears.
3. Select the FIR program to load as follows.
   a. Open the Analog Devices folder and double-click:

      VisualDSP 5.0\Blackfin\Examples\Tutorial\fir

   b. Double-click the Debug subfolder.
   c. Double-click FIR.DXE to load the program.

      If VisualDSP++ does not open an editor window (shown in Figure 2-21), right-click in the Disassembly window and select View Source.
Exercise Three: Plotting Data

4. Look at the source code of the FIR program (fir_test.c).
   You can see two global data arrays: (IN and OUT).
   You can also see one function, fir, that operates on these arrays.
You are now ready to open a plot window.

Figure 2-21. Loading the FIR Program
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 2-22.

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

![Plot Configuration Dialog Box]

Figure 2-22. Plot Configuration Dialog Box
Exercise Three: Plotting Data

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

3. Enter two data sets to plot by using the values in Table 2-2.

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

<table>
<thead>
<tr>
<th>Box</th>
<th>Input Data Set</th>
<th>Output Data Set</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Input</td>
<td>Output</td>
<td>Data set</td>
</tr>
<tr>
<td>Memory</td>
<td>BLACKFIN Memory</td>
<td>BLACKFIN Memory</td>
<td>Data memory</td>
</tr>
<tr>
<td>Address</td>
<td>IN</td>
<td>OUT</td>
<td>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.</td>
</tr>
<tr>
<td>Count</td>
<td>128</td>
<td>128</td>
<td>The array is 260 elements long, but you are plotting the first 128 elements.</td>
</tr>
<tr>
<td>Stride</td>
<td>1</td>
<td>1</td>
<td>The data is contiguous in memory.</td>
</tr>
<tr>
<td>Data</td>
<td>short</td>
<td>short</td>
<td>Input and Output are arrays of int values.</td>
</tr>
</tbody>
</table>

After entering each data set, click Add to add the data set to the Data sets list on the left of the dialog box.

The Plot Configuration dialog box should now look like the one in Figure 2-23.
4. Click **OK** to apply the changes and to open a plot window with these data sets.

The plot window now displays the two arrays. By default, the simulator initializes memory to zero, so the **Output** data set appears as one horizontal line, shown in **Figure 2-24**.
Exercise Three: Plotting Data

Resizing the plot window changes the scale on the x and y axis.

5. Right-click in the plot window and choose Modify Settings. On the General page of the Plot Settings dialog box, in the Options group box, select Legend and click OK to display the legend box. Click OK again.

Figure 2-24. Plot Window: Before Running the FIR Program
Step 3: Run the FIR Program and View the Data

To run the FIR program and view the data:

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

When the program halts, you see the results of the FIR filter in the Output array. The two data sets are visible in the plot window, as shown in Figure 2-25.

![Figure 2-25. Plot Window After Running the FIR Program to Completion](image)

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

2. Click the left mouse button inside the plot window and drag the mouse to create a rectangle around the area of interest. Then release the mouse button to magnify the selected region.
Figure 2-26 shows the selected region, and Figure 2-27 shows the magnified result.

Figure 2-26. Plot Window: Selecting a Region to Magnify

Exercise Three: Plotting Data
To return to the previous view (before magnification), right-click in the plot window and choose Reset Zoom from the pop-up menu. You can view individual data points in the plot window by enabling the data cursor, as explained in the next step.

3. Right-click inside the plot window and choose Data Cursor from the pop-up menu. Move to each data point in the current data set by pressing and holding the keyboard’s left or right arrow key. To switch data sets, press the keyboard’s up or down arrow key. The value of the current data point appears in the lower-left corner of the plot window, as shown in Figure 2-28.

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

Next you will look at data sets in the frequency domain.

5. Right-click in the plot window and choose Modify Settings to open the Plot Settings dialog box.
Exercise Three: Plotting Data

Figure 2-28. Plot Window: Using the Data Cursor Feature
6. Complete these steps:
   a. Click the **Data Processing** tab to display the **Data Processing** page, shown in **Figure 2-29**.

![Data Processing Page](image)

*Figure 2-29. Data Processing Page*
Exercise Three: Plotting Data

b. In the Data Sets box, ensure that Input (the default) is selected. In the Data Process box, choose FFT Magnitude.

c. In the Sample rate (Hz) box, type 10000.

d. In the Data Sets box, select Output. In the Data Process box, choose FFT Magnitude.

e. Click OK to exit the Plot Settings dialog box.

VisualDSP++ performs a Fast Fourier Transform (FFT) on the selected data set before it is plotted. The FFT enables you to view the signal in the frequency domain, as shown in Figure 2-30.

Figure 2-30. FFT Performed on a Selected Data Set
Now, complete the following steps to look at the FIR filter’s response in the frequency domain.

1. From the View menu, choose Debug Windows and Plot. Then choose New to open the Plot Configuration dialog box.

2. Set up the Filter Frequency Response plot by completing the Plot and Data Setting group boxes as shown in Figure 2-31.

![Figure 2-31. Filter Frequency Response Data Set](image)

3. Click Add to add the data set to the Data sets box.

4. Click OK to apply the changes and to open the plot window with this data set.

5. Right-click in the plot window and choose Modify Settings to open the Plot Settings dialog box.
Exercise Three: Plotting Data

6. Click the Data Processing tab to display the Data Processing page, shown in Figure 2-29 on page 2-41. Complete this page as follows.

   a. In the Data Sets box, select h.
   b. In the Data Process box, choose FFT Magnitude.
   c. In the Sample rate (Hz) box, type 10000.
   d. Click OK to exit the Data Processing page.

VisualDSP++ performs a Fast Fourier Transform (FFT) on the selected data set, and enables you to view the filter response plot in the frequency domain, as shown in Figure 2-32.

![Filter Frequency Response Plot](image)

Figure 2-32. Filter Frequency Response Plot

This plot shows that the low-pass FIR filter cuts off all frequency components above 4,000 Hz. When you apply a low-pass filter to the input signal, the resulting signal has no output above 4,000 Hz.

You are now ready to begin Exercise Four.
Exercise Four: Linear Profiling

In this exercise, you:

- Load and debug the FIR program from the previous exercise
- Use linear profiling to evaluate the program’s efficiency and to determine where the application is spending the majority of its execution time in the code

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

- You use linear profiling with a simulator. The count in the Linear Profiling window is incremented every time an assembly instruction is executed.
- You use statistical profiling with a JTAG emulator connected to a processor target. The count in the Statistical Profiling window is based on random sampling of the program counter.

Step 1: Load the FIR Program

To load the FIR program:

1. Close all open windows except 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.
Exercise Four: Linear Profiling

3. Select the program to load as follows.
   a. Open the Analog Devices folder and double-click:
      VisualDSP 5.0\Blackfin\Examples\Tutorial\fir
   b. Double-click the Debug subfolder.
   c. Double-click fir.dxe to load and run the FIR program.

