

VISUAL***DSP++***[™] **3.5** **Loader Manual** **for 32-Bit Processors**

Revision 1.0, March 2004

Part Number
82-000035-11

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

Copyright Information

© 2004 Analog Devices, Inc., ALL RIGHTS RESERVED. This document may not be reproduced in any form without prior, express written consent from Analog Devices, Inc.

Printed in the USA.

Disclaimer

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

Trademark and Service Mark Notice

The Analog Devices logo, the Crosscore logo, EZ-KIT Lite, SHARC, the SHARC logo, TigerSHARC, the TigerSHARC logo, VisualDSP, the VisualDSP logo, VisualDSP++, the VisualDSP++ logo, are registered trademarks of Analog Devices, Inc.

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

CONTENTS

PREFACE

Purpose of This Manual	ix
Intended Audience	ix
Manual Contents	x
Technical or Customer Support	x
Supported Processors	xi
Product Information	xi
MyAnalog.com	xi
Embedded Processor and DSP Product Information	xii
Related Documents	xiii
Online Technical Documentation	xiii
From VisualDSP++	xiv
From Windows	xiv
From the Web	xv
Printed Manuals	xv
VisualDSP++ Documentation Set	xv
Hardware Manuals	xvi
Data Sheets	xvi
Notation Conventions	xvi

CONTENTS

INTRODUCTION

Program Development Flow	1-1
Compiling and Assembling	1-2
Linking	1-2
Loading and Splitting	1-2
Boot-loadable Files Versus Non-bootable Files	1-4
Booting Modes	1-5
No-Boot Mode	1-5
PROM Booting Mode	1-6
Host Booting Mode	1-6
Boot Kernels	1-7
Loader Tasks	1-7
Loader Files	1-8
File Searches	1-9

LOADER FOR ADSP-TSXXX TIGERSHARC PROCESSORS

TigerSHARC Loader Guide	2-3
Boot Types	2-5
Determining Boot Mode	2-6
Boot Kernel Files	2-7
Boot Kernel Customization	2-8
TigerSHARC Loader Command Line Reference	2-9

LOADER FOR ADSP-21XXX SHARC PROCESSORS

LOADER FOR ADSP-21XXX SHARC PROCESSORS

Loader Guide	3-2
Booting	3-2
General Power-Up Booting Process	3-4
Boot Mode Selection	3-6
Boot-Loading Process	3-8
Boot Kernel Changes and Software Issues	3-10
Boot Kernels	3-13
Interrupt Vector Table Location	3-14
Boot Types	3-15
EPROM Booting	3-15
Host Booting	3-19
Link Booting	3-23
No Boot Mode	3-24
Multiprocessor Booting	3-24
Multiprocessor EPROM Booting	3-24
Running the Loader	3-26
Load Page	3-26
Loader Command Line	3-26
File Names	3-29
File Extensions	3-29
Loader Command Line Switches	3-30
Processor ID Numbers	3-33

LOADER FOR ADSP-211/2/3xx SHARC PROCESSORS

Loader Guide	4-2
Booting	4-3
Power-Up Booting Process	4-4
Boot Mode Selection	4-5
Boot Kernels	4-6
Boot Kernel Modification and Loader Issues	4-6
Rebuilding a Boot Kernel File	4-7
Rebuilding a Boot Kernel Using Command Lines	4-7
Loader File Issues	4-8
Loader File Header Words	4-9
Interrupt Vector Table Location	4-10
Include Format Files for Booting	4-11
Boot Types	4-12
EPROM Booting	4-12
Host Booting	4-16
Link Port Booting	4-20
SPI Port Booting	4-22
No-Boot Mode	4-23
Multiprocessor Booting	4-24
Multiprocessor EPROM Booting	4-24
Bootting From a Single EPROM	4-25
Sequential EPROM Booting	4-25
Running the Loader	4-26

Load Page	4-26
Loader Command Line	4-26
File Names	4-28
File Extensions	4-29
ADSP-21161, ADSP-2126x, ADSP-2136x Loader Command Line Switches	4-29
Processor ID Numbers	4-29

SPLITTER

Splitter Guide	5-2
Split Tab	5-2
Splitter Command Line Reference	5-4
elfspl21k Command Line Syntax	5-4
File Searches	5-7
Output File Extensions	5-7
Command Line Switches	5-8

FILE FORMATS

Source Files	A-2
C/C++ Source Files	A-2
Assembly Source Files	A-3
Assembly Initialization Data Files	A-3
Header Files	A-4
Linker Description Files	A-4
Linker Command Line Files	A-5
Build Files	A-5

CONTENTS

Assembler Object Files	A-6
Library Files	A-6
Linker Output Files	A-6
Memory Map Files	A-7
Loader Output Files in Intel Hex-32 Format	A-7
Utility Tool hexutil	A-9
Loader Output Files in Include Format	A-10
Loader Output Files in Binary Format	A-10
Splitter Output Files in Motorola S-Record Format	A-11
Splitter Output Files in Intel Hex-32 Format	A-13
Splitter Output Files in Byte Stacked Format	A-13
Debugger Files	A-15
Format References	A-16

INDEX

PREFACE

Thank you for purchasing Analog Devices, Inc. development software for digital signal processor (DSP) applications.

Purpose of This Manual

The *VisualDSP++ 3.5 Loader Manual for 32-Bit Processors* contains information on how to use the loader/splitter to convert executable files into boot-loadable (or non-bootable) files for 32-bit floating point SHARC® and TigerSHARC® processors. These files are then programmed/burned into an external memory device within your target system.

Intended Audience

The primary audience for this manual is a DSP programmer who is familiar with Analog Devices DSPs. This manual assumes that the audience has a working knowledge of the appropriate DSP architecture and instruction set. Programmers who are unfamiliar with Analog Devices DSPs can use this manual but should supplement it with other texts, such as Hardware Reference and Instruction Set Reference manuals, that describe the target architecture.

Manual Contents

The manual contains:

- Chapter 1, “[Introduction](#)”
- Chapter 2, “[Loader for ADSP-TSxxx TigerSHARC Processors](#)”
- Chapter 3, “[Loader for ADSP-2106x/ADSP-21160 SHARC Processors](#)”
- Chapter 4, “[Loader for ADSP-211/2/3xx SHARC Processors](#)”
- Chapter 5, “[Splitter](#)”
- Appendix A, “[File Formats](#)”

Technical or Customer Support

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

- Visit the DSP Development Tools Web site at
www.analog.com/technology/dsp/developmentTools/index.html
- E-mail questions to:
dsptools.support@analog.com
- Phone questions to **1-800-ANALOGD**
- Contact the Analog Devices, Inc. local sales office or authorized distributor

- Send questions by mail to:

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

Supported Processors

VisualDSP++ 3.5® for 32-bit processors currently supports these parts:

- **TigerSHARC processors:** ADSP-TS101, ADSP-TS201, ADSP-TS202, and ADSP-TS203
- **SHARC processors:** ADSP-21020, ADSP-21060, ADSP-21061, ADSP-21062, ADSP-21065L, ADSP-21160, ADSP-21161, ADSP-21261, ADSP-21262, ADSP-21266, ADSP-21267, ADSP-21363, ADSP-21364, and ADSP-21365

Product Information

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

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

MyAnalog.com

MyAnalog.com is a free feature of the Analog Devices web site that allows customization of a web page to display only the latest information on products you are interested in. You can also choose to receive weekly

Product Information

E-mail notification containing updates to the web pages that meet your interests. MyAnalog.com provides access to books, application notes, data sheets, code examples, and more.

Registration:

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

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

Embedded Processor and DSP Product Information

For information on digital signal processors, 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 dsp.support@analog.com
- Fax questions or requests for information to
1-781-461-3010 (North America)
+49 (0) 089 76 903 557 (Europe)
- Access the FTP web site at
[ftp ftp.analog.com](ftp://ftp.analog.com) or [ftp 137.71.23.21](ftp://137.71.23.21)
<ftp://ftp.analog.com>

Related Documents

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

VisualDSP++ 3.5 Getting Started Guide for 32-Bit Processors

VisualDSP++ 3.5 User's Guide for 32-Bit Processors

VisualDSP++ 3.5 Product Release Bulletin for 32-Bit Processors

VisualDSP++ 3.5 C/C++ Compiler and Library Manual for SHARC Processors

VisualDSP++ 3.5 C/C++ Compiler and Library Manual for TigerSHARC Processors

VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors

VisualDSP++ 3.5 Assembler and Preprocessor Manual for SHARC Processors

VisualDSP++ 3.5 Assembler and Preprocessor Manual for TigerSHARC Processors

VisualDSP++ 3.5 Kernel (VDK) User's Guide for 32-Bit Processors

VisualDSP++ 3.5 Component Software Engineering User's Guide for 32-Bit Processors

VisualDSP++ 3.5 Quick Installation Reference Card

For hardware information, refer to the Hardware Reference manual and Data Sheet for the processor.

Online Technical Documentation

Online documentation comprises the 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

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

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

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

From VisualDSP++

- Access VisualDSP++ online Help by selecting Help on the menu.
- Open online Help from context sensitive user interface items (toolbar buttons, menu commands, and windows).

From Windows

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

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

Using Windows Explorer

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

Using the Windows Start Button

Access VisualDSP++ online Help by clicking the **Start** button and choosing

Programs>Analog Devices>VisualDSP++ for 32-bit processors>VisualDSP++ Documentation.

From the Web

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

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

Printed Manuals

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

VisualDSP++ Documentation Set

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

Notation Conventions

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

Hardware Manuals

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

Data Sheets

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

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

Notation Conventions

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



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



Code has been formatted to fit the page width of this manual.

Example	Description
Close command (File menu)	Text in bold style indicates the location of an item within the VisualDSP++ IDE menu system. For example, the Close command appears on the File menu.
{this that}	Alternative required items in syntax descriptions are separated by vertical bars and appear within curly brackets. One or the other is required. Read the example as <i>this</i> or <i>that</i> .
[this that]	Optional items in syntax descriptions are separated by vertical bars and appear within brackets. Read the example as an optional <i>this</i> or <i>that</i> .
[this,...]	Optional item lists in syntax descriptions appear within brackets delimited by commas and terminated with an ellipsis; read the example as an optional comma separated list of <i>this</i> .
.SECTION	Commands, code examples, directives, feature names, keywords, and registers are in text with <code>letter gothic</code> font.
<i>filename</i>	Non-keyword placeholders appear in text with italic style format.
	A note, providing information of special interest or identifying a related topic. In the online version of this book, the word Note appears instead of this symbol.
	A caution, providing information about critical design or programming issues that influence operation of a product. In the online version of this book, the word Caution appears instead of this symbol.

Notation Conventions

1 INTRODUCTION

The majority of this manual describes the loader program (or loader utility) as well as the process of loading and splitting, the final phase of a DSP application program development flow. The process of initializing on-chip and off-chip memories, is often referred to as *booting*.

The majority of this chapter applies to all 32-bit processors. Information applicable to a particular target processor, or to a particular processor family, is provided in the following chapters.

- Chapter 2, “[Loader for ADSP-TSxxx TigerSHARC Processors](#)” on [page 2-1](#)
- Chapter 3, “[Loader for ADSP-2106x/ADSP-21160 SHARC Processors](#)” on [page 3-1](#)
- Chapter 4, “[Loader for ADSP-21161N SHARC Processors](#)” on [page 4-1](#)
- Chapter 5, “[Splitter](#)” on [page 5-1](#)

Program Development Flow

The flow can be split into three phases:

1. “[Compiling and Assembling](#)”
2. “[Linking](#)”
3. “[Loading and Splitting](#)”

Program Development Flow

A brief description of each phase follows.

Compiling and Assembling

Input source files are compiled and assembled to yield object files. Source files are text files containing C/C++ code, compiler directives, possibly a mixture of assembly code and directives, and, typically, preprocessor commands. Refer to the *VisualDSP++ 3.5 Assembler and Preprocessor Manual* or the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual* for information about the assembler and compiler source files.

Linking

Under the direction of the Linker Description File (LDF) and linker settings, the linker consumes separately assembled object and library files to yield an executable file. If specified, shared memory and overlay files are also produced. The linker output conforms to the Executable and Linkable Format (ELF), an industry-standard format for executable files. The linker also produces map files and other embedded information used by the debugger (DWARF-2).

These executable files (.EXE) are not readable by the processor hardware directly. They are neither supposed to be burned onto a EPROM or flash memory device. Executable files are consumed by VisualDSP++ debugging targets, such as the simulator or emulator. Refer to the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors* and online Help for information about linking and debugging.

Loading and Splitting

Upon completing the debug cycle, the processor hardware needs to run on its own, without any debugging tools connected. After power-up, processor memories need to be initialized or to be booted. Therefore, the linker output must be transformed to a format readable by the processor. This

process is handled by the loader/splitter utility. The loader/splitter uses the debugged and tested executable as well as shared memory and overlay files as inputs to yield a processor-loadable file.

VisualDSP++ 3.5 for 32-bit processors includes two loader/splitter programs:

- `elfloader.exe` for ADSP-TS101, ADSP-TS20x, ADSP-2106x, ADSP-2116x, ADSP-2126x and ADSP-2136x processors
- `elfsp121k.exe` for ADSP-TS101, ADSP-TS20x, ADSP-2106x, ADSP-2116x, ADSP-2126x and ADSP-2136x processors

You can run the loader/splitter from the Integrated Development Environment (IDE). In order to do so, change the project type from **DSP Executable** to **DSP Loader File**. The command line interface is also available, if preferred.

Loader operations depend on loader options, which control how the loader processes executable files, letting you select features such as kernels, boot modes, and output file formats. These options are set on the **Load** page of the **Project Options** dialog box in the VisualDSP++ IDE or on the loader command line. Option settings on the **Load** page correspond to switches typed on the command line.

The loader/splitter output is either a boot-loadable or non-bootable file (described in the following “[Boot-loadable Files Versus Non-bootable Files](#)”). The output is meant to be loaded onto the target. There are several ways to use the output:

- Download the loadable file into the processor PROM space on an EZ-KIT Lite™ board via the Flash Programmer plug-in. Refer to VisualDSP++ Help or the *EZ-KIT Lite Evaluation System Manual* for information on the Flash Programmer.

Program Development Flow

- Use VisualDSP++ to simulate booting in a simulator session (where supported). Load the loader file and then reset the processor to debug the booting routines. No hardware is required: just point to the location of the loader file, letting the simulator to do the rest. You can step through the boot kernel code as it brings the rest of the code into memory.
- Store the loader file in an array on a multiprocessor system. A master (host) processor has the array in its memory, allowing a full control to reset and load the file into the memory of a slave processor.

Boot-loadable Files Versus Non-bootable Files

A boot-loadable file is transported into and run from a processor's internal memory. (Note: This is different for ADSP-218x processors.) The file is then programmed (burned) into an external memory device within your target system. The loader outputs files in industry-standard file formats, such as Intel hex-32 and Motorola S, which are readable by most EPROM burners. For advanced usage, other file formats and boot modes are supported.

A non-bootable EPROM image file executes from the processor's external memory, bypassing the built-in boot mechanisms. Preparing a non-bootable EPROM image is called *splitting*. In most cases, developers working with 32-bit processors use the splitter instead of the loader to produce a non-bootable memory image file.

A processor's booting sequence and the application program design dictate the way loader/splitter programs are called to consume and transform executables. For 32-bit processors, splitter and loader features are handled by a single program. The splitter is invoked by a completely different set of command line switches than the loader. Refer to the guide sections of the following chapters for information about splitting.

Booting Modes

A fully debugged program can be automatically downloaded to the processor after power-up or after a software reset. This process is called booting. The way the loader creates a boot-loadable file depends upon how the program is booted into the processor.

Once an executable is fully debugged, it is ready to be converted into a processor loadable file.

The boot mode of the processor is determined by sampling one or more of input flag pins. Booting sequences, highly processor specific, are detailed in the following chapters.

SHARC and TigerSHARC processors support different boot mechanisms. In general, the following schemes can be used to provide program instructions to the processors after reset.

- “No-Boot Mode”
- “PROM Booting Mode”
- “Host Booting Mode”

No-Boot Mode

After reset the processor starts fetching and executing instructions from EPROM/flash memory devices directly. This scheme does not require any loader mechanism. It is up to the user program to initialize volatile memories.

The splitter utility helps to generate a file that can be burned into the PROM memory.

PROM Bootling Mode

After reset, the processor starts reading data from any parallel or serial PROM device. The PROM stores a formatted boot stream rather than raw instruction code. Beside application data, the boot stream contains additional data, such as destination addresses and word counts. A small program called *kernel*, *loader kernel* or *boot kernel* (described [on page 1-7](#)) parses the boot stream and initializes memories accordingly. The loader kernel runs on the target processors. Depending on the architecture, the loader kernel may execute from on-chip boot RAM or may be preloaded from the PROM device into on-chip SRAM and execute from there.

The loader utility generates the boot stream from the linker's executable file and stores it to file format that can be burned into the PROM.

Host Bootling Mode

In this scheme, the target processor is slave to a host system. After reset, the processor delays program execution until it gets signalled by the host system that the boot process has completed. Depending on hardware capabilities, there are two different methods of host booting. In the first case, the host system has full control over all target memories. It halts the target while it is initializing all memories as required. In the second case, the host communicates by a certain handshake with the loader kernel running on the target processor. This kernel may execute from on-chip ROM or may be preloaded by the host devices into the processor's SRAM by any bootstrapping scheme.

The loader/splitter utility generates a file that can be consumed by the host device. It depends on the intelligence of the host device and on the target architecture whether the host expects raw application data or a formatted boot stream.

In this context, a boot-loadable file is a file that stores instruction code in a formatted manner in order to be processed by a boot kernel. A non-bootable file stores raw instruction code.

Boot Kernels

A (loader) boot kernel refers to the resident program in the boot ROM space responsible for booting the processor. Alternatively (or in absence of the boot ROM), the boot kernel can be preloaded from the boot source by a bootstrapping scheme.

When a reset signal is sent to the processor, the processor starts booting from a PROM, host device, or through a communication port. For example, an ADSP-2106x/2116x processor brings a 256 word program into internal memory for execution. This small program is called a *boot kernel*. The boot kernel then brings the rest of the application code into the processor's memory. Finally, the boot kernel overwrites itself with the final block of application code and jumps to the beginning of the application program.

Loader Tasks

Common tasks performed by the loader include:

- Processing loader option settings or command line switches.
- Formatting the output .LDR file according to user specifications. Supported formats are binary, ASCII, hex-32, and more. Valid file formats are described in Appendix A [on page A-1](#).
- Packing the code for a particular data format: 8-, 16- or 32-bit for some processors.
- Adding a boot kernel on top of the user code by default.

Loader Files

- If specified, preprogramming the location of the .LDR file in PROM space.
- Specifying processor IDs for multiple input .DXEs for a multiprocessor system.

Loader Files

The loader output is essentially the same executable code as in the input .DxE file. The loader repackages the executable, as illustrated in [Figure 1-1 on page 1-8](#).