      If VisualDSP++ does not open an editor window (shown in Figure 2-34), right-click in the Disassembly window and select View Source.

You are now ready to set up linear profiling.

Step 2: Open the Profiling Window

To open the Linear Profiling window:

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

Figure 2-33. Setting Up Linear Profiling for the FIR Program

The Linear Profiling window opens without any data.
2. Click in the profiling window's title bar and then drag and drop the window to the top of the VisualDSP++ main window, as shown in Figure 2-34. You will have a better view of the profile data.

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

You are now ready to collect and examine linear profile data.
Step 3: Collect and Examine the Linear Profile Data

To collect and examine the linear profile data:

1. Press F5 or click \(\text{ }\) to run to the end of the program.

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

2. Examine the results of your linear profiling session.

The Linear Profiling window is divided into two three-column panes.

The left pane shows the results of the profile data. You can see the percentages of total execution time consumed, by function and by address.

Double-clicking a line with a function enables you to display the source file that contains that function. For example, double-click the \text{fir} function (and respond to the prompt) to display the assembly source file (\text{fir.asm}) in the right pane, as shown in Figure 2-35.

Figure 2-35. Linear Profiling Results, FIR Program Performance Analysis

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

| **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 window 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 window and by selecting View Sample Count from the pop-up 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 `PC[xxx]`, where `xxx` is the address of the instruction. If the instructions are part of an assembly file, the execution unit is either an assembly function or the assembly file followed by the line number in parentheses. |
Exercise Four: Linear Profiling

In Figure 2-35 on page 2-48 the left pane shows that the fir function consumes over 93% of the total execution time. The right (source) pane, shown in Figure 2-36, displays the percentage that each line in the fir function consumes.

<table>
<thead>
<tr>
<th>Line</th>
<th>Source Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>66</td>
<td>.firm</td>
<td>Address of the filter buffer initial...</td>
</tr>
<tr>
<td>67</td>
<td>.firm</td>
<td>Address of the filter buffer...</td>
</tr>
<tr>
<td>68</td>
<td>.firm</td>
<td>Address of the delay line buffer...</td>
</tr>
<tr>
<td>69</td>
<td>.firm</td>
<td>Number of filter coefficients...</td>
</tr>
<tr>
<td>70</td>
<td>.firm</td>
<td>Output buffer initial...</td>
</tr>
<tr>
<td>71</td>
<td>.firm</td>
<td>Initialize I2 to the...</td>
</tr>
<tr>
<td>72</td>
<td>.firm</td>
<td>Filter coeff. array...</td>
</tr>
<tr>
<td>73</td>
<td>.firm</td>
<td>Start of the delay line...</td>
</tr>
<tr>
<td>74</td>
<td>.firm</td>
<td>Delay line buffer...</td>
</tr>
<tr>
<td>75</td>
<td>.firm</td>
<td>Delay line buffer...</td>
</tr>
<tr>
<td>76</td>
<td>.firm</td>
<td>I3+P1,</td>
</tr>
<tr>
<td>77</td>
<td>.firm</td>
<td>I1+P1,</td>
</tr>
<tr>
<td>78</td>
<td>.firm</td>
<td>I0+P2,</td>
</tr>
<tr>
<td>79</td>
<td>.firm</td>
<td>I1+P2,</td>
</tr>
<tr>
<td>80</td>
<td>.firm</td>
<td>I2+P2,</td>
</tr>
<tr>
<td>81</td>
<td>.firm</td>
<td>I1+P2,</td>
</tr>
<tr>
<td>82</td>
<td>.firm</td>
<td>I0+P2,</td>
</tr>
<tr>
<td>83</td>
<td>.firm</td>
<td>I3+P1,</td>
</tr>
<tr>
<td>84</td>
<td>.firm</td>
<td>I2+P2,</td>
</tr>
<tr>
<td>85</td>
<td>.firm</td>
<td>I1+P2,</td>
</tr>
<tr>
<td>86</td>
<td>.firm</td>
<td>I0+P2,</td>
</tr>
<tr>
<td>87</td>
<td>.firm</td>
<td>Check if the number of taps is greater than zero...</td>
</tr>
<tr>
<td>90</td>
<td>.firm</td>
<td>As the filter coefficient...</td>
</tr>
<tr>
<td>91</td>
<td>.firm</td>
<td>Initialize the filter...</td>
</tr>
<tr>
<td>92</td>
<td>.firm</td>
<td>Address of the input...</td>
</tr>
<tr>
<td>93</td>
<td>.firm</td>
<td>If I2+J3 &gt; 0,</td>
</tr>
<tr>
<td>94</td>
<td>.firm</td>
<td>Make the filter taps...</td>
</tr>
<tr>
<td>95</td>
<td>.firm</td>
<td>Location where zero...</td>
</tr>
<tr>
<td>96</td>
<td>.firm</td>
<td>Location where zero...</td>
</tr>
<tr>
<td>97</td>
<td>.firm</td>
<td>Location where zero...</td>
</tr>
<tr>
<td>98</td>
<td>.firm</td>
<td>Location where zero...</td>
</tr>
<tr>
<td>99</td>
<td>.firm</td>
<td>Location where zero...</td>
</tr>
<tr>
<td>100</td>
<td>.firm</td>
<td>Set the last filter...</td>
</tr>
<tr>
<td>101</td>
<td>.firm</td>
<td>Force the number of filters...</td>
</tr>
</tbody>
</table>

Figure 2-36. Linear Profile Data for fir.asm

You have now completed the Basic Tutorial.
3 ADVANCED TUTORIAL

This chapter contains the following topics.

- “Overview” on page 3-1
- “Exercise One: Using Profile-Guided Optimization” on page 3-2
- “Exercise Two: Using Background Telemetry Channel” on page 3-23

Overview

This tutorial demonstrates advanced features and techniques that you can use in the VisualDSP++ Integrated Development and Debugging Environment (IDDE). The exercises use sample programs written in C and assembly for Blackfin processors.

- In Exercise One: Using Profile-Guided Optimization, you build a project with PGO support, create PGO files, compile the project without using the information in the PGO files, recompile the project by using the PGO files to optimize the build, check the PGO results, and compare execution times.

- In Exercise Two: Using Background Telemetry Channel, you run two demos that demonstrate BTC functionality.

The ADSP-BF53x Family Simulator and ADSP-BF533 processor are used in Exercise One. The ADSP-BF533 EZ-KIT Lite and an HPPCI-ICE, HPUSB-ICE, or debug agent connection are used in Exercise Two.
Profile-guided optimization (PGO) is an optimization technique that uses collected profile information to guide the compiler optimizer’s decisions.

Traditionally, a compiler compiles each function only once and attempts to produce generated code that will perform well in most cases. The compiler has to make decisions about the best code to generate. For example, given an `if…then…else` construct, the compiler has to decide whether the most common case is the `then` or the `else`. You can offer crude guidelines—compile for speed or compile for space—but, usually, the compiler has to make a default decision.

With PGO, the compiler makes these decisions based on data collected during previous executions of the generated code. This process involves the following steps.

1. Compiling the application to collect profile information
2. Running the application in a simulator session by using representative data sets
   - The simulator accumulates profile data indicating where the application spends most of its time.
3. Recompiling the application by using the collected profile data
   - The compiler uses the collected information rather than the application’s default behavior to make decisions about the relative importance of parts of the application.

The profile data collected from a simulator run is stored in a file with a `.pgo` suffix. You can process multiple data sets to cover the spectrum of potential data and create a separate `.pgo` file for each data set. The recompilation stage can accept multiple `.pgo` files as input.
You must complete these basic steps to use PGO:

1. Build the application with PGO support.
2. Set up one or more streams in the simulator to provide a set of data inputs that represent what the application would see in a real target environment.
3. Tell the simulator to produce a .pgo file with a specified file name.
4. Load and run the application to produce the .pgo file.
5. Rebuild the application and pass all .pgo files to the compiler, which uses the generated PGO results to optimize the application.

In this exercise, you:

- Load the PGO example project in the VisualDSP++ environment
- Create data sets for profile-guided optimization
- Attach input streams to the data sets
- Create .pgo files by executing the project with the data sets as input
- Recompile the project by using the .pgo files to optimize the build
- Run the optimized version of the project with the same data sets as input
- Compare the execution times of all three executions

The files used for this exercise are in the pgo folder. The default installation path of this folder is:

Program Files\Analog Devices\VisualDSP++ 5.0\Blackfin\Examples\Tutorial\pgo
Exercise One: Using Profile-Guided Optimization

Step 1: Load the Project

To open a VisualDSP++ project:

1. Start VisualDSP++ and connect to an ADSP-BF533 simulator session. For information about connecting to a session, refer to “Step 1: Start VisualDSP++ and Open a Project” on page 2-3.

2. Open the PgoExample.dpj project. For details about opening projects, see “Step 1: Start VisualDSP++ and Open a Project” on page 2-3.

This project contains a C file, PgoExample.c, which is in the project’s Source Files subdirectory. When you run the program, it reads data from an address and counts the number of even and odd values. This counting is done with an if...then...else statement. If the majority of values read are odd, the program will spend most of its time in the then... branch. If the majority of values are even, the program will spend most of its time in the else... branch. Normally,
Advanced Tutorial

the compiler has no way of knowing which branch will be taken more often. By using PGO, the compiler can determine which branch is used most often and optimize the next build.

This project also contains a Visual Basic script that demonstrates how to use the VisualDSP++ Automation API to perform PGO. The automation functionality is beyond the scope of this tutorial. Refer to online Help for more information about automation.

Three data files are used as input to the C program. These simple text files contain lists of values.

- **Dataset_1.dat** has 128 even values (50%) and 128 odd values (50%).
- **Dataset_2.dat** has 192 even values (75%) and 64 odd values (25%).
- **Dataset_3.dat** has 256 even values (100%) and 0 odd values (0%).

To view these files, choose the **Open** command on the **File** menu in VisualDSP++. The two possible values in all three files are either 0x01 or 0x02. Each file contains 256 values.

In this exercise, assume that this program will be used in the real world, and that you can expect a similar distribution of values as input from the real world.

By looking at the C code and the potential input, you can easily see that the executed program will spend more time in the **else** branch than in the **then** branch. Without using PGO, the com-
Exercise One: Using Profile-Guided Optimization

Compiler cannot make this same conclusion. By default, it will expect the then... branch to be executed most frequently and will compile the code without optimizing execution time.

Since the example program and input are very simple, you could fix the problem by making a few minor changes to the code. Manually tweaking a large program to speed up execution time, however, would take far too long, and you would have to analyze sample input on your own. PGO provides a quick and easy way to enable the compiler to make these adjustments for you.

Step 2: Configure a Data Set

The first step in the PGO process is to create a data set—a collection of sample input for the program being optimized. A data set feeds the input into the executing program, and this input causes the program to be executed along certain paths. Some paths will be used more often than others. This information is recorded by the simulator and stored in a .pgo file for the compiler to use later for optimization. The most commonly used paths will be optimized to run quickly, and less common paths will run more slowly.
To create the first of three data sets for this exercise:

1. From the Tools menu, choose PGO and then Manage Data Sets, as shown in Figure 3-1.

Figure 3-1. Manage Data Sets Menu Option
Exercise One: Using Profile-Guided Optimization

The Manage Data Sets dialog box (Figure 3-2) is displayed.

![Manage Data Sets Dialog Box](image)

Figure 3-2. Manage Data Sets Dialog Box

This dialog box is where you manage data sets. Note the Optimization level slider bar. This control allows you to customize your optimization. Moving the slider all the way to the left enables you to build as small an executable as possible, but may sacrifice execution speed. Moving the slider all the way to the right enables you to build a fast executable, with a potential space tradeoff. Placing the
slider between the two extremes provides varying ratios of space versus speed optimization. For this exercise, make sure the slider is positioned all the way to the right.

2. Click the New button to open the Edit Data Set dialog box, shown in Figure 3-3.

![Edit Data Set Dialog Box](image)

Figure 3-3. Edit Data Set Dialog Box
Exercise One: Using Profile-Guided Optimization

3. Replace the default Data set name with a more descriptive name. Since the first data file contains an equal number of even and odd values, use a name such as 50% Even - 50% Odd.

4. Specify the Output filename (where the optimization information produced by this data set will be saved). Optimization information is saved in files with a .pgo suffix.

Type in a file name such as dataset_1.pgo. The file will be saved in the project directory. To save the files elsewhere, type in a full path name. You can use Command line arguments for more advanced control of the data set, but they are not covered in this tutorial.

For more information about command-line arguments, see the VisualDSP++ 5.0 C/C++ Compiler and Library Manual for Blackfin Processors.

Now you have to attach an input stream to this data set.
Advanced Tutorial

Step 3: Attach an Input Stream

In this step you attach an input stream to the data set.

1. Click the New button on the Edit Data Set Dialog Box (Figure 3-3) to open the Edit PGO Stream dialog box, shown in Figure 3-4.