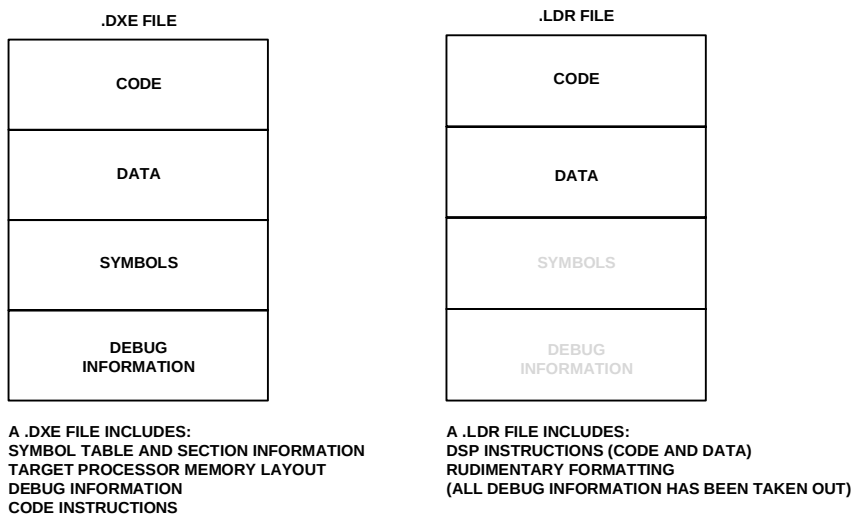


Figure 1-1. A .DxE File Versus a .LDR File

Processor code in a loader file is split into blocks. Each code block is marked with a tag that contains information about the block, such as the number of words or destination in the processor’s memory. Depending on the processor family, there may be additional information in the tag.

Common block types are “zero” (memory is filled with 0s); nonzero (code or data); and final (code or data). Depending on the processor family, there may be other block types.

File Searches

File searches are important in the loader operation. The loader supports relative and absolute directory names and default directories. File searches occur as follows.

- Specified path—If relative or absolute path information is included in a file name, the loader searches only in that location for the file.
- Default directory—If path information is not included in the file name, the loader searches for the file in the current working directory.
- Overlay and shared memory files—The loader recognizes overlay memory files but does not expect these files on the command line. Place the files in the same directory as the executable file that refers to them. The loader can locate them when processing the executable.

When providing an input or output file as a loader/splitter command line parameter, use these guidelines:

- Enclose long file names within straight quotes, “long file name”.
- Append the appropriate file extension to each file.

2 LOADER FOR ADSP-TSXXX TIGERSHARC PROCESSORS

Use the loader utility (`elfloader.exe`) to convert executable files into boot-loadable files for TigerSHARC processors. A boot-loadable file is transported into (and run from) DSP internal memory.

To generate a boot-loadable file, specify options via the **Load** page of the **Project Options** dialog box within the VisualDSP++ IDE. VisualDSP++ invokes the `elfloader` utility and builds the output file.

The loader generates the boot-loadable file by processing executable files. The loader output is a boot-loadable file with an `.LDR` extension. To make a loadable file, the loader processes data from a boot kernel file (`.DXE`) and one or more other executable files (`.DXE`). Once the program is fully debugged, use the loader to generate a set of boot-loadable files for the target system.

This chapter contains the following information.

- [“TigerSHARC Loader Guide” on page 2-3](#)
- [“TigerSHARC Loader Command Line Reference” on page 2-9](#)



Examples in this chapter are for the ADSP-TS201, ADSP-TS202, and ADSP-TS203 processors. For information on these and other processors, refer to the appropriate Hardware Reference.

EE-174: ADSP-TS101S TigerSHARC® Processor Boot Loader Kernels Operation and *EE-200: ADSP-TS20x TigerSHARC Processor Boot Loader Kernels Operation*, provide additional information about booting for different boot scenarios. These EE notes are available from the Analog Devices Web site.

TigerSHARC Loader Guide

The loader (`elfloader.exe`) processes executable (`.DxE`) files, producing a boot-loadable (`.LDR`) file. Loader operations depend on loader options.

Loader options control how the loader processes executable files, letting you select features such as loader kernel, boot type, and output file format. These options appear on the loader command line or the **Load** page of the **Project Options** dialog box in the VisualDSP++ IDE.

 Option settings on the **Load** page correspond to switches in the command line version of the loader.

While working within a VisualDSP++ (simulation, emulation, or EZ-KIT Lite™) session, use the executable (`.DxE`) file, which conforms to the ELF standard.

Upon completing the debug cycle, the project data must be transformed into a format readable by the processor. The loader (`elfloader.exe`) utility handles this process.

You should be familiar with these operations:

- [“Boot Types” on page 2-5](#)
- [“Determining Boot Mode” on page 2-6](#)

Boot Types

Before you select options for loader operation, you should understand how the loader's features apply to the boot type that you are using. This section describes the TigerSHARC boot types and the loader features that support them.

At chip reset, a TigerSHARC processor loads (bootstraps) a 256 instruction program (boot kernel) into the processor's internal memory. This program may be stored on an external PROM, a host processor, or another TigerSHARC processor. The boot type is selected via the processor's $\overline{\text{BMS}}$ pin as described in [“Determining Boot Mode” on page 2-6](#). After the boot kernel loads, it executes itself and then loads the rest of the program and data into the processor. The combination of the boot kernel and the program comprises a boot-loadable file.

TigerSHARC processors support three booting modes: EPROM/flash, host, and link. The boot-loadable files for each of these modes pack the boot data into 32-bit instructions and use a DMA channel of the processor's DMA controller to boot-load the instructions.

EPROM/Flash Booting: For this mode, the loader generates a PROM image that contains all project data and loader code. The project data is then stored in an 8-bit wide external EPROM. After reset, the processor performs a special booting scenario, reading the EPROM content through the processor's external port and initializing on-chip and off-chip memories.

Host Booting: For this mode, the DSP accepts boot data from a 32- or 64-bit synchronous microprocessor (host). The host writes a boot-loadable file to the DSP's AUTODMA register through the DSP's external port, one 32-bit word at a time. Once the last word is written, the DSP takes over and runs the user code.

Link Port Booting: In this mode, the DSP receives boot data via its link port from another TigerSHARC processor.

Determining Boot Mode

To determine the boot mode, a TigerSHARC processor samples its Boot Mode Select ($\overline{\text{BMS}}$) pin. If $\overline{\text{BMS}}$ is sampled low during reset, EPROM/flash boot is selected and, after $\overline{\text{RESET}}$ goes high, $\overline{\text{BMS}}$ becomes output, acting as EPROM chip select. If $\overline{\text{BMS}}$ is sampled high during reset, the TigerSHARC processor will be at an IDLE state, waiting for a host or link boot. The weak (100K ohm) internal pull-down on $\overline{\text{BMS}}$ may not suffice, depending on the line loading. Thus, an external pull-down resistor may be necessary to select the EPROM booting mode. If host or link boot is desired, $\overline{\text{BMS}}$ must be high and may be tied directly to the system power bus (V_{CC}).

Boot Kernel Files

For all boot kernels, the boot-loading process begins by downloading the boot kernel into the processor memory. The boot kernel sets up the processor and loads memory. After initializing the rest of the system, the boot kernel overwrites itself.



You can build an `.LDR` file with or without a kernel. Use the `-NoKernel` command line switch or the **No Kernel** option on the **Load** page to build without a kernel.

VisualDSP++ includes three boot kernel files with each processor type. Boot kernels are loaded at reset into memory segment `seg_ldr`, which is defined in the `ADSP-TSxxx_Loader.ldf` file. The provided files are located in the `...\VisualDSP\TS\ldf` directory.

- `Tsxxx_prom.asm` is the source file for the TigerSHARC PROM boot kernel.
- `Tsxxx_host.asm` is the source file for the TigerSHARC host boot kernel.
- `Tsxxx_link.asm` is the source file for the TigerSHARC link boot kernel.

Boot Kernel Customization

For most systems, some customization of the boot kernel is required. The operation of other tools (notably the C/C++ compiler) is influenced by loader usage.

For more information on boot kernel operations, refer to the comments in the corresponding boot kernel source files and application notes *EE-174: ADSP-TS101S TigerSHARC® Processor Boot Loader Kernels Operation* and *EE-200: ADSP-TS20x TigerSHARC Processor Boot Loader Kernels Operation*.

TigerSHARC Loader Command Line Reference

Invoke the `elfloader.exe` loader utility from the command line to generate a boot-load file, or change the VisualDSP++ project type to **Loader file** on the **Project** page (see [Figure 2-1](#)) of the **Project Options** dialog box and specify the options on the **Load** page (see [Figure 2-2](#)). This section describes the command line only.

When developing a “DSP loader file” project in VisualDSP++, you can modify the default option settings on the **Load** page (see [Figure 2-2 on page 2-11](#)) of the **Projects Options** dialog box. Dialog box buttons and fields correspond to command line switches and parameters. Use the **Additional Options** box to enter options that have no dialog box equivalent. VisualDSP++ invokes the `elfloader` utility to build the output file.

For more information on the user interface, see the *VisualDSP++ 3.5 User's Guide for TigerSHARC DSPs* or online help.

Use the following syntax for the ADSP-TSxxx loader command line.

```
elfloader sourcefile -proc ADSP-TSxxx -switch [-switch ...]
```

where:

- `sourcefile` – Identifies the executable file (.DXX) to be processed for a single-processor boot-loadable file. A file name can include the drive, directory, file name and file extension. Enclose long file names within straight quotes, “long file name”.
- `-proc ADSP-TSxxx` – Specifies the processor for which the boot-loader file is to be built.
- `-switch` – Processes the optional switch. The loader has many switches to select operations and modes.



Command line switches may be placed in the command line in any order.

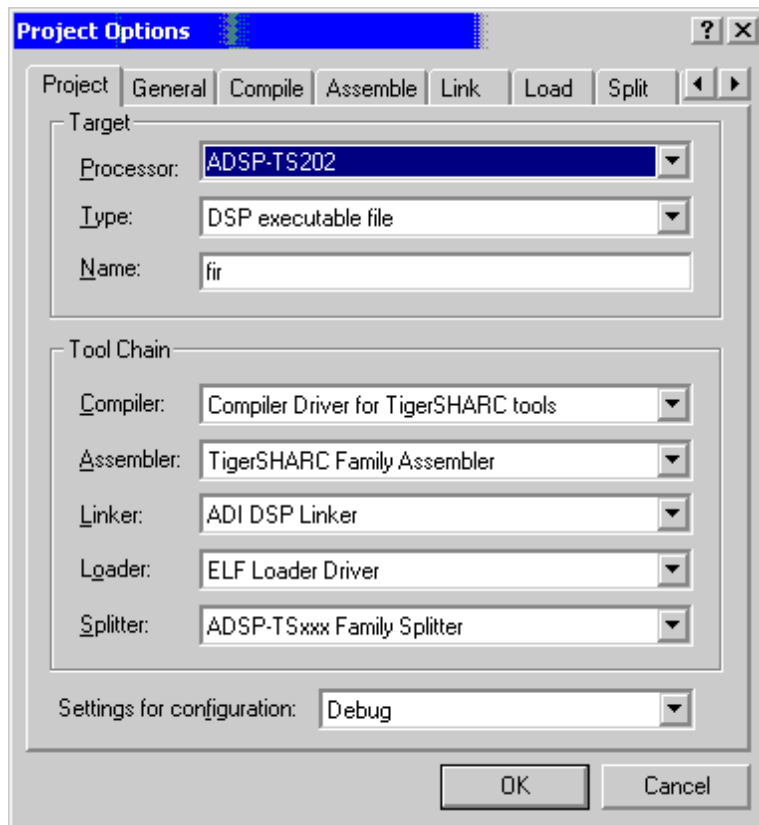


Figure 2-1. Project Page on Project Options

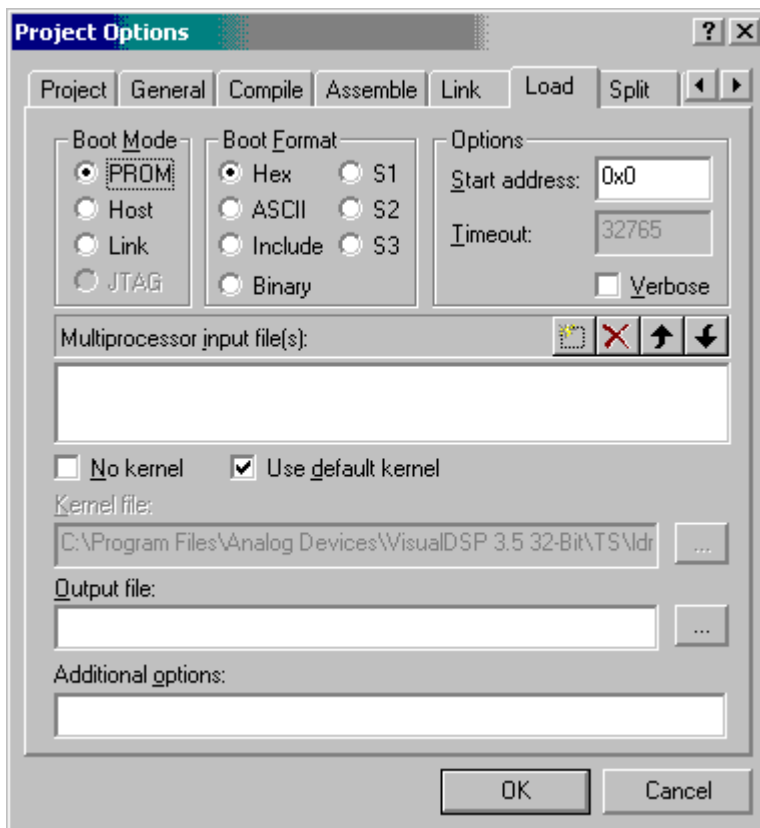


Figure 2-2. Load Page on Project Options

Example

```
elfloader p0.dxe -proc ADSP-TS101 -bprom -fhex -l Ts101_prom.dxe
```

In the above example, the command line runs the loader with:

- `p0.dxe` – Identifies the executable file to process into a boot-loadable file. Note the absence of the `-o` switch causes the output file name to default to `p0.ldr`.
- `-proc ADSP-TS101` – Specifies ADSP-TS101 as the processor type.
- `-bprom` – Specifies EPROM booting as the boot type for the boot-loadable file.
- `-fhex` – Specifies Intel Hex-32 format for the boot-loadable file.
- `-l Ts101_prom.exe` – Specifies the boot kernel file to be used for the boot-loadable file.

File Names

Many loader switches take a file name as an optional parameter.

[Figure 2-2](#) lists the expected file types, names, and extensions.

File searches are important in the loader process. The loader supports relative and absolute directory names, and default directories. File searches occur as follows.

- *Specified path* – If relative or absolute path information is included in a file name, the loader searches only in that location for the file.
- *Default directory* – If path information is not included in the file name, the loader searches for the file in the current working directory.

TigerSHARC Loader Guide

To provide an input or output file name as a command line parameter, use the following guidelines:

- Enclose long file names within straight quotes, “long file name”.
- Append the appropriate file extension to each file.

File Extensions

The loader follows the conventions shown in [Table 2-1](#) for file extensions.

Table 2-1. File Extensions

Extension	File Description
.DxE	Executable files and boot kernel files. The loader recognizes overlay memory (.OVL) and shared memory (.SM) files, but does not expect these files on the command line. Place .OVL or .SM files in the same directory as the .DxE file that refers to them so the loader can find them when processing the .DxE file.
.LDR	Loader output file

Loader Switches

Descriptions for each switch appear in [Table 2-2](#).

Table 2-2. TigerSHARC Loader Command Line Items¹

Switch	Description
sourcefile	A file name without a switch selects an executable file for single-processor boot-loadable files. For multiprocessor boot-loadable files, use the <code>-id#exe=file</code> switch. For information on shared and overlay memory files, see the <i>VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors</i> .
-proc ADSP-TSxxx	Specifies the processor (for example, <code>-proc ADSP-TS101</code> or <code>-proc ADSP-TS201</code>).

Loader for ADSP-TSxxx TigerSHARC Processors

Table 2-2. TigerSHARC Loader Command Line Items¹ (Cont'd)

Switch	Description
-bboot-type	Directs the loader to prepare a boot-loadable file for the specified boot mode. Valid boot types include PROM, HOST, and LINK. If the -b switch does not appear on the command line, the default setting is -bprom. To use a custom kernel, the boot type selected with the -b switch must correspond to the boot kernel selected with the -l switch.
-e filename(s)	Except shared memory. This switch omits the listed shared memory (.SM) file(s) from the output loader file.
-fformat -fhex -fASCII -fbinary -fS1 -fS2 -fS3	Boot-file format. This switch prepares a boot-loadable file in the specified format. Available format selections are: hex (Intel Hex-32), S1, S2, S3 (Motorola S-records), include, ASCII, and binary. Valid formats depend on the -b switch boot type selection. For a PROM boot type, use a hex, S1, S2, S3, include, binary, or ASCII format. For HOST or LINK booting, use ASCII or binary formats. If the -f switch does not appear on the command line, the default boot type format is hex for PROM, and ASCII for HOST or LINK.
-h	Command line help. This switch outputs the list of command line switches to standard output and exits. Combine the -h switch and -proc ADSP-TSxxx to obtain help for TigerSHARC processors. By default, the -h switch alone provides help for the loader driver.
-id#exe=file	Multiprocessor ID. This switch directs the loader to use the processor ID (#) for the corresponding executable file when producing a boot-loadable file for EPROM or host booting a multiprocessor system. This switch is used only to produce a boot-loadable file that boots multiple processors from a single EPROM. Valid # values are 0, 1, 2, 3, 4, 5, 6, and 7. Do not use this switch for single-processor systems. For single-processor systems, use the executable filename as a parameter without a switch.
-NoKernel	Produces an .LDR file that does not include a kernel

Table 2-2. TigerSHARC Loader Command Line Items¹ (Cont'd)

Switch	Description
-l kernelfile	Directs the loader to use the specified <code>kernelfile</code> for the boot-loading routine in the output boot-loadable file. Note the boot kernel file you select with this switch must correspond to the boot type that you select with the <code>-b</code> switch. If the <code>-l</code> switch does not appear on the command line, the loader searches for a default boot kernel file in the installation directory. Based on the <code>-b</code> switch setting, the loader searches for the boot kernel file in the processor-specific <code>\VisualDSP++3.5 32-bit\TS\ldr</code> directory. For <code>-bprom</code>, the <code>TSxxx_prom.dxe</code> file is used. For <code>-bhost</code>, the <code>TSxxx_host.dxe</code> file is used. For <code>-blink</code>, the <code>TSxxx_link.dxe</code> file is used.
-o filename	Directs the loader to use the specified <code>filename</code> for the name of the loader output file. If not specified, the default name is <code>source-file.LDR</code>.
-p address	PROM start address. This switch starts the boot-loadable file at the specified address in the EPROM. If the <code>-p</code> switch does not appear on the command line, the loader starts the EPROM file at address <code>0x0</code> in the EPROM; this EPROM address corresponds to address <code>0x4000000</code> in a TigerSHARC processor.
-t timeout	Timeout cycles. This switch sets the <code>timeout</code> number of cycles as a limit on the number of cycles that the processor spends initializing external memory. Valid values range from 3 to 32765 cycles; 32765 is the default value. The <code>timeout</code> value is directly related to the number of cycles the processor locks the bus for boot-loading, instructing the processor to lock the bus for no more than <code>2X timeout</code> number of cycles. When working with a fast host that cannot tolerate being locked out of the bus, use a relatively small <code>timeout</code> value.
-v	Verbose. This switch outputs status information as the loader processes files.

Table 2-2. TigerSHARC Loader Command Line Items¹ (Cont'd)

Switch	Description
<code>-version</code>	Directs the loader to show its version information. Type “elfloader -version” to display the version of the loader drive. Type “elfloader -proc ADSP-TS101 -version” to display version information of both loader drive and TigerSHARC loader.
<code>-si-version <none x.x></code>	Sets version for the build, with <i>x.x</i> being the revision number for the hardware. The default setting is the latest silicon revision.

¹ Switches are optional. Items in *italics* are user defined.

3 LOADER FOR ADSP-21XXX SHARC PROCESSORS

Use the loader utility (`elfloader.exe`) to convert executable files into boot-loadable files for SHARC DSPs. A boot-loadable file is transported to (and run from) DSP internal memory.

 This chapter describes loader and booting processes for ADSP-21060, ADSP-21061, ADSP-21062, ADSP-21065L, and ADSP-21160 DSPs. Refer to [“Loader for ADSP-21161N SHARC Processors” on page 4-1](#) for information about ADSP-21161N DSPs.