![Edit PGO Stream Dialog Box](image)

Figure 3-4. Edit PGO Stream Dialog Box

An input stream maps a data file to a destination device. In this exercise, the input streams map the three data files to the simulator.
Exercise One: Using Profile-Guided Optimization

The input stream provides the program with input as needed during execution.

For more information about streams, see the “Debugging” chapter in the VisualDSP++ 5.0 User’s Guide.

2. Complete the Input Source File group box as described in Table 3-1.

Table 3-1. Input Source File Group Box Settings

<table>
<thead>
<tr>
<th>Field/Control</th>
<th>Action/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>Specify a file name by clicking the file browse button and selecting the input source file dataset_1.dat from the pgo directory.</td>
</tr>
<tr>
<td>Format</td>
<td>The data in this file is in hexadecimal format, so leave the format setting as is.</td>
</tr>
<tr>
<td>Rewind on reset or restart</td>
<td>Select this option. When you run a program with an input stream, the program may or may not work through all of the data in the stream. If the program encounters a reset or restart event before working through the entire data stream and this option is enabled, the next execution starts at the beginning of the input stream. Otherwise, execution continues where it left off.</td>
</tr>
<tr>
<td>Circular</td>
<td>Select this option. It allows the program to read through an input stream many times during a single execution.</td>
</tr>
</tbody>
</table>
3. In the **Destination Device** group box, specify where the data from the input stream is sent. Refer to **Table 3-2**.

**Table 3-2. Destination Device Group Box Settings**

<table>
<thead>
<tr>
<th>Field/Control</th>
<th>Action/Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>This field lets you specify a peripheral in another processor as the destination device. For this tutorial, you are connected to a single processor session, so this field is disabled.</td>
</tr>
<tr>
<td>Device</td>
<td>This field lets you choose any stream device supported by the simulator target as the destination. Devices can include a memory address or various peripherals. Available devices depend on the processor you are using. For more information on devices, see the hardware manual for your processor. The program reads the input streams from memory, so leave this field as it is.</td>
</tr>
<tr>
<td>Address</td>
<td>Specify where in memory the input will be sent. Since the program in this exercise reads data from address 0xFFD00000 (refer to PgoExample.c), enter this value.</td>
</tr>
</tbody>
</table>
Exercise One: Using Profile-Guided Optimization

The completed dialog box should now look like Figure 3-5.

![Figure 3-5. A Configured PGO Stream](image)

Figure 3-5. A Configured PGO Stream
4. Click **OK** to return to the **Edit Data Set** dialog box. The dialog box with your configured data set should match Figure 3-6.

![Figure 3-6. A Configured Data Set](image)

5. Click **OK** to save the data set and close the dialog box.

You now have to create the remaining two data sets.
Exercise One: Using Profile-Guided Optimization

Step 4: Configure Additional Data Sets

To create the remaining two data sets, you can repeat the steps used to create the first data set and substitute the appropriate files, or use the Copy button.

The following steps explain how to use the Copy button to create a data set.

1. Highlight the 50% Even – 50% Odd data set, and click the Copy button.

   The Edit Data Set dialog box opens with the information for the 50% Even – 50% Odd data set. Clicking the OK button makes a copy of the 50% Even – 50% Odd data set. For this exercise, however, you will edit the data set.

2. In the Data set name field, specify an appropriate name for the new data set.

   The second input source file contains three times as many even values as odd values, so use a name such as 75% Even – 25% Odd.

3. In the Output filename field, type the name dataset_2.pgo to save the .pgo file in the project directory.

   To save the file elsewhere, click the file browse button ( ) to specify a full path.

4. In the Input streams box, highlight the dataset_1.dat Input File and click the Edit button.

5. Click the file browse button ( ) to change the Input Source File from dataset_1.dat to dataset_2.dat.

6. Click the OK button to return to the Edit Data Set dialog box.

   The second data set is now complete.
7. Click the **OK** button to return to the **Manage Data Sets** dialog box.

8. Create the third data set from scratch or modify a copy of one of the existing data sets.

   Make sure that you use the `dataset_3.pgo` and `dataset_3.dat` files. The third data set contains all even values, so give it a name such as *100% Even - 0% Odd*. When you are finished, expand the three data sets listed in the Manage Data Sets dialog box and compare them with the data sets in Figure 3-7.

![Manage Data Sets](image)

**Figure 3-7. Expanded Data Sets**
Exercise One: Using Profile-Guided Optimization

If your data sets match those in Figure 3-7, you have the data sets needed to optimize the program.

9. Click OK to save the data sets and close the dialog box.

You are now ready to create .pgo files.

Step 5: Create PGO Files and Optimize the Program

Now that you have configured the data sets, you are ready to optimize your program.

From the Tools menu, choose PGO and then Execute Data Sets, as shown in Figure 3-8.

Figure 3-8. Execute Data Sets Menu Option

Several things happen during the execute process. First, the project is built with the -pguide switch, which enables the collection of the PGO data that is later fed back into the compiler. The compiler makes default assumptions about which sections of code will be most commonly executed. Next, the resulting executable is run once with each data set. While the program is running, the simulator monitors the paths of execution...
through the program, and the number of cycles used in the execution. As stated before, this information is stored in the .pgo file that you specified when creating each data set.

Once the program has been run with each data set, the project is recompiled. This time, however, the compiler uses the information found in the .pgo files to optimize the resulting executable. This optimized executable is then run with the input provided by each data set, and again, the simulator monitors each execution.

You are now ready to examine the results of the optimization.

Step 6: Compare Execution Times

When the execution is completed, an XML report of the PGO optimization results is generated and displayed in a browser window. This file is in the pgo\debug folder and is named PgoReport.* date and time.*xml (for example, PgoReport.20031027145428.xml).

At the top of the report is a header, shown in Figure 3-9.

Profile Guided Optimization Results

Generated on Thu Nov 11 13:40:02 2004
Application: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\debug\pgoExample.dsp
Project: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgo\pgoExample.dsp
Optimization level: 100
Average cycle reduction: 15.02%

Figure 3-9. PGO Results – Report Header

The header provides basic information such as the project name, location, and when the report was generated. Also listed is the optimization level (which you specified with the slider bar in the Manage Data Sets dialog.
Exercise One: Using Profile-Guided Optimization

box, Figure 3-2 on page 3-8), and an average result. The Average result is the difference in total cycle counts on all executions from before and after optimization.

The Average result obtained on your machine may vary slightly from the result shown in Figure 3-9.

The header is followed by information about each data set (see Figure 3-10).

<table>
<thead>
<tr>
<th>Data Set: 50% Even - 50% Odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command line:</td>
</tr>
<tr>
<td>Input stream: File: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgp\dataset_1.dat</td>
</tr>
<tr>
<td>Device: 0xFF00000-0xFF00000</td>
</tr>
<tr>
<td>PGO output: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgp\dataset_1.pgo</td>
</tr>
<tr>
<td>Before optimization: 7875 cycles</td>
</tr>
<tr>
<td>After optimization: 7575 cycles</td>
</tr>
<tr>
<td>Cycle reduction: 3.20%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Set: 75% Even - 25% Odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command line:</td>
</tr>
<tr>
<td>Input stream: File: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgp\dataset_3.dat</td>
</tr>
<tr>
<td>Device: 0xFF00000-0xFF00000</td>
</tr>
<tr>
<td>PGO output: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgp\dataset_3.pgo</td>
</tr>
<tr>
<td>Before optimization: 8573 cycles</td>
</tr>
<tr>
<td>After optimization: 7575 cycles</td>
</tr>
<tr>
<td>Cycle reduction: 12.17%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Set: 100% Even - 0% Odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command line:</td>
</tr>
<tr>
<td>Input stream: File: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgp\dataset_1.dat</td>
</tr>
<tr>
<td>Device: 0xFF00000-0xFF00000</td>
</tr>
<tr>
<td>PGO output: D:\Program Files\VisualDSP 4.0\Blackfin\Examples\Tutorial\pgp\dataset_3.pgo</td>
</tr>
<tr>
<td>Before optimization: 9399 cycles</td>
</tr>
<tr>
<td>After optimization: 7575 cycles</td>
</tr>
<tr>
<td>Cycle reduction: 24.09%</td>
</tr>
</tbody>
</table>

Figure 3-10. PGO Results – Data Sets

The file information, including the Data Set file name, Input stream file name, and PGO output file name, is listed first. Then the results of optimization are shown. The number of cycles needed to run the original
build with this data set (Before optimization) is followed by the number of cycles needed to run this data set on the optimized build (After optimization). Note that the number of cycles may vary on different machines.

Finally, the percent difference between the two builds (Result) is listed. A positive percentage indicates that the optimized build ran faster than the original build.

The Execution Output section of the log appears first. Figure 3-11 shows selections from the execution output.

This information is the output that appeared in the Console view of the Output window while the PGO was running. The output includes the basic events that occurred during execution.
Exercise One: Using Profile-Guided Optimization

The Build Output section appears next at the bottom of the report. This section contains build output for each build. Figure 3-12 shows a build output sample.

```
Pre-Optimization Build Output

---------------Configuration: PgoExample - Debug---------------
\PgoExample.c
Linking...
Build completed successfully.
```

```
Post-Optimization Build Output

---------------Configuration: PgoExample - Debug---------------
\PgoExample.c
Linking...
Build completed successfully.
```

Figure 3-12. PGO Results – Build Output Sample

This information is the output that was displayed in the Build view of the Output window while the PGO was running.

This output information shows how effective PGO can be. As shown in Figure 3-9 on page 3-19, the optimized executions used approximately 18% fewer cycles than the original executions. The gain in performance is significant, especially given the ease with which it was accomplished.

You are now ready to begin Exercise Two.
Exercise Two: Using Background Telemetry Channel

A background telemetry channel (BTC) enables you to exchange data between a host and target application without halting the processor. This mechanism provides real-time visibility into a running program. Uses for a BTC include:

- Monitoring program status without halting the processor
- Viewing algorithm output in real time
- Injecting data into a program
- Streaming real-time data to a file (data logging)
- Providing I/O, either standard or user-defined

In this exercise, you:

- Run the BTC Assembly demo, designed to demonstrate the basic functionality of BTC.
- Run the BTC FFT demo, which demonstrates the transfer of data from the Blackfin EZ-KIT Lite over background telemetry channels.

Running the BTC Assembly Demo

The BTC assembly demo is designed to demonstrate the basic functionality of BTC. The program defines several BTCs to allow the transfer of data over the BTC interface while the processor is running. For example, one channel counts the number of interrupts that have occurred, and another counts the number of times a push button is pressed. See the Btc_AsmDemo.asm header for more details. You will use the BTC Memory window in the IDDE to view the data in each channel.
Figure 3-13 provides an overview of data transfer over the BTC interface in the BTC assembly demo.

**Figure 3-13. Data Transfer in the Assembly Demo**

**Step 1: Load the BTC_AsmDemo Project**

1. Start VisualDSP++ and connect to the ADSP-BF533 EZ-KIT Lite.

   If you use the Session Wizard, select ADSP-BF533 for a processor, then choose either Emulator or EZ-KIT Lite for the session type, depending on how you want to connect to the kit (Emulator: HPPCI-ICE or HPUSB-ICE, EZ-KIT Lite: Debug Agent).
If you chose Emulator for the session type you need to choose either ADSP-BF533 via HPPCI-ICE or ADSP-BF533 via HPUSB-ICE as the platform, depending on which emulator type is available.

If you chose EZ-KIT Lite as the session type, choose ADSP-BF533 EZ-KIT Lite via Debug Agent as the platform type.

2. Open the Btc_AsmDemo.dpj project, under Analog Devices in:

VisualDSP 5.0\Blackfin\Examples\ADSP-BF533 EZ-Kit
Lite\Background_Telemetry\AsmDemo

For details about loading a project, see “Step 1: Start VisualDSP++ and Open a Project” on page 2-3.

You are now ready to examine BTC commands.

Step 2: Examine the BTC Commands

1. Open the Btc_AsmDemo.asm file by double-clicking on it in the Project window.

2. Scroll down to the section labeled BTC Definitions in the comments (see Figure 3-14). Note the five channels defined..

```
// BTC Definitions
BTC_MAP_BEGIN
  // Channel Name, Starting Address, Length
BTC_MAP_ENTRY("Timer Interrupt Counter", timerCounter, 0x00004)
BTC_MAP_ENTRY("PF10 Counter", pf10Counter, 0x00004)
BTC_MAP_ENTRY("PF11 Counter", pf11Counter, 0x00004)
BTC_MAP_ENTRY("256 byte channel (PF11)", pf11Array, 0x00100)
BTC_MAP_ENTRY("4k byte channel", array1, 0x01000)
BTC_MAP_END
```

Figure 3-14. BTC Channel Definitions
Exercise Two: Using Background Telemetry Channel

3. Scroll down to the main program.

Twenty-four lines below the _main: label is the command to initialize BTC, call _btc_init; (shown in Figure 3-15).

```c

// main program
(section program:
.extern ldf_stack_end;
.globl _main;
_main:
  // initialize the stack pointer
  sp.h = ldf_stack_end;
  sp.l = ldf_stack_end;

  // setup the EVT
  [--sp] = rets;
  call initVectorRegs;
  rets = [sp++];
  // init LEDs
  [--sp] - rets;
  call initLEDs;
  rets = [sp++];
  // setup the Core Timer
  [--sp] - rets;
  call initCoreTimer;
  rets = [sp++];
  // setup the Programmable Flags
  [--sp] = rets;
  call initProgFlags;
  rets = [sp++];

  // initialize the BTC
  [--sp] = rets;
  call _btc_init;
  rets = [sp++];
```

Figure 3-15. BTC Initialize Command

For more information about _btc_init, refer to the VisualDSP++ Help.