The loader processes executable (`.DXX`) files to generate a boot-loadable file with an `.LDR` extension. To make a loadable file, the loader processes data from a boot kernel file (`.DXX`) and one or more other executable files (`.DXX`). After fully debugging a program, use the loader to generate a set of boot-loadable files for the target system.

This chapter contains the following information.

- [“Loader Guide”](#)
- [“Running the Loader”](#)

Refer to *EE-56: Tips & Tricks on the ADSP-2106x EPROM and HOST bootloader*, *EE-189 Link Port Tips and Tricks for ADSP-2106x and ADSP-2116x* and *EE-77 SHARC Link Port Booting* on the Analog Devices Web site for related information.

Loader Guide

The loader (`elfloader.exe`) processes executable files, producing a boot-loadable file. This file is used to automatically download programs to the processor's internal memory after power-up or after a software reset. This process is called **booting**.

ADSP-2106x/21160 DSPs support three booting modes: EPROM, host, and link. The DSPs have a special hardware feature that boot-loads a small, 256 instruction program called a boot kernel (or boot-loader) into the processor's internal memory at chip reset. When executed, the boot kernel facilitates booting the application code.

This section describes the following topics:

- [“Booting”](#)
- [“Boot Types” on page 3-15](#)
- [“Multiprocessor Booting” on page 3-24](#)

Booting

Before you select options for the loader operation, you should understand how the loader features apply to the boot type that you are using. This section describes boot types for SHARC DSPs and relates loader features that support the appropriate boot types.

ADSP-2106x/21160 DSPs support three boot types (modes): EPROM, host, and link, as shown in [Table 3-3](#) and [Table 3-4 on page 3-6](#). Boot-loadable files for these modes pack boot data into 48-bit instructions and use an appropriate DMA channel of the processor's DMA controller to boot-load the instructions.



ADSP-2106x DSPs use DMA channel 6 (DMAC6) when booting. In ADSP-21160 DSPs, DMAC8 is used for link port booting and DMAC10 is used for the host and EPROM booting.

- When booting from an EPROM through the external port, the SHARC processor reads boot data from an 8-bit external EPROM.
- When booting from a host processor through the external port, the SHARC processor accepts boot data from a 8-, 16- or 32-bit host microprocessor.
- When booting through the link port, SHARC DSPs receive boot data through the link port as 4-bit wide data in link buffer 4.
- For the no boot mode, SHARC DSPs begin executing instructions from external memory.

After the boot process has loaded 256 instructions into the appropriate memory location (starting at 0x20000 for ADSP-21060/61/62 DSPs, 0x8000 for ADSP-21065L DSPs, and 0x40000 for ADSP-21160 DSPs), the processor begins to execute instructions. Because most applications require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the application. This loading routine (Loader Kernel or Boot Kernel) is used to load an entire program.

Software developers who use the loader should be familiar with the following operations:

- [“General Power-Up Booting Process” on page 3-4](#)
- [“Boot Mode Selection” on page 3-6](#)
- [“Boot-Loading Process” on page 3-8](#)
- [“Boot Kernel Changes and Software Issues” on page 3-10](#)

Loader Guide

- [“Boot Kernels” on page 3-13](#)
- [“Interrupt Vector Table Location” on page 3-14](#)

General Power-Up Booting Process

At chip reset and software reset, SHARC DSPs boot-load a small, 256 instruction program (boot kernel) into the processor’s internal memory. When executed, the boot kernel loads the user application code. The combination of the boot kernel and the application code comprise a boot-loadable file.

Each processor model has a start address and reset vector, as shown in [Table 3-1](#). Different DMA channels are used by the various processor models, as shown in [Table 3-2](#).

Table 3-1. DSP Addresses

DSP Model	Start Address	Reset Vector
ADSP-21060	0x20000	0x20004
ADSP-21061	0x20000	0x20004
ADSP-21062	0x20000	0x20004
ADSP-21065L	0x8000	0x8004
ADSP-21160	0x40000	0x40004

Table 3-2. DSP DMA Channels

DSP Model	PROM Booting	Host Booting	Link Booting
ADSP-21060	6	6	6
ADSP-21061	6	6	not supported
ADSP-21062	6	6	6

Table 3-2. DSP DMA Channels (Cont'd)

DSP Model	PROM Booting	Host Booting	Link Booting
ADSP-21065L	8 (DMAC0 programs DMA channel 8)	8 (DMAC0 programs DMA channel 8)	not supported
ADSP-21160	10	10	8

At power-up, the general booting process includes the following steps.

1. A DMA channel is automatically configured to transfer 256 48-bit instructions starting at the start address shown in [Table 3-1](#).
These instructions are called a “boot kernel” or “loader kernel”.
2. The boot kernel then runs and starts up 48-bit word DMAs to load the application executable code and data.
3. The boot kernel overwrites itself with the first 256 words of the application and then begins to execute the user application code from locations 0x20000 to 0x200FF (ADSP-21060/61/62), 0x8000 to 0x80FF (ADSP-21065L), and 0x40000 to 0x400FF (ADSP-21160).



The reset vector address (refer to [Table 3-1](#)) must not contain a valid instruction since it is not executed during the booting sequence. Place a NOP or IDLE instruction at this location.

The boot kernel can come from an external PROM, a host processor, or another SHARC processor. Choosing the boot type directs the system to prepare the appropriate boot kernel.

Boot Mode Selection

The state of various pins select the processor boot mode (boot type). For the ADSP-21060, ADSP-21061, ADSP-21062, and ADSP-21160 DSPs, refer to [Table 3-3](#) and [Table 3-4](#). For ADSP-21065L DSPs, refer to [Table 3-5](#) and [Table 3-6](#).

Table 3-3. Boot Mode Pins - ADSP-21060/061/062, and ADSP-21160

Pin	Type	Description
EB00T	I	EPROM Boot – When EB00T is high, the processor boot-loads from an 8-bit EPROM through the processor's external port. When EB00T is low, the LB00T and $\overline{\text{BMS}}$ pins determine the booting mode.
LB00T	I	Link Boot – When LB00T is high and EB00T is low, the processor boots from another SHARC through the link port. When LB00T is low and EB00T is low, the DSP boots from a host processor through the DSP's external port.
$\overline{\text{BMS}}$	I/O/T ¹	Boot Memory Select – When boot-loading from an EPROM (EB00T=1 and LB00T=0), this pin is an <i>output</i> and serves as the chip select for the EPROM. In a multiprocessor system, $\overline{\text{BMS}}$ is output by the bus master. When host-booting or link-booting (EB00T=0), $\overline{\text{BMS}}$ is an <i>input</i> and must be high.

1 Three-statable in EPROM boot mode (when $\overline{\text{BMS}}$ is an output).

Table 3-4. Boot Mode — ADSP-21060/061/062, and ADSP-21160

EB00T	LB00T	$\overline{\text{BMS}}$	Boot Mode
0	0	0 (Input)	No boot (processor executes from external memory)
0	0	1 (Input)	Host processor
0	1	0 (Input)	Reserved
0	1	1 (Input)	Link port
1	0	Output	EPROM ($\overline{\text{BMS}}$ is chip select)
1	1	x (Input)	Reserved

Table 3-5. Boot Mode Pins — ADSP-21065L

Pin	Type	Description
$\overline{\text{BMS}}$	I/O/T ¹	Boot Memory Select When BSEL is low, $\overline{\text{BMS}}$ is an input pin and selects between host boot mode and no boot mode. In no boot mode, the processor executes from external memory. For no boot mode, connect $\overline{\text{BMS}}$ to ground. For host boot mode, connect $\overline{\text{BMS}}$ to VDD. When BSEL is high, $\overline{\text{BMS}}$ is an output pin and the processor starts up in EPROM boot mode. Connect $\overline{\text{BMS}}$ to the EPROM's chip select.
BSEL	I	EPROM Boot Select Hardwire this signal; it is used for system configuration. When BSEL is high, the processor starts up in EPROM boot mode. The processor assumes the EPROM data bus is 8 bits wide. Connect BSEL to the processor data bus in LSB alignment. When BSEL is low, $\overline{\text{BMS}}$ determines the booting mode. Connect BSEL to ground.

1 Three-statable in EPROM boot mode (when $\overline{\text{BMS}}$ is an output).

Table 3-6. Boot Mode — ADSP-21065L

BSEL	$\overline{\text{BMS}}$	Description
0	1	No boot mode. The processor executes from external memory at location 0x20004.
0	1	Host boot mode. The processor defaults to an 8-bit host bus width.
1	Output	EPROM boot mode. The processor assumes an 8-bit EPROM data bus width. Connect to the data bus in LSB alignment.

Boot-Loading Process

Typically, the first 256 instructions brought into the processor is the loader kernel. The development tools provides default kernels for each boot mode in the VisualDSP\...\ldr directory. Once the kernel has been loaded successfully into the processor, the kernel follows the following sequence:

1. Each kernel begins with general initializations for the DAG registers, appropriate interrupts, processor ID information, and various SDRAM or WAIT state initializations.
2. Once the kernel has finished the task of initializing the processor, the kernel's main job is to initialize processor memory, both internal and external, with user application code.
3. The loader file enables the kernel to load code and data block by block. In the loader file, each block of code or data is preceded by a block header, which describes how long the block is, where it belongs, and what type of data or instruction it is. [Table 3-7](#) includes the block header format. After the block header, the loader outputs the block body, which includes the actual data or instructions that need to be placed.

Table 3-7. Boot Data Block Format

Block Header	First word	Bits 16—47 are not used. Bits 0—15 define the type of data block (tag — see step 4 below.)
	Second word	Bits 16—47 are the start address of the block. Bits 0—15 are the word count for the block.
Block Body (if not a zero block)		Word 1 (48 bits) Word 2 (48 bits) :

4. The loader identifies the data type in the block header with a tag. The 16-bit tag that precedes the block identifies each initialization block. Each type of initialization has a unique tag number (tag number - initialization type) as shown in [Table 3-8](#).

Table 3-8. ADSP-2106x/21160 DSP Loader Block Tags

Tag Number	Block Type	Tag Number	Block Type
0x0000	final init	0x000A	zero pm48
0x0001	zero dm16	0x000B	init pm16
0x0002	zero dm32	0x000C	init pm32
0x0003	zero dm40	0x000E	init pm48
0x0004	init dm16	0x000F	zero dm64 (21160 only)
0x0005	init dm32	0x0010	init dm64 (21160 only)
0x0007	zero pm16	0x0011	zero pm64 (21160 only)
0x0008	zero pm32	0x0012	init pm64 (21160 only)
0x0009	zero pm40		

5. The loader kernel enables the boot port (external or link) to read the block header. Then, based on that information, the kernel places the block body information in the appropriate place in memory using the core.
6. The final section, which is identified by a tag of 0x0, is called the final initialization stage. This section is self modifying code that, when executed, facilitates a DMA over the kernel, replacing it with the user application code that actually belongs in that space at run time. The final initialization code also takes care of interrupts and returns the DSP registers such as SYSCON and DMAC or LCTL to their default values.

7. When the loader detects this tag, it reads the next 48-bit word, indicating the instruction when the loading is completing. This instruction is loaded into the 48-bit `PX` register after the boot-loader finishes initializing memory.
8. The boot kernel requires that the interrupt, External Port, or Link Port address (depending on the boot type) contains an `RTI` instruction. This `RTI` is inserted automatically by the loader utility and is needed to ensure that the kernel executes from the reset vector once the DMA that overwrites the kernel is complete. A last remnant of the kernel code is left at the reset vector location to replace the `RTI` with the user's intended code. Because of this last kernel remnant, user application code should not use the first location of the reset vector. This first location should be a `NOP` or `IDLE` instruction. The kernel will automatically complete, and the Program Controller will begin sequencing the user application code at the second location in the DSP reset vector space.
9. When the boot process is complete, the processor automatically executes the user application code. The only remaining evidence of the boot kernel will be at the first location of the interrupt vector. Almost no memory is sacrificed to the boot code.

Boot Kernel Changes and Software Issues

Some systems require boot kernel customization. The operation of other tools (such as the C/C++ compiler) is influenced by whether the loader is used. This section describes loader usage issues.

When producing a boot-loadable file, the loader reads a DSP executable file and uses information in it to initialize memory correctly. However, the loader cannot determine how the DSP `SYSCON` and `WAIT` registers are to be configured for external memory loading in the system.

If you modify the boot kernel by inserting values for your system, you must rebuild it before generating the boot-loadable file. The boot kernel contains default values for `SYSCON`. The initialization code may be found in the comments for the boot kernel source file.

After modifying a copy of the applicable boot kernel source file (such as `060_link.asm`, `060_host.asm`, or `060_prom.asm`), refer to [“SHARC Loader Command Line Items” on page 3-30](#) for the list of kernel files. Use the loader kernel (-l) command line switch to appropriately initialize the `SYSCON` and `WAIT` registers.



When using VisualDSP++, specify the name of the modified kernel executable in the **Kernel file** box on the **Load** page of the **Project Options** dialog box.

If you modify the boot kernel for EPROM-, host-, or link-booting modes, ensure that the `seg_ldr` memory segment is defined in the `.LDF` file. Refer to the source of this segment in the `.LDF` file located in the `...\21k\ldr\` or `(...\211xx\ldr\)` directory.

The loader generates a warning when vector address (`0x20004` for ADSP-21060/61/62 DSPs, `0x40004` for ADSP-21160 DSPs, or `0x8004` for ADSP-21065L DSPs) does not contain `NOP` or `IDLE`. Because the loader uses this address for the first location of the reset vector during the boot-load process, avoid placing code at this address. When using any of the DSP's power-up booting modes, ensure that this address does not contain a critical instruction, because this address is not executed during the booting sequence. Place a `NOP` or `IDLE` instruction at this location.

When working with the C/C++ compiler and using the loader, do not use the `mem21k` utility, which initializes DSP run time memory for compiled C/C++ code. Instead, use the compiler `-no-mem` switch to disable `mem21k`. For code compiled with the `-no-mem` switch, the `seg_init` segment defined in the `.LDF` file need only contain 16 words (address locations).

Loader Guide

The load kernel project can be rebuilt from the VisualDSP++ IDE. The command line can also be used to rebuild various default boot kernels for ADSP-2106x DSPs.

EPROM Booting

The default boot kernel source file for EPROM booting is `060_prom.asm`. Copy this file to `my_prom.asm` and modify it to suit your system. Then, use the following commands to rebuild the boot kernel.

```
easm21k -21060 my_prom.asm
```

or

```
easm21k -proc ADSP-21060 my_prom.asm  
linker -T 060_ldr.ldr my_prom.doj
```

Host Booting

The default boot kernel source file for host booting is `060_host.asm`. Copy this file to `my_host.asm` and modify it to suit your system. Then, use the following commands to rebuild the boot kernel:

```
easm21k -21060 my_host.asm
```

or

```
easm21k -proc ADSP-21060 my_host.asm  
linker -T 060_ldr.ldr my_host.doj
```

Link Port Booting

The default boot kernel source file for link port booting is `060_link.asm`. Copy this file to `my_link.asm` and modify it to suit your system. Then, use the following commands to rebuild the boot kernel:

```
easm21k -21060 my_link.asm
```

or

```
easm21k -proc ADSP-21060 my_link.asm  
linker -T 060_ldr.ldf my_link.doj
```

Rebuilding Boot Kernels

To rebuild boot kernels for ADSP-21065L DSPs, use these commands.

```
easm21k -21065L my_prom.asm
```

or

```
easm21k -proc ADSP-21065L my_prom.asm  
linker -T 065L_ldr.ldf my_prom.doj
```

To rebuild boot kernels for ADSP-21160 DSPs, use these commands.

```
easm21k -21160 my_prom.asm
```

or

```
easm21k -proc ADSP-21160 my_prom.asm  
linker -T 160_ldr.ldf my_prom.doj
```

Boot Kernels

For all boot kernels, the boot-loading process begins by downloading the boot kernel into the DSP memory. The boot kernel sets up the processor and loads the boot data. After the boot kernel has finished initializing the rest of the system, it performs a final DMA transfer and loads boot data over itself.

Boot kernels are loaded at reset into program segment `seg_ldr`, which is defined in `06x_ldr.ldf` (for ADSP-2106x DSPs) and in `160_ldr.ldf` (for ADSP-21160 DSPs).

Loader Guide

This file is stored in the DSP tools installation directory in (...\\21k\\ldr) for ADSP-2106x DSPs and (...\\211xx\\ldr) for ADSP-21160 DSPs.

For example,

- 060_prom.asm (for ADSP-2106x PROM boot kernels)
- 060_host.asm (for ADSP-2106x host boot kernels)
- 060_link.asm (for ADSP-2106x link boot kernels)



The source file for the ADSP-21065L PROM boot kernel is 065L_prom.asm. For the ADSP-21160 PROM boot kernel, the source file is 160_prom.asm, and so on.

Interrupt Vector Table Location

If a SHARC processor is booted from an external source (EPROM, host, or another SHARC processor), the interrupt vector table is located in internal memory. If, however, the DSP is not booted and executes from external memory, the vector table must be located in the external memory.

The `IIVT` bit of the `SYSCON` control register can be used to override the booting mode in determining where the interrupt vector table is located. If the DSP is not booted (no boot mode), setting `IIVT` to 1 selects an internal vector table, and setting `IIVT` to 0 selects an external vector table. If the DSP is booted from an external source (any mode other than no boot mode), `IIVT` has no effect. The `IIVT` default initialization value is 0.

Boot Types

The following sections describe available boot types.

- “EPROM Booting”
- “Host Booting” on page 3-19
- “Link Booting” on page 3-23
- “No Boot Mode” on page 3-24

For multiprocessor booting, refer to “Multiprocessor Booting” on page 3-24.

EPROM Booting

EPROM booting through the external port is selected when the $\overline{\text{EBOOT}}$ pin is high and the $\overline{\text{LBOOT}}$ pin is low. These settings cause the $\overline{\text{BMS}}$ pin to become an output, serving as chip select for the EPROM. Table 3-9 lists all PROM to DSP connections.

Table 3-9. PROM Connections to ADSP-21xxx DSPs

Processor	Connection
ADSP-21060/61/62	PROM/EPROM connected to DATA23-16 pins
ADSP-21065L	PROM/EPROM connected to DATA0-7 pins
ADSP-21160	PROM/EPROM connected to DATA32-39 pins
ADSP-21xxx	Address pins of PROM connect to lowest address pins of any DSP
ADSP-21xxx	Chip select connect to the $\overline{\text{BMS}}$ pin
ADSP-21060/61/62/65L	Output enable connect to the $\overline{\text{RD}}$ pin
ADSP-21160	Output enable connect to $\overline{\text{RDH}}$ pin

During reset, the ACK line is pulled high internally with a 2K ohm equivalent resistor and is held high with an internal keeper latch. It is not necessary to use an external pull-up resistor on the ACK line during booting or at any other time.

The DMA channel parameter registers are initialized at reset for EPROM booting as shown in Table 3-10 and Table 3-11. The count is initialized to 0x0100 to transfer 256 words to internal memory. The external count register (ECx), which is used when external addresses (BMS space) are generated by the DMA controller, is initialized to 0x0600 (0x100 words at six bytes per word).