In this example, the call _btc_poll command is placed in the EVT15_LOOP, defined below main and shown in Figure 3-16.
Refer to the online Help for more information about _btc_poll. This function is called when the _evt15 interrupt is triggered. This interrupt has the lowest priority on this particular processor.

Now that you have seen how BTC has been added to this example, it is time to build the project.

4. On the toolbar, click Rebuild All or choose Rebuild All from the Project menu.

This command builds the project and automatically downloads the application to the target. For details about building projects, refer to “Exercise One: Building and Running a C Program” on page 2-3.
Exercise Two: Using Background Telemetry Channel

Step 3: Set Up the BTC Memory Window and View Data

1. From the View menu, choose Debug Windows and BTC Memory as shown in Figure 3-17.

Figure 3-17. BTC Memory Menu Option
The **BTC Memory** window, shown in Figure 3-18, is displayed.

![ BTC Memory Window ](image)

**Figure 3-18. BTC Memory Window**

The **BTC Memory** window displays BTC data in real time. Data is read from the target when the IDDE issues a read request, and is written when a value is edited in the **BTC Memory** window. You can adjust the rate at which the IDDE requests data by changing the refresh rate.
Exercise Two: Using Background Telemetry Channel

2. Right-click in the BTC Memory window to display a menu of features, shown in Figure 3-19.

<table>
<thead>
<tr>
<th>Go To...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Map</td>
</tr>
<tr>
<td>Lock Columns</td>
</tr>
<tr>
<td>Select Format</td>
</tr>
<tr>
<td>Refresh Rate</td>
</tr>
<tr>
<td>Auto Refresh</td>
</tr>
<tr>
<td>Channel Timeout</td>
</tr>
<tr>
<td>□ Allow Docking</td>
</tr>
<tr>
<td>Close</td>
</tr>
<tr>
<td>Float in Main Window</td>
</tr>
</tbody>
</table>

Figure 3-19. BTC Memory Window Right-Click Menu

Each menu option is described as follows.

**Go To** – Enables you to enter an address or browse for a symbol, and displays memory starting at that address in the BTC Memory window. If the address entered is outside the range of the defined BTC channels, an error message is displayed.

**Show Map** – When this option is enabled, a map of all the defined channels is displayed, as shown in Figure 3-20.
Double-clicking on a channel displays the corresponding memory in the BTC Memory window. When Show Map is disabled, you can choose a channel from a drop-down list selected from the BTC Memory window’s right-click menu (Figure 3-19).

Lock Columns – Locks the number of columns displayed in the BTC Memory window.

- If this option is not enabled, VisualDSP++ displays as many columns as the window’s width can accommodate.
- If this option is enabled, the number of columns does not change, regardless of the window’s width. For example, if four columns are displayed when the option is enabled, four...
columns are displayed, regardless of the window’s width. See Figure 3-21, Figure 3-22, and Figure 3-23 for comparisons.

Figure 3-21 shows the original window.

Figure 3-21. Original Window Width

Figure 3-22 shows the original window expanded with Lock Columns enabled.

Figure 3-23 shows the original window expanded with Lock
**Advanced Tutorial**

**Figure 3-22. Expanded Window With Lock Columns Enabled**

<table>
<thead>
<tr>
<th>Memory Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FF8002A4</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002A8</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002AC</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002B0</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002B4</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002B8</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002BC</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002C0</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002C4</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002C8</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002CC</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002D0</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002D4</td>
<td>00 00 00 00</td>
</tr>
<tr>
<td>FF8002D8</td>
<td>00 00 00 00</td>
</tr>
</tbody>
</table>
Columns disabled.

Figure 3-23. Expanded Window With Lock Columns Disabled

Select Format – Enables you to select how memory is displayed. The choices are 8-bit, 16-bit, and 32-bit hex.

Refresh Rate – Enables you to specify the rate at which the BTC Memory window is refreshed. The IDDE issues a read request based on the rate you select. Select one of the preexisting options (1, 5, 10, or 15 seconds), or use a custom refresh rate. The custom rate is specified in milliseconds.

Auto Refresh – Automatically refreshes the BTC Memory window, based on the refresh rate you select. If this option is disabled, the BTC Memory window is not refreshed until the program is
halted.

**Channel Timeout** – The amount of time that VisualDSP++ will wait for a memory request to the target. After this time, the IDDE stops polling the BTC to prevent a hang.

**Allow Docking** – Docking locks the BTC Memory window to a fixed location (for example, the right side of the workspace). Disabling docking enables you to position the window anywhere in the workspace, including on top of docked windows.

**Close** – Closes the BTC Memory window.

**Float In Main Window** – Disables docking and centers the BTC Memory window in the center of the workspace. You can then move it to any location, but it will not dock. If you move it to a location shared by a docked window, the docked window sits on top.

3. Select the **Timer Interrupt Counter** channel from the drop-down list in the BTC Memory window. Set the **Refresh Rate** to 1 second, and enable **Auto Refresh**.

4. Run the program. Notice how the values in the BTC Memory window are updated each second.

5. Select the **PF10 Counter** channel. This channel counts the number of times that the PF10 button on the ADSP-BF533 EZ-KIT Lite board is pressed. Press this button and watch the PF10 Counter increment in the BTC Memory window.

   **For the PF10 button to work, the SW9.3 switch on the ADSP-BF533 EZ-KIT Lite board must be set to ON.**

   You have now seen some of the basic functionality of BTC.

6. Halt the program and close the **Btc_AsmDemo** project.
Exercise Two: Using Background Telemetry Channel

You are now ready to run the BTC FFT demo.

Running the BTC FFT Demo

The BTC FFT demo demonstrates the transfer of data from the Blackfin EZ-KIT Lite over background telemetry channels. The program generates an input sine wave that increases in frequency over time, and performs a Fast Fourier Transform (FFT) on this input signal. The input and output data are transferred to the IDDE over BTC.

Figure 3-24 provides an overview of the data transfer in the BTC FFT demo.

![Data Transfer Diagram](image)

Figure 3-24. Data Transfer in the BTC FFT Demo

For more information, see the included `readme.txt` file.
Step 1: Build the FFT Demo

1. Start VisualDSP++ and connect to ADSP-BF533 via HPPCI-ICE, ADSP-BF533 via HPUSB-ICE if you connect by way of an emulator, or an ADSP-BF533 EZ-KIT Lite via Debug Agent if you connect by way of an EZ-KIT Lite.

   If you already ran through the first BTC demo, you do not need to create a new session via the session wizard. Simply connect using the session you already created.

2. Open the FFT demo, located in the following folder.

   \Program Files\Analog Devices\VisualDSP 5.0\Blackfin\Examples\ADSP-BF533 EZ-Kit Lite\Background_Telemetry\fftDemo

3. Build the FFT project by clicking Rebuild All ( ) on the toolbar or by choosing Rebuild All from the Project menu.

   This command builds the project and automatically downloads the application to the target. For more details about building projects, refer to “Exercise One: Building and Running a C Program” on page 2-3.
Exercise Two: Using Background Telemetry Channel

Step 2: Plot BTC Data

1. Open the BTC Memory window if it is not already open.

2. From the View menu, choose Debug Windows, Plot, and then Restore, as shown in Figure 3-25.

Figure 3-25. Plot Restore Menu Option

The Restore command opens the Select Plot Settings File dialog box, shown in Figure 3-26.
Select the `fft_in.vps` file and open it. A plot window appears. Follow the same procedure to restore the `fft_out.vps` file.
Exercise Two: Using Background Telemetry Channel

3. Right-click in the FFT In plot window and select Auto Refresh Settings to open the Auto Refresh Settings dialog box, shown in Figure 3-27.

![Auto Refresh Settings Dialog Box](image)

Figure 3-27. Auto Refresh Settings Dialog Box

This dialog box enables you to configure the plotting tool to plot the BTC data in realtime.
4. Complete the dialog box as follows.

In the **Options** group box, select the **Use BTC** option.

The **Use run/halt method** option plots the data, but refreshes the plot window only when the program is halted.

The **Refresh rate** enables you to choose the interval between plot window refreshes. Use the default setting of 150 milliseconds.

The **BTC Modes** group box includes two methods of transferring data to the plot window:

- **Transfer an array of data** (default) – This method uses the `btc_write_array` function. Data is captured at a specific point in the DSP application, copied to a transfer buffer, and held until the host reads the data.

- **Sample a test point over time** – This method uses a data buffer in the DSP program and the `btc_write_value` function. The sampled input data value is copied to the data transfer buffer and read according to the plot refresh rate. The minimum size of the transfer buffer is the product of the plot refresh rate and the data sampling rate (PRR * DSR).

Use the default method, **Transfer an array of data**, for transferring data.

In the **Data Log File** group box, the **Convert to ASCII** button enables you to convert log data to ASCII format. This subject is discussed in more detail in “Step 3: Record and Analyze BTC Data” on page 3-43.

5. Click **OK** to close the **Auto Refresh Settings** dialog box.

6. Enable the **Use BTC** option in the **FFT Out** plot window as you did in step 4.
Exercise Two: Using Background Telemetry Channel

7. Right-click in both plot windows and enable Auto Refresh.

A toolbar appears at the top of each plot window, as shown in Figure 3-28. This toolbar enables you to record BTC data to a file and play back BTC data from a file.

![Figure 3-28. Plot Window With Toolbar](image)

8. In the FFT In plot window, enter a file name, such as Sample.bin, in the text box.

9. Run the program.

Both plot windows should display data being plotted in realtime.
Advanced Tutorial

Step 3: Record and Analyze BTC Data

1. In the FFT In plot window toolbar, click Record ( ). All data in the FFT_Input channel is logged to a file until you stop recording.

2. In the BTC Memory window, select the FREQ STEP SIZE channel. First, right-click and change the format to Hex32. Then change the value in memory from 10 to 100, and notice its effect on the plots. If you would like, try using other values.

3. In the FFT In plot window toolbar, click Stop ( ) to stop logging BTC data.

4. Halt the program.

5. In the FFT In plot window toolbar, click Play ( ). The plot window displays the logged data. The window should appear as if the FFT program is still running.

6. Right-click in the FFT In plot window, open the Auto Refresh Settings dialog box, and click the Convert to ASCII button. The Convert Log File dialog box, shown in Figure 3-29, is displayed.
Exercise Two: Using Background Telemetry Channel

Figure 3-29. Convert Log File Dialog Box
7. Complete the dialog box as follows.

In the Input file text box, click the browse button ( ) to select Sample.bin.

Sample.bin has only one data set, which is selected when you enter the Input file name. If Sample.bin contained more than one data set, you would be able to choose among them in the Data set selection drop-down list.

Next, click the file browse button ( ) next to the Output file text box. The Select Log Output File dialog box that appears should have the file name Sample.dat already in the File name text box. Click Save.

If your window matches Figure 3-30, click OK. The log file is converted from binary to ASCII, which is readable by other programs.

![Completed Convert Log File Dialog Box](image)

Figure 3-30. Completed Convert Log File Dialog Box
Exercise Two: Using Background Telemetry Channel

8. Launch Microsoft Excel. Then open the Sample.dat file and follow the instructions in the Text Import Wizard.

   The .DAT file is a tab-delimited file. Importing the file into Excel or another program, such as MATLAB, enables you to analyze or modify the log file.

You have now completed the BTC FFT demo and the Advanced Tutorial.
INDEX

A
Add Files dialog box, 2-21, 3-25
advanced tutorial
overview, 3-1
PGO steps, 3-2
Auto Refresh command, 3-42
auto refresh rate, setting (for BTC), 3-34
Auto Refresh Settings dialog box, 3-40

B
background telemetry channels
channel definitions, 3-25
commands, 3-25
converting BTC log data to ASCII, 3-41
defined, 3-23
map of, 3-30
map of defined channels, 3-30
Memory window, 3-29
modes of transferring data, 3-41
polling loop command, 3-26
running assembly demo, 3-23
using, 3-23
basic tutorial
features of, 2-2
overview, 2-1
Blackfin processor simulators
cycle-accurate interpreted, 1-7
functional compiled, 1-7
bookmarks, adding to source files, 2-23
breakpoint symbols
red circle, 2-11
yellow arrow, 2-11
BTC, see background telemetry channels
BTC FFT demo
building, 3-37
plotting BTC data, 3-38
running, 3-36
Build Project command, 2-7
Build view, of Output window, 2-7, 2-9