Table 3-10. ADSP-2106x DSPs – DMA Settings for EPROM Booting

		Processor	
		ADSP-21060/61/62	ADSP-21065L
BMS space		4M x 8-bit	8M x 8-bit
DMA Channel		DMAC6 = 0x2A1	DMAC0 = 0x2A1
II6	IIEP0	0x20000	0x8000
IM6	IMEP0	0x1 (Implied)	0x1 (Implied)
C6	CEP0	0x100	0x100
EI6	EIEP0	0x80 0000	0x40 0000
EM6	EMEP0	0x1 (Implied)	0x1 (Implied)
EC6	ECEP0	0x600	0x600
IRQ Vector		0x20040	0x8040

Table 3-11. ADSP-21160 DSPs – DMA Settings for EPROM Booting

		ADSP-21160
BMS space		8M x 8-bit
DMA Channel		DMAC10 = 0x4A1

Table 3-11. ADSP-21160 DSPs – DMA Settings for EPROM Booting

	ADSP-21160
II10	0x40000
IM10	0x1 (Implied)
C10	0x100
EI10	0x800000
EM10	0x1 (Implied)
EC10	0x600
IRQ Vector	0x40050

After the DSP `RESET` pin goes inactive on start-up, a SHARC system configured for EPROM booting undergoes the following boot-loading sequence:

1. The DSP `BMS` pin becomes the boot EPROM chip select.
2. The DSP goes into an idle state, identical to that caused by the `IDLE` instruction. The program counter (PC) is set to the DSP reset vector address (refer to [Table 3-1 on page 3-4](#)).
3. The DMA controller reads 8-bit EPROM words, packs them into 48-bit instruction words, and transfers them into internal memory (low-to-high byte packing order) until the 256 words are loaded.
4. The DMA parameter registers for appropriate DMA channels are initialized, as shown in [Table 3-10](#) and [Table 3-11](#). The external port DMA channel (6 or 10) becomes active following reset; it is initialized to set external port DMA enable and selects `DTYPE` for instruction words. The packing mode bits (`PMODE`) are ignored, and 48- to 8-bit packing is forced with least significant word first. The `UBWS` and `UBWM` fields of the `WAIT` register are initialized to generate six wait states for the EPROM access in unbanked external memory space.

5. The DSP begins 8-bit DMA transfers from the EPROM to internal memory using the following external port data bus lines:

D23–D16 (ADSP-21060/61/62 DSPs)

D0–D7 (ADSP-21065L DSPs)

D32–D39 (ADSP-21160 DSPs)

6. Data transfers begin and increment after each access. The external address lines (ADDR 31–0), start at:

0x40 0000 (ADSP-21060/61/62 DSPs)

0x00 0000 (ADSP-21065L DSPs)

0x80 0000 (ADSP-21160 DSPs)

7. The DSP $\overline{RD}$ pin asserts as in a normal memory access, with six wait states (seven cycles).
8. After finishing DMA transfers to load the boot kernel into the processor, the BS0 bit is cleared in the SYSCON register, deactivating $\overline{BMS}$ and activating normal external memory select.

The boot kernel uses three copies of SYSCON – one that contains the original value of SYSCON, a second that contains SYSCON with the BS0 bit set (allowing the DSP to gain access to the boot EPROM), and a third with the BS0 bit cleared.

When BS0=1, the EPROM packing mode bits in the DMACx control register are ignored and 8- to 48-bit packing is forced. (8-bit packing is available only during EPROM booting or when BS0 is set.)

When an external port DMA channel is being used in conjunction with the BS0 bit, none of the other three channels may be used. In this mode, $\overline{BMS}$ is not asserted by a core processor access, but only by a DMA transfer. This allows the boot kernel to perform other external accesses to non-boot memory.

The EPROM is automatically selected by the $\overline{\text{BMS}}$ pin after reset, and other memory select pins are disabled. The DSP's DMA controller reads the 8-bit EPROM words, packs them into 48-bit instruction words, and transfers them to internal memory until 256 words have been loaded. The Master DMA internal and external count registers (CX and ECX) decrement after each EPROM transfer. When both counters reach zero, DMA transfer has stopped and RTI returns the program counter to the address where kernel begins.

To EPROM boot a single-processor system, include the executable on the command line without a switch. Do not use the `-id#exe` switch with `ID=0`.

The WAIT register UBWM (used for EPROM booting) is initialized at reset to both internal wait and external acknowledge required. The internal keeper-latch on the ACK pin initially holds acknowledge high (asserted). If acknowledge is driven low by another device during an EPROM boot, the keeper latch may latch acknowledge low.

The DSP views the deasserted (low) acknowledge as a hold off from the EPROM. In this condition, wait states are continually inserted, preventing completion of the EPROM boot. When writing a custom boot kernel, change the WAIT register early within the boot kernel so UBWM is set to internal wait mode (01).

Host Booting

ADSP-2106x/21160 DSPs accept data from a 8-, 16-, or 32-bit host microprocessor (or other external device) through the external port EPB0 and pack boot data into 48-bit instructions using an appropriate DMA channel. The host is selected when the EBOOT and LBOOT inputs are low and $\overline{\text{BMS}}$ is high. Configured for host booting, the DSP enters the slave mode after reset and waits for the host to download the boot program.

[Table 3-12](#) lists host connections to DSPs.

Table 3-12. Host Connections to ADSP-2106x/21160 DSP

Processor	Connection/Data Bus Pins
ADSP-21060/61/62	Host connected to DATA47–16 or DATA31–16 pins (based on HPM-bits)
ADSP-21065L	Host connected to DATA31–0 or DATA15–0 or DATA7–0 pins (based on HBW-bits)
ADSP-21160	Host connected to DATA63–32 or DATA47–31 pins (based on HPM-bits)
ADSP-21xxx	ADSP-2106x host address to IOP registers and internal memory
ADSP-21060/61/62/65L	ADSP-21065L host address to IOP registers only
ADSP-21160	ADSP-21160 host address to IOP registers and internal memory

After reset, the DSP goes into an idle stage with:

- PC set to address 0x20004 (ADSP-21060/61/62 DSPs)
- PC set to address 0x8004 (ADSP-21065L DSP)
- PC set to address 0x40004 (ADSP-21160 DSP)

The parameter registers for the external port DMA channel (0, 8, or 10) are initialized as shown in [Table 3-10](#) and [Table 3-11](#), except that registers EIX, EMx and ECx are not initialized and no DMA transfers start.

The DMA Channel Control register (DMAC6 for ADSP-21060/61/62), (DMAC0 for ADSP-21065L), or (DMAC10 for ADSP-21160) is initialized, which allows external port DMA enable and selects DTYPE for instruction words, PMODE for 16- to 48-bit word packing (8- to 48-bit for ADSP-21065L), and least significant word first.

Because the host processor is accessing the EPB0 external port buffer, the HPM host packing mode bits of the SYSCON register must correspond to the external bus width specified by the PMODE bits of DMACx control register. For a different packing mode, the host must write to DMACx and SYSCON to

change the `PMODE` and `HBW` (`HPW` for ADSP-21065L) setting. The host boot file created by the loader requires the host processor to perform the following sequence of actions:

1. The host initiates the synchronous booting operation (synchronous not valid for ADSP-21065L) by asserting the DSP $\overline{\text{HBR}}$ input pin , informing the processor that the default 8-/16-bit bus width is used. The host may optionally assert the $\overline{\text{CS}}$ chip select input to allow asynchronous transfers.
2. After the host receives the $\overline{\text{HBG}}$ signal (and `ACK` for synchronous operation or `READY` for asynchronous operation) from the DSP, the host can start downloading instructions by writing directly to the external port DMA buffer 0 or the host can change the reset initialization conditions of the processor by writing to any of the `IOP` control registers. The host must use data bus pins as shown in [Table 3-12](#).
3. The host continues to write 16-bit words (8-bit for ADSP-21065L) to `EPB0` until the entire program is boot-loaded. The host must wait between each host write to external port DMA buffer 0.

After the host boot-loads the first 256 instructions (boot kernel), the initial DMA transfers stop, and the boot kernel:

1. Activates external port DMA channel interrupt (`EP0I`), stores the `DMACx` control setting in `R2` for later restore, clears `DMACx` for new setting, and sets the `BUSLCK` bit in the `MODE2` register to lock out the host.
2. Stores the `SYSCON` register value in `R12` for restore.
3. Enables interrupts and nesting for DMA transfer, sets up the `IMASK` register to allow DMA interrupts, and sets up the `MODE1` register to enable interrupts and allow nesting.

4. Loads the DMA Control register with `0x00A1` and sets up its parameters to read the data word by word from external buffer 0.

Each word is read into the reset vector address (refer to [Table 3-1 on page 3-4](#)) for dispatching. The data through this buffer has structure of boot section which could include more than one initialization block.

5. Clears the `BUSLCK` bit in the `MODE2` register to let the host write in the external buffer 0 right after the appropriate DMA channel is activated.

For information on the data structure of the boot section and initialization, see [“EPROM Booting” on page 3-15](#).

6. Loads the first 256 words of target the executable file during the final initialization stage, and then the kernel overwrites itself.

The final initialization works the same way as with EPROM booting, except that the `BUSLCK` bit in the `MODE2` register is cleared to allow the host to write to the external port buffer.

The default boot kernel for host booting assumes `IMDW` is set to 0 during boot-loading, except during the final initialization stage. When using any power-up booting mode, the reset vector address (refer to [Table 3-1 on page 3-4](#)) must not contain a valid instruction because it is not executed during the booting sequence. Place a `NOP` or `IDLE` instruction at this location.

If the kernel is going to initialize external memory, create a custom boot kernel that sets appropriate values in the `SYSCON` and `WAIT` register. Be aware that the value in the DMA channel register is nonzero, and `IMASK` is set to allow DMA channel register interrupts. Because the DMA interrupt remains enabled in `IMASK`, this interrupt must be cleared before using the DMA channel again. Otherwise, unintended interrupts may occur.

A master SHARC DSP may boot a slave SHARC DSP by writing to its `DMACx` control register and setting the packing mode (`PMODE`) to 00. This allows instructions to be downloaded directly without packing. The wait state setting of 6 on the slave DSP does not affect the speed of the download since wait states affect bus master operation only.

Link Booting

 Link booting is supported on all SHARC DSPs except ADSP-21061 and ADSP-21065L DSPs.

When link-booting SHARC DSPs, the DSP receives data from 4-bit link buffer 4 and packs boot data into 48-bit instructions using the appropriate DMA channels (DMA channel 6 for ADSP-2106x, DMA channel 8 for ADSP-21160).

Link mode is selected when the `EBOOT` is low and `LB00T` and $\overline{\text{BMS}}$ are high. The external device must provide a clock signal to the link port assigned to link buffer 4. The clock can be any frequency, up to a maximum of the DSP clock frequency. The clock falling edges strobe the data into the link port. The most significant 4-bit nibble of the 48-bit instruction must be downloaded first. The link port acknowledge signal generated by the processor can be ignored during booting since the link port cannot be preempted by another DMA channel.

Link booting is similar to host booting – the parameter registers (`IIx` and `Cx`) for DMA channels are initialized to the same values. The DMA Channel 6 Control register (`DMAC6`) is initialized to `0x00A0`, and the DMA Channel 10 Control register (`DMAC10`) is initialized to `0x100000`. This disables external port DMA and selects `DTYPE` for instruction words. The `LCTL` and `LCOM` link port control registers are overridden during link booting to allow link buffer 4 to receive 48-bit data.

After booting completes, the `IMASK` remains set, allowing DMA channel interrupts. This interrupt must be cleared before link buffer 4 is again enabled; otherwise, unintended link interrupts may occur.

Loader Guide

Refer to [“Host Booting” on page 3-19](#) for more information on booting process.

No Boot Mode

No boot mode causes the processor to start fetching and executing instructions at address 0x400004 (ADSP-2106x), 0x20004 (ADSP-21065L) and 0x800004 (ADSP-21160) in external memory space. All DMA control and parameter registers are set to their default initialization values.

Multiprocessor Booting

A multiprocessor system can be booted from a host processor, external EPROM, through a link port, or from external memory.

 Currently, the loader generates single-processor loader files for host and link port booting. As a result, the loader supports multiprocessor EPROM booting only. The application code must be modified to properly set up multiprocessor booting in host and link port booting modes.

To set up DMA processes to boot a single processor, refer to [“Boot Types” on page 3-15](#).

Multiprocessor EPROM Booting

The loader can produce boot-loadable files that permit the SHARC DSPs in a multiprocessor system to boot from a single EPROM. In such a system, the $\overline{\text{BMS}}$ signals from each SHARC DSP are OR'ed together to drive the chip select pin of the EPROM. Each SHARC DSP boots in turn, according to its priority. When the last DSP finishes booting, it must inform the DSPs to begin program execution.

Besides taking turns when booting, EPROM booting of multiple DSPs is similar to single-processor EPROM booting.

When booting a multiprocessor system through a single EPROM:

- Connect all $\overline{\text{BMS}}$ pins to EPROM.
- Processor ID#1 boots first. The other processors follow.
- The EPROM boot kernel accepts multiple .EXE and .DXE files and reads the ID field in SYSTAT to determine which area of EPROM to read.
- All processors require a software flag or hardware signal (FLAG pins) to indicate that booting is complete.

When booting a multiprocessor system through an external port:

- The host can use the host interface.
- A SHARC DSP that is EPROM-, host-, or link-booted can boot the other processors through the external port (host boot mode).

For multiprocessor EPROM booting, select the **Multiprocessor** check box on the **Load** page of the **Project Options** dialog box or specify the `-id#exe` switch on the loader command line. These options specify the executable file targeted for a specific DSP.

Do not use the `-id#exe` switch to EPROM-boot a single processor whose ID is 0. Instead, name the executable on the command line without a switch. For a single processor with ID=1, use the `-id1exe` switch.

Multiple .DXE files can be specified in EPROM booting mode only. Multiple files may not be specified for host port booting or link port booting.

Running Loader

This section provides reference information on loader options. Specify options with the loader command line switches (command invocation) or through the dialog box settings in the VisualDSP++ IDE.

Load Page

When developing a “DSP loader file” project from VisualDSP++, modify the default options from the **Load** page (also called Load property page) of the **Projects Options** dialog box. [Figure 3-1 on page 3-27](#) shows a representative screen shot of the Load page for a SHARC processor. For information relative to a specific processor, refer to the VisualDSP++ online help for that processor.

VisualDSP++ invokes the `elfloader` utility to build the output file. The **Load** page buttons and fields correspond to loader command line switches and parameters (see [Table 3-14 on page 3-30](#)). Use the **Additional Options** box to enter options that do not have dialog box equivalents.

For ADSP-21020 DSPs, the only permitted boot mode is JTAG:
-bJTAG is automatically entered in the **Additional Options** box.

Loader Command Line

This section provides reference information about the loader command line. Switches and their descriptions appear in [Table 3-14 on page 3-30](#).

Invoke the `elfloader.exe` loader utility from the command line to generate a boot-loader file. The loader uses the following command line syntax:

```
elfloader sourcefile -proc processor -switch [-switch ...]
```

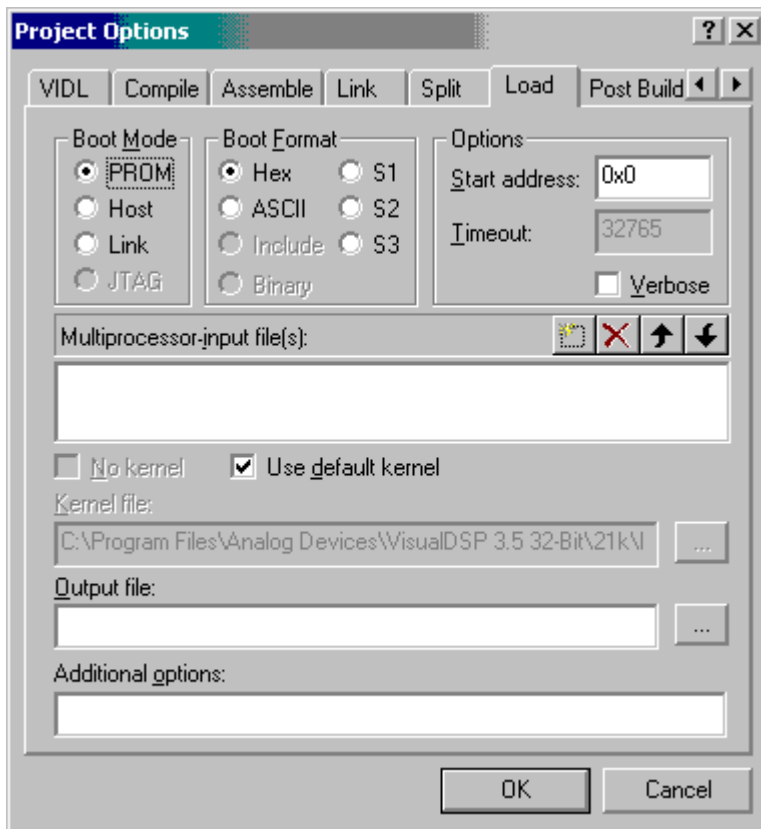


Figure 3-1. Load Page on Project Options

Running the Loader

where:

- *sourcefile* – Identifies the executable file (.DXE) to be processed for a single-processor boot-loadable file. Include the drive, directory, file name and file extension. Enclose long file names within straight quotes, “long file name”.
- *processor* – Specifies the processor (for example, ADSP-21065L) for which the boot-loader file is to be built. If the processor is not specified, the default is ADSP-21062.
- *-switch* – Processes the optional switch. The loader has many switches to select operations and modes.



Command line switches may be placed in the command line in any order.

Example

```
elfloader p0.dxe -bprom -fhex -l 060_prom.dxe -proc ADSP-21060
```

In the above example, the command line runs the loader with:

- *p0.dxe* – Identifies the executable file to process into a boot-loadable file. Note the absence of the *-o* switch causes the output file name to default to *p0.ldr*.
- *-bprom* – Specifies EPROM booting as the boot type for the boot-loadable file.
- *-fhex* – Specifies Intel hex-32 format for the boot-loadable file.
- *-l 060_prom.exe* – Specifies *060_prom.exe* as the boot kernel file to be used in the boot-loadable file.
- *-proc ADSP-21060* – Specifies the DSP type as ADSP-21060.

File Names

Many loader switches take a file name as an optional parameter.

[Figure 3-14](#) lists the expected file types, names, and extensions.

File searches are important in the loader process. The loader supports relative and absolute directory names, and default directories. File searches occur as follows.

- *Specified path* – If relative or absolute path information is included in a file name, the loader searches only in that location for the file.
- *Default directory* – If path information is not included in the file name, the loader searches for the file in the current working directory.

When providing an input or output file as a command line parameter, use the following guidelines:

- Enclose long file names within straight quotes, “long file name”.
- Append the appropriate file extension to each file.

File Extensions

The loader follows the conventions shown in [Table 3-13](#) for file extensions.

Table 3-13. File Extensions

Extension	File Description
.DXE	Executable files and boot kernel files. The loader recognizes overlay memory (.OVL) and shared memory (.SM) files, but does not expect these files on the command line. Place .OVL and/or .SM files in the same directory as the .DXE file that refers to them so the loader can find them when processing the .LDR file.
.LDR	Loader output file

Running the Loader

Loader Command Line Switches

Descriptions for each loader switch and file name appear in [Table 3-14](#).

Table 3-14. SHARC Loader Command Line Items ¹

Switch	Description
<i>sourcefile</i>	A file name without a switch specifies an executable file for a single-processor boot-loadable file. For multiprocessor boot-loadable files, use the <code>-id#exe=file</code> switch.
<code>-proc processor</code>	Specifies the processor. <i>processor</i> is ADSP-21060, ADSP-21061, ADSP-21062, ADSP-21065L, ADSP-21160, ADSP-21261, ADSP-21262, ADSP-21266, ADSP-21267, ADSP-21363, ADSP-21364, or ADSP-21365.
<code>-bprom</code> <code>-bhost</code> <code>-blink</code> <code>-bJTAG</code>	Specifies the boot mode. The <code>-b</code> switch directs the loader to prepare a boot-loadable file for the specified boot mode. Valid boot modes (boot types) include PROM, host, and link. For ADSP-21020 DSPs, JTAG is the only permitted boot mode. If <code>-b</code> does not appear on the command line, the default is <code>-bprom</code> . To use a custom kernel, the boot type specified with the <code>-b</code> switch must correspond to the boot kernel selected with the <code>-l</code> switch.
<code>-caddress</code>	Custom option. This switch directs the loader to use the specified address. Valid addresses are: 20004 and 20040 (ADSP-2106x DSPs) 8004 and 8040 (ADSP-21065L DSPs) 40000 and 40050 (ADSP-21160 DSPs) The loader obtains the proper address even when this switch is absent from the command line.
<code>-e filename</code>	Except shared memory. This switch omits specified shared memory (.SM) file from the output loader file. Use this option to omit the shared parts of the executable from multiprocessor boot files. To omit multiple .SM file, use this switch/parameter multiple times in the command line. For example, to omit two files, use: <code>-e fileA.SM -e fileB.SM</code> .

Table 3-14. SHARC Loader Command Line Items ¹ (Cont'd)

Switch	Description
-fhex -fASCII -fbinary -finclude -fS1 -fS2 -fS3	Specifies the boot file format. The -f switch prepares a boot-loadable file in the specified format. Valid selections (Intel hex 32, ASCII, S1, S2, S3, binary, or include) depend on the target processor and boot type selection (-b switch). For a PROM boot type, select hex ASCII, S1, S2, S3, or include. For host or link booting, select ASCII, binary, or include format. If the -f switch does not appear on the command line, the default boot type format is hex for PROM, and ASCII for HOST or LINK.
-h or -help	Command line help. This switch outputs the list of command line switches to standard output and exits. Type -proc ADSP-21xxx -h, where xxx is 060, 061, 062, 065L, or 160 to obtain help for SHARC processors. By default, the -h switch alone provides help for the loader driver.
-id#exe=filename	Specifies the multiprocessor ID. The -id switch directs the loader to use the processor ID (#) for the corresponding executable file when producing a boot-loadable file for EPROM booting a multiprocessor system. This switch is used only to produce a boot-loadable file that boots multiple DSPs from a single EPROM. Valid values for # are 1, 2, 3, 4, 5, and 6. Do not use this switch for single-processor systems. For single-processor systems, use the executable filename as a parameter without a switch. For more information, refer to “Processor ID Numbers” on page 3-33 .
-version	Directs the loader to show its version information. Type “elfloader -version” to display the version of the loader drive. Type “elfloader -proc ADSP-21060 -version” to display version information of both loader drive and SHARC loader.
-si-version <none x.x>	Sets version for the build, with x.x being the revision number for the hardware. The default setting is the latest silicon revision.
-id#exe=N	Points the ID(N) loader jump table entry to the ID(#) image. If the executable for the ID(#) processor is identical to the executable of the ID(N) processor, this switch can be used to set the PROM start address of ID(#) to be the same as for ID(N). This effectively reduces the size of the loader file by providing a single copy of an executable to two or more processors in a multiprocessor system.

Running the Loader

Table 3-14. SHARC Loader Command Line Items ¹ (Cont'd)

Switch	Description
<code>-l kernelfile</code>	Directs the loader to use the specified <i>kernelfile</i> for the boot-loading routine in the output boot-loadable file. The boot kernel file selected with this switch must correspond to the boot type selected with the <code>-b</code> switch. If the <code>-l</code> switch does not appear on the command line, the loader searches for a default boot kernel file in the installation directory. Based on the boot type (<code>-b</code> switch parameter), the loader searches in the processor specific loader directory (...\<i>21k\ldr</i>) or (...\<i>211x\ldr</i>) for the boot kernel file. For <code>-bprom</code>, the default boot kernel file is: 060_prom.dxe (ADSP-21060, ADSP-21061, and ADSP-21062 DSPs) 065L_prom.dxe (ADSP-21065L DSPs) 160_prom.dxe (ADSP-21160 DSPs) For <code>-bhost</code>, the default boot kernel file is: 060_host.dxe (ADSP-21060, ADSP-21061, and ADSP-21062 DSPs) 065L_host.dxe (ADSP-21065L DSPs) 160_host.dxe (ADSP-21160 DSPs) For <code>-blink</code>, the default boot kernel file is: 060_link.dxe (ADSP-21060 and ADSP-21062 DSPs) 160_prom.dxe (ADSP-21160 DSPs)
<code>-o filename</code>	Directs the loader to use the specified <i>filename</i> as the name for the loader output file. If not specified, the default name is <i>sourcefile.LDR</i>.
<code>-paddress</code>	PROM start address. Places the boot-loadable file at the specified address in the EPROM. If the <code>-p</code> switch does not appear on the command line, the loader starts the EPROM file at address 0x0; this EPROM address corresponds to 0x400000 in ADSP-2106x DSPs.

Table 3-14. SHARC Loader Command Line Items ¹ (Cont'd)

Switch	Description
-t <i>timeout</i>	(Host boot only) Specifies timeout cycles. This limits the number of cycles that the DSP spends initializing external memory with zeros. Valid values range from 3 to 32765 cycles; 32765 is the default. <i>timeout</i> is directly related to the number of cycles the DSP locks the bus for boot-loading, instructing the DSP to lock the bus for no more than two times the <i>timeout</i> number of cycles. When working with a fast host that cannot tolerate being locked out of the bus, use a relatively small <i>timeout</i> value.
-v	Verbose loader messages. This switch outputs status information as the loader processes files.

¹ Switches are optional. Items in *italics* are user defined.

Processor ID Numbers

When used with a single-processor system, there is only one input (.DXE) file without any prefix and suffix to the input file name, for example:

```
elfloader -proc ADSP-21060 -bprom Input.dxe <switches>
```

A multiprocessor system requires a distinct processor ID number for each input file in the command line. A processor ID is provided via the -id#exe= switch ,where # is an ID number (1 to 6).

In the following example, the loader processes the input file Input1.dxe for the processor with an ID of 1, and it processes the input file Input2.dxe for the processor with an ID of 2.

```
elfloader -proc ADSP-21060 -bprom -id1exe=Input1.dxe  
-id2exe=Input2.dxe <switch>
```

Running the Loader

If the executable for the ID(#) processor is identical to the executable of the ID(N) processor, the output loader file will contain only one copy of the code from the input file.

```
elfloader -proc ADSP-21060 -bprom -idl exe=Input.dxe  
          -id2exe=1 <switch>
```

The loader points the ID(2) loader jump table entry to the ID(#) image. This effectively reduces the size of the loader file.

4 LOADER FOR ADSP-211/2/3xx SHARC PROCESSORS

Use the loader utility (`elfloader.exe`) to convert executable files into boot-loadable files for SHARC DSPs. A boot-loadable file is transported to (and run from) DSP memory.

 This chapter describes loader and booting processes for ADSP-21161N, ADSP-2126x, and ADSP-2136x DSPs only. Refer to [“Loader for ADSP-2106x/ADSP-21160 SHARC Processors” on page 3-1](#) for information about ADSP-21060, ADSP-21061, ADSP-21062, ADSP-21065L, and ADSP-21160 DSPs.

The loader processes executable (`.DXX`) files to generate a boot-loadable file with an `.LDR` extension. To make a loadable file, the loader processes data from a boot kernel file (`.DXX`) and one or more other executable files (`.DXX`). After fully debugging a program, use the loader to generate a set of boot-loadable files for the target system.

This chapter contains the following information.

- [“Loader Guide”](#)
- [“Running the Loader”](#)

Refer to *EE-177 SHARC SPI Booting*, *EE-199 Link Port Booting on the ADSP-21161 SHARC DSP*, *EE-209 Asynchronous Host Interface on the ADSP-21161 SHARC DSP* on the Analog Devices DSP Web site for related information.

Loader Guide

The loader (`elfloader.exe`) processes executable files, producing a boot-loadable file. This file is used to automatically download programs to the processor memory after power-up or a software reset. This process is called **booting**.

ADSP-21161, ADSP-2126x, ADSP-2136x DSPs support four booting modes: EPROM, host, link, and SPI. ADSP-21161, ADSP-2126x, ADSP-2136x DSPs have a special hardware feature that boot-loads a small, 256 instruction program (called a boot kernel or loader kernel) into the processor's internal memory at chip reset. When executed, the boot kernel facilitates the booting of application code.

Before you select options for the loader operation, you should understand how the loader features apply to the boot type that you are using. This section describes the boot types for ADSP-21161, ADSP-2126x, ADSP-2136x processors and the loader features that support them.

This section described the following topics:

- [“Booting”](#)
- [“Boot Types” on page 4-12](#)
- [“Multiprocessor Booting” on page 4-24](#)

Bootling

ADSP-21161N DSPs support four bootling modes: EPROM, host processor, link port, and SPI, as shown in [Table 4-4 on page 4-10](#) and [Table 4-5 on page 4-15](#). Boot-loadable files for these modes pack boot data into words in appropriate widths and use an appropriate DMA channel of the DSP's DMA controller to boot-load the words.

- When bootling from an EPROM through the external port, the ADSP-21161, ADSP-2126x, ADSP-2136x DSP reads boot data from an 8-bit external EPROM.
- When bootling from a host processor through the external port, the ADSP-21161, ADSP-2126x, ADSP-2136x DSP accepts boot data from 8-, 16- or 32-bit host microprocessor.
- When bootling through the link port, ADSP-21161, ADSP-2126x, ADSP-2136x DSPs receive boot data through the link port as 4-bit wide data in link buffer 4.
- When bootling through the SPI port, the ADSP-21161, ADSP-2126x, ADSP-2136x DSP uses DMA channel 8 of the I/O processor to transfer instructions to internal memory. In this boot mode, the processor receives data in the `SPIRX` register.
- For no-boot mode, the ADSP-21161, ADSP-2126x, ADSP-2136x DSPs begin executing instructions from external memory.

After the boot process has loaded 256 words (called a boot kernel or loader kernel) into memory, the processor begins to execute instructions. Because most applications require more than 256 words of instructions and initialization data, the boot kernel typically serves as a loading routine for the application to load an entire program.

Loader Guide

Software developers who use the loader should be familiar with the following operations:

- [“Power-Up Booting Process” on page 4-4](#)
- [“Boot Mode Selection” on page 4-5](#)
- [“Boot Kernels” on page 4-6](#)
- [“Boot Kernel Modification and Loader Issues” on page 4-6](#)
- [“Loader File Header Words” on page 4-9](#)
- [“Interrupt Vector Table Location” on page 4-10](#)

Power-Up Booting Process

ADSP-21161, ADSP-2126x, ADSP-2136x DSPs have a hardware feature that boot-loads a small, 256 instruction program (called a boot kernel or a loader kernel) into the processor’s internal memory at chip reset. When executed, the boot kernel facilitates booting user application code. The combination of the boot kernel and the application code comprise the boot-loadable file.

At power-up, the general booting process includes the following steps.

1. A DMA channel is automatically configured for a 256 instruction (48-bit) transfer.

The first 256 instructions are called the boot kernel.

2. The boot kernel then runs and loads the application executable code and data.
3. The boot kernel overwrites itself with the first 256 words of the application.

The boot kernel can come from an external PROM, a host processor, or another SHARC DSP. The boot type selection directs the system to prepare the appropriate boot kernel.

Boot Mode Selection

The boot mode is selected using the $\overline{\text{LBOOT}}$, $\overline{\text{EBOOT}}$, and $\overline{\text{BMS}}$ pins. [Table 4-1](#) and [Table 4-2](#) show how the booting mode is selected for ADSP-21161N DSPs.

Table 4-1. Boot Mode Pins (Descriptions)

Pin	Type	Description
$\overline{\text{EBOOT}}$	I	EPROM Boot – When $\overline{\text{EBOOT}}$ is high, the DSP boot-loads from an 8-bit EPROM through the DSP's external port. When $\overline{\text{EBOOT}}$ is low, the $\overline{\text{LBOOT}}$ and $\overline{\text{BMS}}$ pins determine booting mode.
$\overline{\text{LBOOT}}$	I	Link Boot – When $\overline{\text{LBOOT}}$ is high (and $\overline{\text{EBOOT}}$ is low), the DSP boots from another SHARC DSP through the DSP's link port. When $\overline{\text{LBOOT}}$ is low (and $\overline{\text{EBOOT}}$ is low), the DSP boots from a host processor through the DSP's external port.
$\overline{\text{BMS}}$	I/O/T ¹	Boot Memory Select – When boot-loading from EPROM ($\overline{\text{EBOOT}}=1$ and $\overline{\text{LBOOT}}=0$), this pin is an <i>output</i> and serves as the chip select for the EPROM. In a multiprocessor system, $\overline{\text{BMS}}$ is output by the bus master. When host-booting, link-booting, or SPI-booting, ($\overline{\text{EBOOT}}=0$), $\overline{\text{BMS}}$ is an <i>input</i> and must be high.

1 Three-statable in EPROM boot mode (when $\overline{\text{BMS}}$ is an output).

Table 4-2. Boot Mode Pins (States)

$\overline{\text{EBOOT}}$	$\overline{\text{LBOOT}}$	$\overline{\text{BMS}}$	Booting Mode
1	0	Output	EPROM (connect $\overline{\text{BMS}}$ to EPROM chip select)
0	0	1 (Input)	Host processor
0	1	1 (Input)	Link port
0	1	0 (Input)	Serial port (SPI)
0	0	0 (Input)	No-boot (processor executes from external memory)

Boot Kernels

Boot loading begins by downloading the boot kernel into the DSP memory. The boot kernel sets up the processor and loads boot data. After the boot kernel finishes initializing the rest of the system, the boot kernel loads boot data over itself with a final DMA transfer.

Four boot kernels ship with VisualDSP++; refer to [Table 4-3](#).

Table 4-3. Default Boot Kernel Files

PROM	Link	Host	SPI
161_prom.asm	161_link.asm	161_host.asm	161_spi.asm

At DSP reset, a boot kernel is loaded into the `seg_ldr` memory segment as defined in `161_ldr.ldf`, which is stored in the DSP tools installation directory (...\\211xx\\ldr).

Boot Kernel Modification and Loader Issues

The loader reads a DSP executable file (.DXE) to produce a boot-loadable file (.LDR). However, the loader cannot determine how the processor SYSCON, WAIT, and SDRAM registers are to be configured for external memory loading.

Boot kernel customization is required for some systems. The operation of other tools (such as the C/C++ compiler) is influenced by whether the loader is used.

If you do not specify a boot kernel file via the **Load** page of the **Project Options** dialog box in VisualDSP++ (or via the `-l` command line switch), the loader places a default boot kernel in the loader output file based on the specified boot type.

Rebuilding a Boot Kernel File

If you modify the boot kernel source file (.ASM) by inserting correct values for your system, you must rebuild the boot kernel before generating the boot-loadable file. The boot kernel source file contains default values for SYSCON. The WAIT, SDCTL, and SDRDIV initialization code are in boot kernel file comments.

To Modify a Boot Kernel Source File

1. Copy the applicable boot kernel source file (161_link.asm, 161_host.asm, 161_prom.asm, or 161_spi.asm).
2. Apply the appropriate initializations of the SYSCON and WAIT registers.

After modifying the boot kernel source file, rebuild the boot kernel (.DXE) file. Do this from within the VisualDSP++ IDE (refer to VisualDSP++ online Help for details), or rebuild a boot kernel file from the command line..

Rebuilding a Boot Kernel Using Command Lines

Rebuild a boot kernel using command lines as follows.

EPROM Booting. The default boot kernel source file for EPROM booting is 161_prom.asm. After copying the default file to my_prom.asm and modifying it to suit your system, use the following command lines to rebuild the boot kernel.

```
easm21k -proc ADSP-21161 my_prom.asm  
linker -T 161_ldr.ldf my_prom.doj
```

Loader Guide

Host Booting. The default boot kernel source file for host booting is `161_host.asm`. After copying the default file to `my_host.asm` and modifying it to suit your system, use the following command lines to rebuild the boot kernel:

```
easm21k -proc ADSP-21161 my_host.asm  
linker -T 161_ldr.ldf my_host.doj
```

Link Booting. The default boot kernel source file for link booting is `161_link.asm`. After copying the default file to `my_link.asm` and modifying it to suit your system, use the following command lines to rebuild the boot kernel:

```
easm21k -proc ADSP-21161 my_link.asm  
linker -T 161_ldr.ldf my_link.doj
```

SPI Booting. The default boot kernel source file for link booting is `161_SPI.asm`. After copying the default file to `my_SPI.asm` and modifying it to suit your system, use the following command lines to rebuild the boot kernel:

```
easm21k -proc ADSP-21161 my_SPI.asm  
linker -T 161_ldr.ldf my_SPI.doj
```

Loader File Issues

If you modify the boot kernel for the EPROM, host, SPI, or link booting modes, ensure that the `seg_ldr` memory segment is defined in the `.LDF` file. Refer to the source of this memory segment in the `.LDF` file located in the `...\211xx\ldr\` directory.

Because the loader uses this address for the first location of the reset vector during the boot-load process, avoid placing code at this address. When using any of the processor's power-up booting modes, ensure that this address does not contain a critical instruction, because this address is not

executed during the booting sequence. Place a `NOP` or `IDLE` in this location. The loader generates a warning if the vector address `0x40004` does not contain `NOP` or `IDLE`.



When using VisualDSP++ to create the loader file, specify the name of the customized boot kernel executable in the **Kernel file** box on the **Load** page of the **Project Options** dialog box.

Loader File Header Words

The loader inserts header words at the beginning of data blocks in the loader file. The boot kernel uses header words to properly place data and instruction blocks into DSP memory. The header format for PROM, host, and link boot-loader files is as follows.

```
0x00000000DDDD
0xAAAAAAAALLLL
```

In the above example, `D` = data block type tag, `A` = block start address, and `L` = block word length.

Data block headers for in a loader boot file for the SPI boot type consists of three 32-bit words.

For single-processor systems, the data header has three 32-bit words in SPI boot type, as follows:

0x00000LLL	First word. Data word length or data word count of the data block.
0xAAAAAAAA	Second word. Data block start address.
0x000000DD	Third word. Tag of data block type.

Data structures and packing schemes in each data block for multiprocessor systems are the same as those used for single-processor systems.

The loader kernel examines the tag to determine the type of data or instruction being loaded. [Table 4-1](#) lists ADSP-21161N DSP tags.

Table 4-4. ADSP-21161N DSP Loader Tags

Tag Number	Block Type	Tag Number	Block Type
0x0000	final init	0x000E	init pm48
0x0001	zero dm16	0x000F	zero dm64
0x0002	zero dm32	0x0010	init dm64
0x0003	zero dm40	0x0012	init pm64
0x0004	init dm16	0x0013	init pm8 ext
0x0005	init dm32	0x0014	init pm16 ext
0x0007	zero pm16	0x0015	init pm32 ext
0x0008	zero pm32	0x0016	init pm48 ext
0x0009	zero pm40	0x0017	zero pm8 ext
0x000A	zero pm48	0x0018	zero pm16 ext
0x000B	init pm16	0x0019	zero pm32 ext
0x000C	init pm32	0x001A	zero pm48 ext
0x0011	zero pm64		

Interrupt Vector Table Location

If the ADSP-21161, ADSP-2126x, ADSP-2136x DSP is booted from an external source (EPROM, host, link port, or SPI), the interrupt vector table is located in internal memory. If the processor is not booted and executes from external memory (no boot mode), the vector table must be located in external memory.

The `IIVT` bit in the `SYSCON` control register can be used to override the booting mode in determining where the interrupt vector table is located. If the processor is not booted (no-boot mode), setting `IIVT` to 1 selects an internal vector table, and setting `IIVT=0` selects an external vector table. If the processor is booted from an external source (any boot mode other than no-boot), `IIVT` has no effect. The default initialization value of `IIVT` is zero.