C
channels, BTC, 3-25
channel timeout, setting (for BTC), 3-35
code development tools
features, 1-4
overview, 1-2
commands
BTC, 3-25
BTC polling loop, 3-26
Build Project, 2-7
Execute Data Sets, 3-18
initialize BTC, 3-26
Lock Columns (for BTC), 3-31
Rebuild All, 2-7, 3-27, 3-37
Restore, 3-38
Select Format (for BTC), 3-34
Show Map (for BTC), 3-30
View Sample Count, 2-49
comment characters, moving in source files, 2-23
INDEX

compiled simulators, 1-7
Compile page, 2-19
console view, of Output window, 2-14
converting BTC log data to ASCII, 3-41
Convert Log File dialog box, 3-43
C programs
building and running, 2-3
modifying to call assembly routine, 2-15
custom hardware, 1-6
cycle-accurate simulators, 1-7

D
Data Cursor command, 2-39
data log file (for BTC), 3-41
data sets
attaching to input streams, 3-11
configuring (for PGO), 3-6
configuring with the Copy command, 3-16
plotting, 2-33
debug sessions
creating, 2-10
emulator type, 1-7
EZ-KIT Lite type, 1-6
features, 1-3
simulator type, 1-7
tutorial, 2-9
demos, running
BTC assembly, 3-23
BTC FFT, 3-36
dialog boxes
Add Files, 2-21, 3-25
Auto Refresh Settings, 3-40
Breakpoints, 2-12
Convert Log File, 3-43
Edit Data Set, 3-9
Edit PGO Stream, 3-11
Find, 2-22
Manage Data Sets, 3-8, 3-17
Plot Configuration, 2-33, 2-43
Project Options, 2-18, 2-19
Select Plot Settings, 3-38
Disassembly window
adding and deleting breakpoints, 2-13
information displayed, 2-11
dotprodasm.ldf
modifying, 2-25
viewing, 2-25
dotprodc
automatically loading, 2-11
building, 2-7
running, 2-14
dotprod_main.c
modifying to call a_dot_c_asm, 2-22
opening, 2-8
using to find error, 2-8
dot_product
rebuilding, 2-29
running, 2-30
dot_product_asm, building the project, 2-17

E
Edit Data Set dialog box, 3-9
editor windows, 2-8, 2-23, 3-35
Edit PGO Stream dialog box, 3-11
emulators, 1-7
Enable optimization check box, 2-19
Execute Data Sets command, 2-19
execution units, definition of, 2-49
exercises
building and running C programs, 2-3
linear profiling, 2-45
modifying a C program to call an assembly routine, 2-15
plotting data, 2-31
using background telemetry channels (for BTC), 3-23
using profile-guided optimization (PGO), 3-2

F
Fast Fourier Transform (FFT), 2-42, 2-44
FFT Magnitude command, 2-42, 2-44
files
data log file (for BTC), 3-41
.ldf, 2-1
.PGO, 3-18
Find dialog box, 2-22
finite impulse response (FIR) filter, 2-31
FIR program
global data arrays, 2-32
running, 2-37
viewing the filter results, 2-37
functional compiled simulators, 1-7

G
Generate assembly code annotations check box, 2-19
Generate debug information check box, 2-19

H
histogram, defined, 2-49

I
initialize BTC command, 3-26
input data sets, entering, 2-34

L
linear profiling
collecting and examining data, 2-48
enabling, 2-46
linear and statistical defined, 2-45
results of analyzing the FIR program, 2-48
viewing profile data for the FIR function, 2-50
Linear Profiling window (empty), 2-47
linker description file (LDF), folder, 2-1
Load executable after build command, 2-9
loading
PGO projects, 3-4
projects, 2-3
Lock Columns command (for BTC), 3-31

M
magnifying selected regions, 2-37
Manage Data Sets dialog box, 3-8, 3-17
messages
Output window, 2-11
project is up to date, build completed successfully, 2-7

N
new projects, creating, 2-15

O
% of Histogram data, defined, 2-49
optimizing, programs with PGO, 3-18
output data sets, entering, 2-34
Output window, 2-7
  Build view, 2-7
  Console view, 2-14
  information displayed, 2-11
  viewing a linker error, 2-25

P
PGO, see profile-guided optimization (PGO)
.PGO files, creating, 3-18
Plot Configuration dialog box, 2-33, 2-34, 2-43
plotting
  BTC data, 3-38
  data, 2-31
plot windows
  after running the FIR program, 2-37
  before running the FIR program, 2-35
  FFT In (for BTC), 3-42
  magnified result, 2-38
  magnifying data points, 2-39
  opening, 2-33
  selecting a region to magnify, 2-38
  toolbar (for BTC), 3-42
  viewing data points, 2-39
  viewing signals in the frequency domain,
  2-42, 2-44
  zooming in on a region, 2-37
preferences, specifying, 2-6
Preferences dialog box, 2-6, 2-16
processor
  hardware model (emulator), 1-7
  software model (simulator), 1-7
profile-guided optimization (PGO)
attaching an input stream, 3-11
build output results report, 3-22
configuring a data set, 3-6
configuring data sets, 3-6
configuring data sets with the Copy command, 3-16
data set information results report, 3-20
execution output results report, 3-21
header results report, 3-19
loading PGO projects, 3-4
using, 3-2
Project Options dialog box
displayed, 2-18
Project page, 2-18
projects
  adding files to dot_product_asm, 2-21
  building dotprodc, 2-7
  building dot_product, 2-29
  creating new, 2-15
  dotprodc files, 2-5
  managing, 1-1
  modifying source files, 2-22
  opening, 2-3, 3-4
  opening (PGO), 3-4
  options, 2-18
project source files
  adding, 2-21
project source files, editing, 2-22
Project Wizard, running, 2-16

R
Rebuild All command, 2-7, 3-27, 3-37
recording, BTC data, 3-43
refresh rate, setting for BTC Memory window, 3-34
Reset Zoom command, 2-39
Restore command, 3-38
S

Select Format command (for BTC), 3-34
Select Plot Settings File dialog box, 3-38
Show Map command (for BTC), 3-30
simulators, 1-7
source files
  adding to projects, 2-21
  modifying, 2-22
Statistical Profiling window, 2-45

T

text import wizard, 3-46
timer interrupt counter, setting (for BTC), 3-35
toolbars
  buttons (VisualDSP++), 2-2
plot windows (for BTC), 3-42
transferring data, modes of (for BTC), 3-41

V

views
  Build, 2-7, 2-9
  Console, 2-14

View Sample Count command, 2-49
VisualDSP++
  debug sessions, 1-6
  features, 1-1
  main window, 1-6
  simulators, 1-7
  starting, 2-3
  toolbar buttons, 2-2

W

windows
  BTC Memory, 3-29
  Disassembly, 2-11, 2-14
  editor, 2-8, 2-23, 3-35
  Linear Profiling, 2-46
  Output, 2-8, 2-11
  plot, 2-35, 2-37, 2-38, 2-39
  Project Wizard, 2-16
  Statistical Profiling, 2-45
  wizards
    Project Wizard, 2-16
    Text Import, 3-46