Include **Format Files for Booting**

Include format files support booting for ADSP-21161N DSPs. The word width (8-bit, 16-bit) depends on the specified boot type. Specify word width with the loader `LDR Format Width` switch. Refer to [Table 4-10 on page 4-30](#) for information on command line switches.

Similar to Intel hex-32 output, loader output files in include format have four basic parts in the following order:

1. PROM kernel
2. Multiprocessor jump table
3. User application code
4. Saved user code in conflict with the kernel code

The kernel code is the first part. The processor jump table (including the processor IDs and the associated jump addresses) follows. User application code is followed by the saved user code.

Refer to [“Loader Output Files in Include Format” on page A-10](#) for information about `include` format files.

Boot Types

The following sections describe these boot types:

- “EPROM Booting”
- “Host Booting” on page 4-16
- “Link Port Booting” on page 4-20
- “SPI Port Booting” on page 4-22
- “No-Boot Mode” on page 4-23

 For multiprocessor booting, refer to “Multiprocessor Booting” on page 4-24.

EPROM Booting

EPROM booting through the external port is selected when the $\overline{\text{EB00T}}$ input is high and the $\overline{\text{LB00T}}$ input is low. These settings cause the $\overline{\text{BMS}}$ pin to become an output, serving as chip select for the EPROM.

The DMAC10 control register is initialized for booting packing boot data into 48-bit instructions. EPROM boot mode uses channel 10 of the I/O processor’s DMA controller to transfer the instructions to internal memory. For EPROM booting, the processor reads data from an 8-bit external EPROM.

After the boot process loads 256 words into memory locations 0×40000 through $0 \times 400FF$, the processor begins to execute instructions. Because most DSP programs require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the

application. VisualDSP++ includes loading routines (loader kernels) that can load entire programs. See “[Boot Kernels](#)” on [page 4-6](#) for more information.

-  Refer to the *ADSP-21161 SHARC DSP Hardware Reference* for detailed information on DMA and system configurations.
-  Be aware that DMA channel differences between the ADSP-21161, ADSP-2126x, ADSP-2136x DSPs and previous SHARC family DSPs (ADSP-2106x) account for booting differences. Even with these differences, the ADSP-21161 DSP supports the same boot capability and configuration as the ADSP-2106x DSPs. The `DMAC` register default values differ because the ADSP-21161, ADSP-2126x, ADSP-2136x DSP has additional parameters and different DMA channel assignments. EPROM boot mode uses `EPB0`, DMA channel 10. Similar to ADSP-2106x DSPs, the ADSP-21161, ADSP-2126x, ADSP-2136x DSP boots from `DATA23-16`.

The DSP determines the booting mode at reset from the `EBOOT`, `LB00T`, and `BMS` pin inputs. When `EBOOT=1` and `LB00T=0`, the DSP boots from an EPROM through the external port and uses `BMS` as the memory select output. For information on boot mode selection, see the Boot Memory Select pin descriptions in [Table 4-4 on page 4-10](#) and [Table 4-2 on page 4-5](#).

-  When using any of the power-up booting modes, address `0x40004` should not contain a valid instruction since it is not executed during the booting sequence. Place a `NOP` or `IDLE` instruction at this location.

EPROM booting (boot space 8M x 8-bit) through the external port requires that an 8-bit wide boot EPROM be connected to the processor data bus pins 23-16 (`DATA23-16`). The DSP's lowest address pins should be connected to the EPROM address lines. The EPROM's chip select should be connected to `BMS`, and its output enable should be connected to `RD`.

In a multiprocessor system, the $\overline{\text{BMS}}$ output is driven by the ADSP-21161N bus master only. This allows the wired OR of multiple $\overline{\text{BMS}}$ signals for a single common boot EPROM.

 Systems can boot any number of ADSP-21161N DSPs from a single EPROM using the same code for each processor or differing code for each processor.

During reset, the ACK line is internally pulled high with the equivalent of an internal 20K ohm resistor and is held high with an internal keeper latch. It is not necessary to use an external pull-up resistor on the ACK line during booting or at any other time.

The RBWS and RBAM fields of the WAIT register are initialized to perform asynchronous access and generate seven wait states (eight cycles total) for the EPROM access in external memory space. Note that wait states defined for boot memory are applied to $\overline{\text{BMS}}$ asserted accesses.

Table 4-2 shows how DMA channel 10 parameter registers are initialized at reset. The count register (CEP0) is initialized to $0x0100$ to transfer 256 words to internal memory. The external count register (ECEP0), used when external addresses (BMS space) are generated by the DMA controller, is initialized to $0x0600$ ($0x0100$ words at six bytes per word). The DMAC10 control register is initialized to $0x00\ 0561$.

The default value sets up external port transfers as follows:

- $\text{DEN} = 1$, external port enabled
- $\text{MSWF} = 0$, LSB first
- $\text{PMODE} = 101$, 8-bit to 48-bit packing, Master = 1
- $\text{DTYPE} = 1$, three column data

Table 4-5. DMA Channel 10 Parameter Register Initialization for EPROM Booting

Parameter Register	Initialization Value
IIEP0	0x40000
IMEP0	Uninitialized (increment by 1 is automatic)
CEP0	0x100 (256 instruction words)
CPEP0	Uninitialized
GPEP0	Uninitialized
EIEP0	0x800000
EMEP0	Uninitialized (increment by 1 is automatic)
ECEP0	0x600 (256 words x 6 bytes/word)

The following sequence occurs at system start-up, when the DSP $\overline{\text{RESET}}$ input goes inactive

1. The DSP goes into an idle state, identical to that caused by the IDLE instruction. The program counter (PC) is set to address 0x40004.
2. The DMA parameter registers for channel 10 are initialized as shown in [Table 4-2](#).
3. $\overline{\text{BMS}}$ becomes the boot EPROM chip select.
4. 8-bit Master Mode DMA transfers from EPROM to the first internal memory address on the external port data bus lines 23-16.
5. The external address lines (ADDR23-0) start at 0x800000 and increment after each access.
6. The $\overline{\text{RD}}$ strobe asserts as in a normal memory access with seven wait states (eight cycles).

Loader Guide

The DSP's DMA controller reads the 8-bit EPROM words, packs them into 48-bit instruction words, and transfers them to internal memory until 256 words have been loaded. The EPROM is automatically selected by the $\overline{\text{BMS}}$ pin; other memory select pins are disabled.

The Master DMA internal and external count registers (ECEP0/CEP0) decrements after each EPROM transfer. When both counters reach zero, the following wake-up sequence occurs:

1. DMA transfers stop.
2. External port DMA channel 10 interrupt (EP0I) is activated.
3. $\overline{\text{BMS}}$ is deactivated, and normal external memory selects are activated.
4. The DSP vectors to the EP0I interrupt vector at $0\text{x}40050$.

At this point the DSP has completed its booting mode and is executing instructions normally. The first instruction at the EP0I interrupt vector location, address $0\text{x}40050$, should be an RTI (return from interrupt). This process returns execution to the reset routine at location $0\text{x}40005$ where normal program execution can resume. After reaching this point, a program can write a different service routine at the EP0I vector location $0\text{x}40050$.

Host Booting

The DSP can boot from a host processor through the external port. Host booting is selected when the EBOOT and LBOOT inputs are low and $\overline{\text{BMS}}$ is high. Configured for host booting, the DSP enters the slave mode after reset and waits for the host to download the boot program.

The DMAC10 control register is initialized for booting, packing boot data into 48-bit instructions. Channel 10 of the I/O processor's DMA controller is used to transfer instructions to internal memory. DSPs accept data 8-, 16-, or 32-bit host microprocessor (or other external devices).

After the boot process loads 256 words into memory locations 0x40000 through 0x400FF, the DSP begins executing instructions. Because most DSP programs require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the application. VisualDSP++ includes loading routines (loader kernels) that can load entire programs. These routines come with the development tools. Refer to [“Boot Kernels” on page 4-6](#) for more information.

-  Refer to the *ADSP-21161 SHARC DSP Hardware Reference* for detailed information on DMA and system configurations.
-  DMA channel differences between ADSP-21161, ADSP-2126x, ADSP-2136x DSPs and previous SHARC family DSPs (ADSP-2106x) account for booting differences. Even with these differences, ADSP-21161 DSPs support the same boot capability and configuration as ADSP-2106x DSPs. The `DMAC10` register default values differ because the ADSP-21161, ADSP-2126x, ADSP-2136x DSP has additional parameters and different DMA channel assignments. Host boot mode uses `EPB0`, DMA channel 10.

The DSP determines the boot mode at reset from the `EBOOT`, `LB00T`, and `BMS` pin inputs. When `EB00T=0`, `LB00T=0`, and `BMS=1`, the DSP boots from a host through the external port. Refer to [Table 4-1 on page 4-5](#) and [Table 4-2 on page 4-5](#) on boot mode selection.

When using any of the power-up booting modes, address 0x40004 should not contain a valid instruction since it is not executed during the booting sequence. Place a `NOP` or `IDLE` instruction at this location.

During reset, the DSP `ACK` line is internally pulled high with an equivalent 20K ohm resistor and is held high with an internal keeper latch. It is not necessary to use an external pull-up resistor on the `ACK` line during booting or at any other time.

Table 4-5 on page 4-15 shows how the DMA channel 10 parameter registers are initialized at reset for host booting. The internal count register (CEP0) is initialized to 0x0100 to transfer 256 words to internal memory. The DMAC10 control register is initialized to 0000 0161.

The default value sets up external port transfers as follows:

- DEN = 1, external port enabled
- MSWF = 0, LSB first
- PMODE = 101, 8-bit to 48-bit packing
- DTYPE = 1, three column data

Table 4-6. Host Booting – DMA Channel 10 Parameter Register Initialization

Parameter Register	Initialization Value
IIEP0	0x0004 0000
IMEP0	Uninitialized (increment by 1 is automatic)
CEP0	0x0100 (256 instruction words)
CPEP0	Uninitialized
GPEP0	Uninitialized
EIEP0	Uninitialized
EMEP0	Uninitialized
ECEP0	Uninitialized

At system start-up, when the DSP $\overline{\text{RESET}}$ input goes inactive, the following sequence occurs:

1. The DSP goes into an idle state, identical to that caused by the `IDLE` instruction. The program counter (PC) is set to address `0x40004`.
2. The DMA parameter registers for channel 10 are initialized as shown in [Table 4-5 on page 4-15](#).
3. The host uses `HBR` and `CS` to arbitrate for the bus.
4. The host can write to `SYSCON` (if `HBG` and `READY` are returned) to change boot width from default.
5. The host writes boot information to external port buffer 0.

The slave DMA internal count register (`CEP0`) decrements after each transfer. When `CEP0` reaches zero, the following wake-up sequence occurs:

1. The DMA transfers stop.
2. The external port DMA channel 10 interrupt (`EP0I`) is activated.
3. The DSP vectors to the `EP0I` interrupt vector at `0x40050`.

At this point the DSP has completed its booting mode and is executing instructions normally. The first instruction at the `EP0I` interrupt vector location, address `0x40050`, should be an `RTI` (return from interrupt). This process returns execution to the reset routine at location `0x40005` where normal program execution can resume. After reaching this point, a program can write a different service routine at the `EP0I` vector location `0x40050`.

Link Port Booting

Link port booting uses DMA channel 8 of the I/O processor to transfer instructions to internal memory. In this boot mode, the DSP receives 4-bit wide data in link buffer 0.

After the boot process loads 256 words into memory locations $0x40000$ through $0x400FF$, the DSP begins to execute instructions. Because most DSP programs require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the application. VisualDSP++ includes loading routines (loader kernels) that load an entire program through the selected port. Refer to [Table 4-3 on page 4-6](#) for more information.

-  Refer to the *ADSP-21161 SHARC DSP Hardware Reference* for detailed information on DMA and system configurations.
-  DMA channel differences between ADSP-21161, ADSP-2126x, ADSP-2136x DSPs and previous SHARC family DSPs (ADSP-2106x) account for booting differences. Even with these differences, ADSP-21161 DSPs support the same boot capability and configuration as ADSP-2106x DSPs.

The DSP determines the booting mode at reset from the $EBOOT$, $LB00T$, and $\overline{BMS}$ pin inputs. When $EBOOT=0$, $LB00T=1$, and $\overline{BMS}=1$, the DSP boots through the link port. For information on boot mode selection, see [Table 4-1 on page 4-5](#) and [Table 4-2 on page 4-5](#).

-  When using any of the power-up booting modes, address $0x40004$ should not contain a valid instruction since it is not executed during the booting sequence. Place a `NOP` or `IDLE` instruction at this location.

In link port booting, the DSP gets boot data from another DSP link port or 4-bit wide external device after system power-up.

The external device must provide a clock signal to the link port assigned to link buffer 0. The clock can be any frequency up to the DSP clock frequency. The clock falling edges strobe the data into the link port. The most significant 4-bit nibble of the 48-bit instruction must be downloaded first.

Table 4-5 on page 4-15 shows how the DMA channel 8 parameter registers are initialized at reset. The count register (CLB0) is initialized to 0x0100 to transfer 256 words to internal memory. The LCTL register is overridden during link port booting to allow link buffer 0 to receive 48-bit data.

Table 4-7. Link Port Booting – DMA Channel 8 Parameter Register Initialization

Parameter Register	Initialization Value
IILB0	0x0004 0000
IMLB0	Uninitialized (increment by 1 is automatic)
CLB0	0x0100 (256 instruction words)
CPLB0	Uninitialized
GPLB0	Uninitialized

In systems where multiple DSPs are not connected by the parallel external bus, booting can be accomplished from a single source through the link ports. To simultaneously boot all the DSPs, make a parallel common connection to link buffer 0 on each of the processors. If a daisy chain connection exists between the processors' link ports, each DSP can boot the next processor in turn. Link buffer 0 must always be used for booting.

SPI Port Booting

Serial Peripheral Interface (SPI) port booting uses DMA channel 8 of the I/O processor to transfer instructions to internal memory. In this boot mode, the DSP receives 8-bit wide data in the SPIRX register.

During the booting process, the program loads 256 words into memory locations 0x40000 through 0x400FF. The DSP subsequently begins executing instructions. Because most DSP programs require more than 256 words of instructions and initialization data, the 256 words typically serve as a loading routine for the application. VisualDSP++ includes loading routines (loader kernels) that load an entire program through the selected port. See “[Boot](#)ing” on [page 4-3](#) for more information.

 Refer to the *ADSP-21161 SHARC DSP Hardware Reference* for detailed information on DMA and system configurations. For information about SPI slave booting, refer to EE-177, entitled *SHARC SPI Booting*, located on the Analog Devices DSP Web site.

The DSP determines the booting mode at reset from the EBOOT, LBOOT, and $\overline{\text{BMS}}$ pin inputs. When EBOOT=0, LBOOT=1, and $\overline{\text{BMS}}$ =0, the DSP boots through its SPI port. For information on boot mode selection, see [Table 4-1 on page 4-5](#) and [Table 4-2 on page 4-5](#).

 When using any of the power-up booting modes, address 0x40004 should not contain a valid instruction because it is not executed during the booting sequence. Place a NOP or IDLE instruction placed at this location.

For SPI port booting, the DSP gets boot data after system power-up from another DSP's SPI port or another SPI compatible device.

[Table 4-8](#) shows how the DMA channel 8 parameter registers are initialized at reset. The SPI Control Register (SPICTL) is configured to 0x0A001F81 upon reset during SPI boot.

This configuration sets up the SPIRX register for 32-bit serial transfers. The SPIRX DMA channel 8 parameter registers are configured to DMA in 0x180 32-bit words into internal memory normal word address space starting at 0x40000. Once the 32-bit DMA transfer completes, the data is then accessed as 3 column, 48-bit instructions. The DSP executes a 256 word (0x100) loader kernel upon completion of the 32-bit, 0x180 word DMA.

For 16-bit SPI hosts, two words are shifted into the 32-bit receive shift register before a DMA transfer to internal memory occurs. For 8-bit SPI hosts, four words are shifted into the 32-bit receive shift register before a DMA transfer to internal memory occurs.

Table 4-8. SPI Port Booting – DMA Channel 8 Parameter Register Initialization

Parameter Register	Initialization Value
IISRX	0x0004 0000
IMSRX	Uninitialized (increment by 1 is automatic)
CSRX	0x0180 (256 instruction words)
GPSRX	Uninitialized

No-Boot Mode

No-boot mode causes the processor to start fetching and executing instructions at address 0x200004 in external memory space. In no-boot mode, the processor does not boot-load and all DMA control and parameter registers are set to their default initialization values.

Multiprocessor Booting

A multiprocessor system can be booted from a host processor, external EPROM, through a link port, or through an SPI port.

 Currently, the loader generates single-processor loader files for host, link, and SPI port booting. As a result, the loader supports multiprocessor EPROM booting only, and you must modify the application code to properly set up multiprocessor booting in host, link, and SPI port booting modes.

To set up DMA processes for booting a single processor, refer to:

- [“EPROM Booting” on page 4-12](#)
- [“Host Booting” on page 4-16](#)
- [“Link Port Booting” on page 4-20](#)
- [“SPI Port Booting” on page 4-22](#)

 Refer to the *ADSP-21161 SHARC DSP Hardware Reference* for detailed information on DMA and system configurations.

Multiprocessor EPROM Booting

There are two methods by which a multiprocessor system can be booted: booting from a single EPROM, and sequential EPROM booting. Regardless of the method, processors perform the following steps.

1. Arbitrate for the bus.
2. Upon becoming bus master, DMA the 256 word boot stream.
3. Release the bus.
4. Execute the loaded instructions.

Booting From a Single EPROM

The $\overline{\text{BMS}}$ signals from each DSP may be wire ORed together to drive the EPROM's chip select pin. Each processor can boot in turn, according to its priority. When the last processor has finished booting, it must inform the other DSPs (which may be in the idle state) that program execution can begin (if all DSPs are to begin executing instructions simultaneously).

When multiple DSPs boot from a single EPROM, the processors can boot identical code or different code from the EPROM. If the processors load differing code, use a jump table (based on processor ID) to select the code for each processor.

Sequential EPROM Booting

Set the EB00T pin of the DSP with $\text{IDx}=1$ high for EPROM booting.

The other DSPs should be configured for host booting ($\text{EB00T}=0$, $\text{LB00T}=0$, and $\text{BMS}=1$), leaving them in the idle state at startup and allowing the DSP with $\text{IDx}=1$ to become bus master and boot itself. Connect the $\overline{\text{BMS}}$ pin of DSP #1 only to the EPROM's chip select pin. When DSP #1 has finished booting, it can boot the remaining processors by writing to their external port DMA buffer 0 (EPB0) via the multiprocessor memory space.

The loader can produce boot-loadable files that permit SHARC DSPs in a multiprocessor system to boot from a single EPROM. In such a system, the processors BMS signals are ORed together to drive the EPROM's chip select pin. Each DSP boots in turn, according to its priority. When the last DSP has finished booting, it must inform the other DSPs to begin program execution.

Running Loader

This section provides reference information on loader options specified from VisualDSP++ or from a command line.

Load Page

When developing a “DSP loader file” project in VisualDSP++, modify the default option settings on the **Load** page of the **Projects Options** dialog box. [Figure 3-1 on page 3-27](#) shows a representative screen shot of the Load page for an (ADSP-21160) SHARC processor. For information specific to the ADSP-21161N processor, refer to the VisualDSP++ online help for that processor.

VisualDSP++ invokes the `elfloader` utility to build the output file. Dialog box buttons and fields correspond to command line switches and parameters (see [Table 4-10 on page 4-30](#)). Use the **Additional Options** box to enter options that have no dialog box equivalent.

Loader Command Line

This section provides reference information on the loader command line. A list of all switches and their descriptions appears in [Table 4-10 on page 4-30](#).

Invoke the `elfloader.exe` loader utility from the command line to generate a boot-loader file. This section describes the command line only.

Use the following syntax for the SHARC DSP loader command line.

```
elfloader sourcefile -proc ADSP-21161 -switch [-switch ...]
```

where:

- *sourcefile* – Identifies the executable file (.DXE) to be processed for a single-processor boot-loadable file. A file name can include the drive, directory, file name and file extension. Enclose long file names within straight quotes, “long file name”.
- ADSP-21161 – Specifies the processor for which the boot-loader file is to be built.
- -switch – Processes the optional switch. The loader has many switches to select operations and modes.



Command line switches are not case sensitive and may be placed in the command line in any order.

Single-Processor Systems

```
elfloader Input.dxe -bSPI -proc ADSP-21161
```

In the above example for a single-processor system, the command line runs the loader with:

- Input.dxe – Identifies the executable file to process into a boot-loadable file. Note that the absence of the -o switch causes the output file name to default to Input.ldr.
- -bSPI – Specifies SPI port booting as the boot type for the boot-loadable file.
- -proc ADSP-21161 – Specifies ADSP-21161 as the target processor.

Multiprocessor Systems

```
elfloader -proc ADSP-21161 -bprom -idlexe=Input1.dxe  
-id2exe=Input2.dxe <switch>
```

Running the Loader

In the above example for a multiprocessor system, the command line runs the loader with:

- `-proc ADSP-21161` – Specifies ADSP-21161 as the target processor.
- `-bprom` – Specifies EPROM booting as the boot type for the boot-loadable file.
- `id1exe=Input1.dxe` – Identifies the executable file to processed for ID#1 as Input1.dxe.
- `id2exe=Input2.dxe` – Identifies the executable file to processed for ID#2 as Input2.dxe.

File Names

Many loader switches take a file name as an optional parameter.

[Figure 4-10](#) lists expected file types, names, and extensions.

File searches are important in the loader process. The loader supports relative and absolute directory names, and default directories. File searches occur as follows.

- *Specified path* – If relative or absolute path information is included in a file name, the loader searches only in that location for the file.
- *Default directory* – If path information is not included in the file name, the loader searches for the file in the current working directory.

When providing an input or output file name as a command line parameter, follow these guidelines.

- Enclose long file names within straight quotes, “long file name”.
- Append the appropriate file extension to each file.

File Extensions

The loader follows the conventions shown in [Table 4-9](#) for file extensions.

Table 4-9. File Extensions

Extension	File Description
.DXE	Executable files and boot kernel files. The loader recognizes overlay memory files (.OVL) and shared memory files (.sm), but does not expect these files on the command line. Place .OVL and .SM files in the same directory as the .DXE file that refers to them so the loader can find them when processing the .LDR file. The .OVL and .SM files may also be placed in the .OVL and .SM file output directory specified in the .LDF file
.LDR	Loader output file

ADSP-21161, ADSP-2126x, ADSP-2136x Loader Command Line Switches

A descriptions of each ADSP-21161, ADSP-2126x, ADSP-2136x loader switch appears in [Table 4-10](#).



Do not place space between some switches and their parameters. For example, there is no space between `-b` and `PROM`.

Processor ID Numbers

When used with a single-processor system, there is only one input (.DXE) file without any prefix and suffix to the input file name, for example:

```
elfloader -proc ADSP-21161 -bprom Input.dxe <switches>
```

A multiprocessor system requires a distinct processor ID number for each input file in the command line. A processor ID is provided via the `-id#exe=` switch (Not applicable to the ADSP-2126x, ADSP-2136x), where `#` is an ID number (1 to 6).

Running the Loader

Table 4-10. ADSP-21161N Loader Command Line Items ¹

Switch	Description
<i>sourcefile</i>	A file name without a switch specifies an executable file for a single-processor boot-loadable file. For multiprocessor system boot-loadable files, use the <code>-id#exe=file</code> switch, where # is a number (1 to 6, inclusive). The loader accepts .DxE files only and automatically finds and processes associated .OVL and .SM files. Place .OVL and .SM files in the specified directory and in the current working directory.
<code>-proc ADSP-21161</code>	Specifies the processor. <code>-proc ADSP-21161</code> is the preferred switch.
<code>-bprom</code> <code>-bhost</code> <code>-bspi</code> <code>-blink</code>	Specifies the boot mode. The <code>-b</code> switch directs the loader to prepare a boot-loadable file for the specified boot mode. The valid modes (boot types) are PROM, host, SPI, and link. If <code>-b</code> does not appear on the command line, the default is <code>-bprom</code> . To use a custom boot kernel, the boot type selected with the <code>-b</code> switch must correspond with the boot kernel selected with the <code>-l</code> switch. Otherwise, the loader automatically selects a default boot kernel based on the selected boot type.
<code>-bspislave</code> <code>-bspimaster</code> <code>-bspiprom</code> <code>-bspiflash</code>	These additional <code>-b</code> switches are only applicable to the ADSP-2126x and ADSP-2136x. Directs the loader to prepare a boot-loadable file for the specified boot mode: SPI SLAVE, SPI MASTER, SPI PROM, or SPI FLASH.
<code>-efilename</code>	Except shared memory. The <code>-e</code> switch omits the listed shared memory (.SM) file from the output loader file.

Loader for ADSP-21161N SHARC Processors

Table 4-10. ADSP-21161N Loader Command Line Items (Cont'd) ¹

Switch	Description
-fhex -fASCII -fbinary -finclude -fS1 -fS2 -fS3	Species the format of the boot-loadable file. The <code>-f</code> switch prepares a boot-loadable file in the specified <i>format</i> . Available <i>format</i> selections include hex (Intel hex-32), S1, S2, S3 (Motorola S-records), ASCII, include, and binary. Available formats depend on the <code>-b</code> switch boot type selection. For a PROM boot type, select a hex, S1, S2, S3, ASCII, or include format. For host or link booting, select an ASCII, binary, or include format. For SPI booting, select an ASCII or binary format. If the <code>-f</code> switch does not appear on the command line, the default boot type format is hex for PROM, and ASCII for host, link, or SPI.
-h or -help	Command line help. This switch outputs the list of command line switches to standard output and exits. Combining the <code>-h</code> switch with <code>-proc ADSP-21161</code> ; for example, <code>elfloader -proc ADSP-21161 -h</code> , yields the syntax and switches for the elfloader utility for ADSP-21161, ADSP-2126x, ADSP-2136x DSPs. By default, the <code>-h</code> switch alone provides help for the loader driver.
-LDR format width #	This sets up the .LDR file word width. By default, the word width for PROM and host is 8, for link is 16, and for SPI is 32. The valid word widths for the various boot modes are: PROM (8 for hex or ASCII format, 8 or 16 for include format), host (8 or 16 for ASCII or binary format, 16 for include format), link (16 for ASCII, binary, or include format), SPI (8, 16, or 32 for hex or ASCII format).
-si-version <none x.x>	Sets version for the build, with <code>x.x</code> being the revision number for the hardware. The default setting is the latest silicon revision.

Table 4-10. ADSP-21161N Loader Command Line Items (Cont'd) ¹

Switch	Description
<code>-id#exe=filename</code>	Specifies the multiprocessor ID. The <code>-id</code> switch directs the loader to use the processor ID (<code>#</code>) for the corresponding executable files when producing a boot-loadable file for a multiprocessor system. This switch is not applicable to the ADSP-2126x, ADSP-2136x. This switch is used only to produce a boot-loadable file that boots multiple DSPs from a single EPROM. Valid values for <code>#</code> are 1, 2, 3, 4, 5, and 6. Do not use this switch for single-processor systems; instead, use the executable <i>filename</i> as a parameter without a switch. Refer to “Processor ID Numbers” on page 4-29 for details.
<code>-id#exe=N</code>	Points the ID(N) loader jump table entry to the ID(<code>#</code>) image. If the executable for the ID(<code>#</code>) processor is identical to the executable of the ID(N) processor, this switch can be used to set the PROM start address of ID(<code>#</code>) to be the same as for ID(N). This effectively reduces the size of the loader file by providing a single copy of an executable to two or more processors in a multiprocessor system. This switch is not applicable to the ADSP-2126x, ADSP-2136x.

Table 4-10. ADSP-21161N Loader Command Line Items (Cont'd) ¹

Switch	Description
<code>-l kernelfile</code>	The <code>-l</code> switch directs the loader to use the specified <i>kernelfile</i> for the boot-loading routine in the output boot-loadable file. Note that the boot kernel file selected with this switch must correspond to the boot type (selected with the <code>-b</code> switch). If the <code>-l</code> switch does not appear on the command line, the loader searches for a default boot kernel file in the installation directory. Based on the boot type, the loader searches for the boot kernel file in the installation directory (...\\21k\\ldr) as follows: For a PROM boot type (<code>-bprom</code>), the default boot kernel file is <code>161_prom.dxe</code>. For a host boot type (<code>-bhost</code>), the default boot kernel file is <code>161_host.dxe</code>. For a link boot type (<code>-blink</code>), the default boot kernel file is <code>161_link.dxe</code>. For an SPI boot type (<code>-bspi</code>), the default boot kernel file is <code>161_SPI.dxe</code>.
<code>-NoKernel [message1 message2]</code>	This switch is only applicable to the ADSP-2126x, ADSP-2136x. Directs the loader not to include the kernel code in the loader file. Also, optionally puts two 32-bit hex messages in the final block header.
<code>-o filename</code>	Specifies an output file. The <code>-o</code> switch directs the loader to use the specified <i>filename</i> for the name of the loader output file. If the <code>-o</code> switch is not specified, the default name is <i>sourcefile.LDR</i>.

Running the Loader

Table 4-10. ADSP-21161N Loader Command Line Items (Cont'd) ¹

Switch	Description
<i>-paddress</i>	Specifies the PROM start address. This EPROM address corresponds to 0x8000000 in the ADSP-21161, ADSP-2126x, ADSP-2136x. The <i>-p</i> switch starts the boot-loadable file at the specified address in the EPROM. If the <i>-p</i> switch does not appear on the command line, the loader starts the EPROM file at address 0x0.
<i>-ttimeout</i>	(Host boot type only) Timeout cycles. The <i>-t</i> switch (for example, <i>-t100</i>) limits the number of cycles that the DSP can spend initializing external memory with zeros. Valid values range from 3 to 32765 cycles; 32765 is the default value. The <i>timeout</i> value is directly related to the number of cycles the DSP locks the bus for boot-loading, instructing the DSP to lock the bus for no more than two times the <i>timeout</i> number of cycles. When working with a fast host that cannot tolerate being locked out of the bus, use a relatively small <i>timeout</i> value.
<i>-v</i>	Verbose loader messages. The <i>-v</i> switch outputs status information as the loader processes files.
<i>-version</i>	Directs the loader to show its version information. Type “elfloader -version” to display the version of the loader drive. Type “elfloader -proc ADSP-21060 -version” to display version information of both loader drive and SHARC loader.

¹ Switches are optional. Items in *italics* are user defined.

In the following example, the loader processes the input file `Input1.dxe` for the processor with an ID of 1, and it processes the input file `Input2.dxe` for the processor with an ID of 2.

```
elfloader -proc ADSP-21161 -bprom -id1exe=Input1.dxe  
-id2exe=Input2.dxe <switch>
```

or

Loader for ADSP-21161N SHARC Processors

```
elfloader -proc ADSP-21161 -bprom -idlexe = Input1.dxe-id2exe=1  
<switch>
```

In this case, the processor with ID of 2 shares the same jump table entry with the processor with ID of 1. The code from `Input2.dxe` will be loaded into both processors, but the loader file contains only one copy of the code from `Input1.dxe`.

5 SPLITTER

The VisualDSP++ 3.5 splitter supports both the ADSP-21xxx SHARC and the ADSP-TSxxx TigerSHARC processors. The TigerSHARC splitter creates a 32-bit image file while the SHARC creates a 48-/40-bit image file.

Once the program is debugged, use the splitter utility (`elfspl21k.exe`) to generate non-bootable PROM image files, which execute from DSP external memory. These files are often used with ADSP-21065L DSP systems, which have limited internal memory.

In most instances, developers working with SHARC DSPs should use the loader instead of the splitter – the exception is when a SHARC system must execute instructions from external memory.

This chapter provides the following information:

- “[Splitter Guide](#)” on page 5-2 provides usage information
- “[Splitter Command Line Reference](#)” on page 5-4 provides reference information on splitter commands

For information about the DSP program generation process, refer to “[Program Development Flow](#)” on page 1-1.

Splitter Guide

The splitter (`elfspl21k.exe`) processes executable files, producing one or more non-bootable PROM image files. Splitter operation relies on splitter options, which control the processing of the executable files into output files.

There are two ways to run the splitter:

- Command line invocation. Refer to [“Splitter Command Line Reference” on page 5-4](#).
- VisualDSP++ IDE. Generate non-bootable PROM image files by selecting **DSP splitter file** as the project output. Specify splitter options from the **Split** page of the **Project Options** dialog box. Refer to online Help for details.

Split Tab

When developing a DSP project, you can modify splitter default option settings directly from within the VisualDSP++ IDE.

Splitter options are available from the **Split** tab (also called the Split property page) of the **Project Options** dialog box. Settings correspond to `elfspl21k` command-line switches. Refer to [Figure 5-1 on page 5-3](#).

For more information, refer to VisualDSP++ online Help.

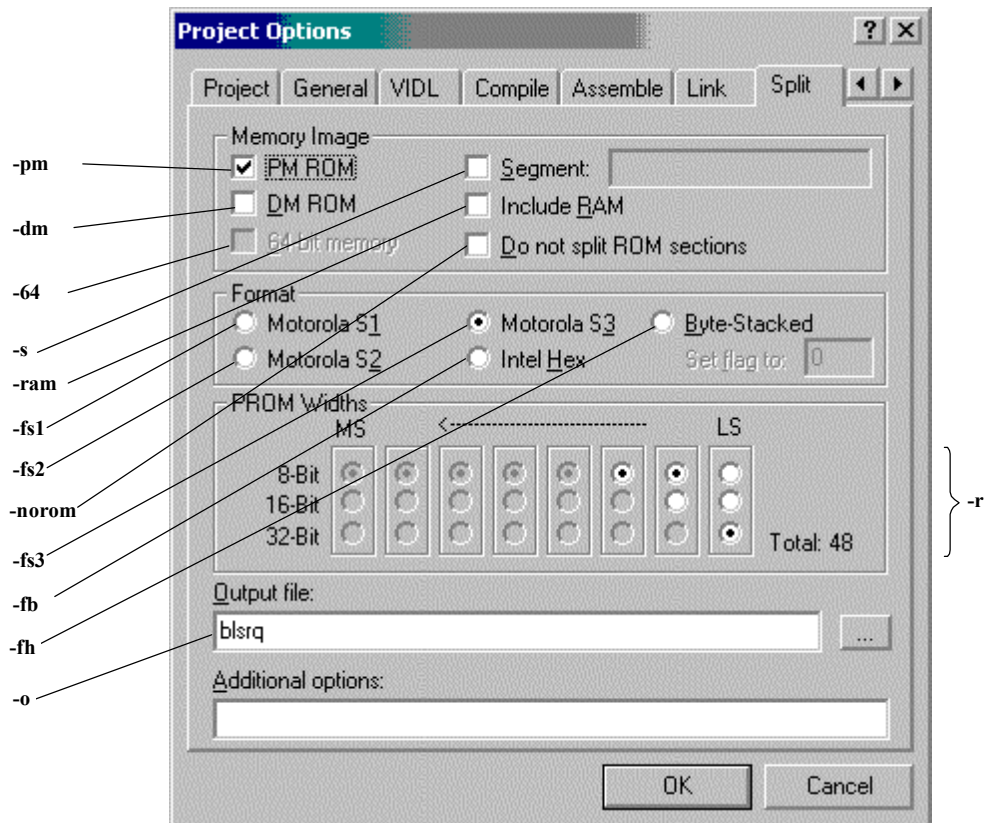


Figure 5-1. Split Page on Project Options

Splitter Command Line Reference

The splitter (`elfspl21k.exe`) generates non-bootable PROM image files for SHARC DSPs by processing executable (`.DXE`) files. Splitter output is a set of PROM files with `.S_#`, `.H_#`, or `.STK` file extensions.

 When using the splitter from the VisualDSP++ IDE, the settings on the **Split** tab of the **Project Options** dialog box correspond to splitter command line switches. For more information, refer to [“Split Tab” on page 5-2](#) and VisualDSP++ online Help.

elfspl21k Command Line Syntax

The splitter command line requires the following syntax:

```
elfspl21k [-switch ...] -pm &| -dm &| -64 &| executable
```

or

```
elfspl21k [-switch ...] -s segment executable
```

where:

- `[-switch...]` – Specifies the name of one or more optional switches to be processed. These switches select the operations and modes for the splitter.
- `-pm &| -dm &| -64` – The `&|` symbol between these three switches indicates AND/OR. The splitter command line must include one or more of the `-pm`, `-dm`, and `-64` switches (or the `-s` switch).
- `-s segment` – The splitter command line must include one or more of the `-pm`, `-dm`, and, `-64` switches or the `-s` switch. The `-s` switch can be used without the `-pm`, `-dm`, or `-64` switch.
- `executable` – Specifies the name of the executable file (`.DXE`) to be processed into a boot-loadable file for a single-processor system.

Items shown between bracket characters [] are optional. Items in *italic font* are user defined and are described with each switch.

TigerSHARC processors do not have -pm, -dm, or -64 switches.

[Table 5-1](#) lists file types, file names, and extensions that the splitter expects on the elfspl21k command line.

Splitter command line items and their descriptions appear in [Table 5-2 on page 5-8](#).

Command Lines Rules

Follow these rules when using command line invocation:

- Most items in the splitter command line are not case sensitive; for example, -pm and -PM are interchangeable. However, the names of memory segment names must be identical, including case, to the names used in the executable.
- Switches may be used in any order.
- The name of the executable must appear at the end of the command. The name can include the drive, directory, file name, and file extension. Enclose long file names within straight quotes; for example, "long file name".

Example Splitter Command Lines

```
elfspl21k -pm -o pm_stuff my_proj.dxe -proc ADSP21161
elfspl21k -dm -o dm_stuff my_proj.dxe -proc ADSP21161
elfspl21k -64 -o dm_stuff my_proj.dxe -proc ADSP21161
elfspl21k -s seg-code -0 seg-code my_proj.dxe
```

Splitter Command Line Reference

In the above examples, each command line runs the splitter. The first command produces a PROM file for program memory. The second command produces a PROM file for data memory. The third command produces a PROM file for DATA64 memory. The fourth command produces a PROM file for section of `seg-code`.

The switches on these command lines are as follows.

-pm	Selects program memory (-pm), data memory (-dm), or DATA64 memory (-64) as sources in the executable for extraction and placement into the image. DATA64 memory does not apply to ADSP-2106x DSPs. The -pm, -dm, or -64 switches do not apply to ADSP-TSxxx DSPs. Because these are the only switches used to identify the memory source, the specified sources are PM, DM, or DATA64 memory segments. Because no other content switches appear on these command lines, the output file format defaults to a Motorola 32-bit format, and the PROM word width of the output defaults to 8 bits for all PROMs.
-dm	
-64	
-o pm_stuff	Specify names for the output files. Use different names so the output of a run does not overwrite the output of a previous run. The output names are <code>pm_stuff.s_#</code> and <code>dm_stuff.s_#</code> .
-o dm_stuff	
-o seg-code	
my_proj.dxe	Specifies the name of the input file (.DXE) to be processed into non-bootable PROM image files.

File Searches

Some switches take a file name as an optional parameter. “[Output File Extensions](#)” lists the expected file types, names, and extensions.

File searches are important in the splitter process. The splitter supports relative and absolute directory names, and default directories. File searches occur as follows.

- *Specified path* – If relative or absolute path information is included in a file name, the splitter searches only in that location for the file.
- *Default directory* – If path information is included in the file name, the splitter searches for the file in the current working directory.

To provide an input or output file name as a command line parameter, use these guidelines:

- Enclose long file names within straight quotes, “long file name”.
- Append the appropriate file extension to each file.

Output File Extensions

The splitter follows the conventions shown in [Table 5-1](#) for output file extensions.

Table 5-1. Output File Extensions

Extension	File Description
.S_#	Motorola S-record format file. The # indicates the position (0 = least significant, 1 = next-to-least significant, and so on). For info about Motorola S-record file format, refer to “ Splitter Output Files in Motorola S-Record Format ” on page A-11.

Splitter Command Line Reference

Table 5-1. Output File Extensions

Extension	File Description
.H_#	Intel hex-32 format file. The # indicates the position (0 = least significant, 1 = next-to-least significant, and so on). For information about Intel hex-32 file format, refer to “Splitter Output Files in Intel Hex-32 Format” on page A-13 .
.STK	Byte stacked format file. These files are intended for host transfer of data, not for PROMs. For more information about byte stacked file format, format files, , refer to “Splitter Output Files in Byte Stacked Format” on page A-13 .

Command Line Switches

This section provides reference information about valid switches and parameters used in the `elfspl21k` command line. A list of all command line items and their descriptions appears in [Table 5-2](#) .

Table 5-2. `elfspl21k` Utility Command Line Switches

Item	Description
<code>executable</code>	Specifies the executable file (.EXE) to be processed into splitter files. The file name must appear at the end of the command line.
<code>-64</code>	The <code>-64</code> (include DATA64 memory) switch directs the splitter to extract all segments declared as 64-bit memory segments from the .EXE file. This switch influences the operation of the <code>-ram</code> and <code>-norom</code> switches, adding 64-bit data memory as their target.
<code>-dm</code>	The <code>-dm</code> (include data memory) switch directs the splitter to extract memory segments declared as data memory ROM from the .EXE file. The <code>-dm</code> switch influences the operation of the <code>-ram</code> and <code>-norom</code> switches, adding data memory as their target.
<code>-o imagefile</code>	The <code>-o</code> (output file) switch directs the splitter to use <code>imagefile</code> as the name of the splitter output file(s). If not specified, the default name for the splitter output file is <code>“executable.ext”</code> , where <code>ext</code> depends on the output format.

Table 5-2. elfspl21k Utility Command Line Switches

Item	Description
-norom	The -norom (no ROM in PROM) switch directs the splitter to ignore ROM memory segments in <i>executable</i> when extracting information for the output image. The -dm and -pm switches select data memory or program memory. The operation of the -s switch is not influenced by the -norom switch.
-pm	The -pm (include program memory) switch directs the splitter to extract memory segments declared program memory ROM from the executable. The -pm switch influences the operation of the -ram and -norom switches, adding program memory as the target.
-r # [# ...]	The -r (PROM widths) switch specifies the number of PROM files and their width (in bits). The splitter can create PROM files for 8-, 16-, and 32-bit wide PROMs. The default width is 8 bits. Each # parameter specifies the width of one PROM file. Place # parameters in order from most significant to least significant. The sum of the # parameters must equal the bit width of the destination memory (40 bits for DM, 48 bits for PM, or 64 bits for 64-bit memory). Example: elfspl21k -dm -r 16 16 8 myfile.dxe This command extracts data memory ROM from myfile.dxe and creates the following output PROM files: myfile.s_0 8 bits wide, contains bits 7—0 myfile.s_1 16 bits wide, contains bits 23—8 myfile.s_2 16 bits wide, contains bits 39—24 The width of the three output files is 40 bits.
-ram	The -ram (include RAM in PROM) switch directs the splitter to extract RAM segments from <i>executable</i> . The -dm, -pm, and -64 switches select the memory. The -s switch is not influenced by the -ram switch.

Splitter Command Line Reference

Table 5-2. elfspl21k Utility Command Line Switches

Item	Description
-f h -f s1 -f s2 -f s3 -f b	The -f (PROM file format) switch directs the splitter to generate a non-bootable PROM image file in the specified <i>format</i>. Available selections for <i>format</i> include: - h = Intel hex-32 format - s1 = Motorola EXORciser format - s2 = Motorola EXORMAX format - s3 = Motorola 32-bit format - b = byte stacked format If the -f switch does not appear on the command line, the default format for the PROM file is Motorola 32-bit (s3). For information on file formats, see “Build Files” on page A-5.
-s <i>segmentname</i>	The -s (include memory segment) switch directs the splitter to extract the contents of the specified memory segment (<i>segment-name</i>). Use the -s <i>segmentname</i> switch as many times as needed. Each instance of the -s switch can specify only one <i>segmentname</i>. Do not use -s with (-pm, -dm, or -64).
-proc processor	Specifies the processor type to the splitter. Valid processors are: ADSP-2106x, ADSP-2116x, ADSP-2126x, ADSP-2136x , ADSP-TSxxx,
-u <i>number</i>	(Byte stacked format files only) The -u (user flags) switch, which may be used only in combination with the -f b switch, directs the splitter to use the <i>number</i> in the user-flags field of a byte stacked format file. If the -u switch is not used, the default value for <i>number</i> is 0. By default, <i>number</i> is decimal. If <i>number</i> is prefixed with “0x”, the splitter interprets the <i>number</i> as hexadecimal. For more information, see “Splitter Output Files in Byte Stacked Format” on page A-13.
-si-version <none x.x>	Sets version for the build, with x.x being the revision number for the hardware. The default setting is the latest silicon revision.
-version	Directs the splitter to show its version information.

A FILE FORMATS

VisualDSP++ development tools support many file formats, in some cases several for each development tool. This appendix describes file formats that are prepared as inputs and produced as outputs.

The appendix describes three types of files:

- [“Source Files” on page A-2](#)
- [“Build Files” on page A-5](#)
- [“Debugger Files” on page A-15](#)

Most of the development tools use industry-standard file formats. These formats are described in [“Format References” on page A-16](#).

Source Files

This section describes the following input file formats:

- “C/C++ Source Files” on page A-2
- “Assembly Source Files” on page A-3
- “Assembly Initialization Data Files” on page A-3
- “Header Files” on page A-4
- “Linker Description Files” on page A-4
- “Linker Command Line Files” on page A-5

C/C++ Source Files

C/C++ source files are text files (.C, .CPP, .CXX, and so) containing C/C++ code, compiler directives, possibly a mixture of assembly code and directives, and, typically, preprocessor commands.

Several dialects of C code are supported: pure (portable) ANSI C, and at least two subtypes¹ of ANSI C with ADI extensions. These extensions include memory type designations for certain data objects, and segment directives used by the linker to structure and place executable files.

The C/C++ compiler, run time library, as well as a definition of ADI extensions to ANSI C are detailed in the *VisualDSP++ 3.5 C/C++ Compiler and Library Manual* for the target processor.

¹ With and without built-in function support; a minimal differentiator. There are others.

Assembly Source Files

Assembly source files (.ASM) are text files containing assembly instructions, assembler directives, and (optionally) preprocessor commands. For information on assembly instructions, see the *Programming Reference* for your processor.

The processor's instruction set is supplemented with assembly directives. Preprocessor commands control macro processing and conditional assembly or compilation.

For information on the assembler and preprocessor, see the *VisualDSP++ 3.5 Assembler and Preprocessor Manual*.

Assembly Initialization Data Files

Assembly initialization data files (.DAT) are text files that contain fixed or floating point data. These files provide initialization data for an assembler .VAR directive or serve in other tool operations.

When a .VAR directive uses a .DAT file for data initialization, the assembler reads the data file and initializes the buffer in the output object file (.DOJ). Data files have one data value per line and may have any number of lines.

The .DAT extension is explanatory or mnemonic. A directive to `#include <file>` can take any file name and extension as an argument.

Fixed point values (integers) in data files may be signed, and they may be decimal, hexadecimal, octal, or binary base values. The assembler uses the prefix conventions listed in [Table A-1](#) to distinguish between numeric formats.

For all numeric bases, the assembler uses 16-bit words for data storage; 24-bit data is for the program code only. The largest word in the buffer determines the size for all words in the buffer. If there is some 8-bit data

Source Files

in a 16-bit wide buffer, the assembler loads the equivalent 8-bit value into the most significant 8 bits into the 8-bit memory location and zero-fills the lower eight bits.

Table A-1. Numeric Formats

Convention	Description
<code>0xnumber</code> <code>H#number</code> <code>h#number</code>	Hexadecimal number
<code>number</code> <code>D#number</code> <code>d#number</code>	Decimal number
<code>B#number</code> <code>b#number</code>	Binary number
<code>O#number</code> <code>o#number</code>	Octal number

Header Files

Header files (`.H`) are ASCII text files that contain macros or other preprocessor commands which the preprocessor substitutes into source files. For information on macros and other preprocessor commands, see the *VisualDSP++ 3.5 Assembler and Preprocessor Manual*.

Linker Description Files

Linker Description Files `.LDF` are ASCII text files that contain commands for the linker in the linker scripting language. For information on this scripting language, see the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Linker Command Line Files

Linker command line files (.TXT) are ASCII text files that contain command line input for the linker. For more information on the linker command line, see the *VisualDSP++ 3.5 Linker and Utilities Manual for 32-Bit Processors*.

Build Files

Build files are produced by the VisualDSP++ development tools while building a project. This section describes the following build file formats:

- “Assembler Object Files” on page A-6
- “Library Files” on page A-6
- “Linker Output Files” on page A-6
- “Memory Map Files” on page A-7
- “Loader Output Files in Intel Hex-32 Format” on page A-7
- “Loader Output Files in Include Format” on page A-10
- “Loader Output Files in Binary Format” on page A-10
- “Splitter Output Files in Motorola S-Record Format” on page A-11
- “Splitter Output Files in Intel Hex-32 Format” on page A-13
- “Splitter Output Files in Byte Stacked Format” on page A-13

Assembler Object Files

Assembler output object files (`.DOJ`) are binary, executable and linkable files (ELF). Object files contain relocatable code and debugging information for a DSP program's memory segments. The linker processes object files into an executable file (`.DXE`). For information on the object file ELF format, see the [“Format References” on page A-16](#).

Library Files

Library files (`.DLB`), the output of the archiver, are binary, executable and linkable files (ELF). Library files (called archive files in previous software releases) contain one or more object files (archive elements).

The linker searches through library files for library members used by the code. For information on the ELF format used for executable files, refer to [“Format References” on page A-16](#).



The archiver automatically converts legacy input objects from COFF to ELF format.

Linker Output Files

The linker output files (`.DXE`, `.SM`, `.OVL`) are binary, executable and linkable files (ELF). The executable files contain program code and debugging information. The linker fully resolves addresses in executable files. For information on the ELF format used for executable files, see the TIS Committee texts cited in [“Format References” on page A-16](#).

The loaders/splitters are used to convert executable files into boot-loadable or non-bootable files.

Executable files are converted into a boot-loadable file (`.LDR`) for the ADI processors using a loader program. Once an application program is fully debugged, it is ready to be converted into a boot-loadable file. A

boot-loadable file is transported into and run from a processor's internal memory. This file is then programmed (burned) into an external memory device within the target system.

A splitter generates non-bootable, PROM image files by processing executable files and producing an output PROM file. A non-bootable PROM image file executes from DSP external memory.

Memory Map Files

The linker can output memory map files (.XML), which are ASCII text files that contain memory and symbol information for the executable file(s). The .XML file contains a summary of memory defined with MEMORY{ } commands in the .LDF file, and provides a list of the absolute addresses of all symbols.

Loader Output Files in Intel Hex-32 Format

The loader can output Intel hex-32 format files (.LDR). The files support 8-bit wide PROMs and are used with an industry-standard PROM programmer to program memory devices. One file contains data for the whole series of memory chips to be programmed.

This example shows how the Intel hex-32 format appears in the loader output file. Each line in the Intel hex-32 file contains an extended linear address record, a data record, or the end-of-file record.

:020000040000FA	Extended linear address record
:0402100000FE03F0F9	Data record
:00000001FF	End-of-file record

Extended linear address records are used because data records have a 4-character (16-bit) address field, but in many cases, the required PROM size is greater than or equal to 0xFFFF bytes. Extended linear address records specify bits 31–16 for the data records that follow.

Table A-2 shows an example of an extended linear address record.

Table A-2. Extended Linear Address Record Example

Field	Purpose
:020000040000FA	Example record
:	Start character
02	Byte count (always 02)
0000	Address (always 0000)
04	Record type
0000	Offset address
FA	Checksum

Table A-3 shows the organization of an example data record.

Table A-3. Data Record Example

Field	Purpose
:0402100000FE03F0F9	Example record
:	Start character
04	Byte count of this record
0210	Address
00	Record type
00	First data byte
F0	Last data byte
F9	Checksum

Table A-4 shows an end-of-file record.

Table A-4. End-of-File Record Example

Field	Purpose
:00000001FF	End-of-file record
:	Start character
00	Byte count (zero for this record)
0000	Address of first byte
01	Record type
FF	Checksum

Utility Tool `hexutil`

The utility tool `hexutil` converts Intel hex-32 to Motorola S format, or produces an unformatted data file. This example shows how to include this file in a program.

```
%hexutil input_file <switches>
```

where <switches> are:

```
-s1|s2|s3|StripHex
```

that specifies the output format (s3 is the default and StripHex generates unformatted data), and

```
-o
```

that specifies the output file name in the form <input_root>.s.

Loader Output Files in Include Format

The loader can output `include` format files (`.LDR`). These files permit the inclusion of the loader file in a C program.

Files in `include` format are ASCII text files that consist of 48-bit instructions, one per line. Each instruction is presented as three 16-bit hexadecimal numbers. For each 48-bit instruction, the data order is lower, middle, and then upper 16 bits. Example lines from an `include` format file are:

```
0x005c, 0x0620, 0x0620,  
0x0045, 0x1103, 0x1103,  
0x00c2, 0x06be, 0x06be
```

This example shows how to include this file in a C program.

```
const unsigned loader_file[] =  
{  
    #include "foo.ldr"  
};  
  
const unsigned loader_file_count = sizeof loader_file  
                                   / sizeof loader_file;
```

`loader_file_count` reflects the actual number of elements in the array and cannot be used to process the data.

Loader Output Files in Binary Format

The loader can output binary format files (`.LDR`) to support a variety of PROM and microcontroller storage applications.

Binary format files use less space than the other loader file formats. Binary files have the same contents as the corresponding ASCII file, but in binary format.

Splitter Output Files in Motorola S-Record Format

The splitter can output Motorola S-record format files (.S_#), which conform to the Intel standard. The three file formats supported by the PROM splitter differ only in the width of the address field: S1 (16 bits), S2 (24 bits), or S3 (32 bits).

An S-record file begins with a header record and ends with a termination record. Between these two records are data records, one per line.

S00600004844521B	Header record
S10D00043C4034343426142226084C	Data record (S1)
S903000DEF	Termination record (S1)

[Table A-5](#) shows the organization of an example header record.

Table A-5. Example – Header Record

Field	Purpose
S00600004844521B	Example record
S0	Start character
06	Byte count of this record
0000	Address of first data byte
484452	Identifies records that follow
1B	Checksum

[Table A-6](#) shows the organization of an S1 data record.

The S2 data record has the same format, except that the start character is S2 and the address field is six characters wide. The S3 data record is the same as the S1 data record except that the start character is S3 and the address field is eight characters wide.

Table A-6. Example – S1 Data Record

Field	Purpose
S10D00043C4034343426142226084C	Example record
S1	Record type
0D	Byte count of this record
0004	Address of the first data byte
3C	First data byte
08	Last data byte
4C	Checksum

Termination records have an address field that is 16-, 24-, or 32 bits wide, whichever matches the format of the preceding records. [Table A-7](#) shows the organization of an S1 termination record.

Table A-7. Example – S1 Termination Record

Field	Purpose
S903000DEF	Example record
S9	Start character
03	Byte count of this record
000D	Address
EF	Checksum

The S2 termination record has the same format, except that the start character is S8 and the address field is six characters wide.

The S3 termination record is the same as the S1 format, except the start character is S7 and the address field is eight characters wide.

For more information, see “Utility Tool hexutil” on page A-9.

Splitter Output Files in Intel Hex-32 Format

The splitter can output Intel hex-32 format (.H_#) files. These ASCII files support a variety of PROM devices. For an example of how the Intel hex-32 format appears for an 8-bit wide PROM, see [“Source Files” on page A-2](#).

The splitter prepares a set of PROM files. Each PROM holds a portion of each instruction or data. This configuration differs from the loader output.

Splitter Output Files in Byte Stacked Format

The splitter can output files in byte stacked (.STK) format. These files are not intended for PROMs, but are ideal for microcontroller data transfers.

A file in byte stacked format comprises a series of one line headers, each followed by a block (one or more lines) of data. The last line in the file is a header that signals the end of the file.

Lines consist of ASCII text that represents hexadecimal digits. Two characters represent one byte. For example, F3 represents a byte whose decimal value is 243.

[Table A-8](#) shows an example of a header record in byte stacked format.

Table A-8. Example – Header Record in Byte Stacked Format

Field	Purpose
200080000000000080000001E	Example record
20	Width of address and length fields (in bits)
00	Reserved
80	PROM splitter flags (90 = PM, 00 = DM)
00	User defined flags (loaded with -u switch)

Table A-8. Example – Header Record in Byte Stacked Format

Field	Purpose
00000008	Start address of data block
0000001E	Number of bytes that follow

In the above example, the start address and block length fields are 32 bits wide. The file contains program memory data (the MSB is the only flag currently used in the PROM splitter flags field). No user flags are set. The address of the first location in the block is 0x08. The block contains 30 (1E) bytes (5 program memory code words). The number of bytes that follow (until next header record or termination record) must be nonzero.

A block of data records follows its header record, five bytes per line for data memory, and six byte per line for program memory. For example:

Program Memory Segment (Code or Data)

```
3C4034343426
142226083C15
```

Data Memory Segment

```
3C40343434
2614222608
```

The bytes are ordered left to right, most significant to least.

The termination record has the same format as the header record, except for the rightmost field (number of records), which is all zeros.

Debugger Files

Debugger files provide input to the debugger to define support for simulation or emulation of your program. The debugger supports all the executable file types produced by the linker (.DXX, .SM, .OVL). To simulate I/O, the debugger also supports the assembler data file format (.DAT) and the loader loadable file formats (.LDR).

The standard hexadecimal format for a SPORT data file is one integer value per line. Hexadecimal numbers do not require an 0x prefix. A value can have any number of digits but is read into the SPORT register as follows:

- The hexadecimal number is converted to binary.
- The number of binary bits read in matches the word size set for the SPORT register, which starts reading from the LSB. The SPORT register then fills with zero values shorter than the word size or conversely truncates bits beyond the word size on the MSB end.

In this example ([Table A-9](#)), a SPORT register is set for 20-bit words and the data file contains hexadecimal numbers. The simulator converts the hex numbers to binary and then fills/truncates to match the SPORT word size. The A5A5 is filled and 123456 is truncated.

Table A-9. SPORT Data File Example

Hex Number	Binary Number	Truncated/Filled
A5A5A	1010 0101 1010 0101 1010	1010 0101 1010 0101 1010
FFFF1	1111 1111 1111 1111 0001	1111 1111 1111 1111 0001
A5A5	1010 0101 1010 0101	0000 1010 0101 1010 0101
5A5A5	0101 1010 0101 1010 0101	0101 1010 0101 1010 0101
11111	0001 0001 0001 0001 0001	0001 0001 0001 0001 0001
123456	0001 0010 0011 0100 0101 0110	0010 0011 0100 0101 0110

Format References

The following texts define industry-standard file formats supported by VisualDSP++.

- Gircys, G.R. (1988) *Understanding and Using COFF* by O'Reilly & Associates, Newton, MA
- (1993) *Executable and Linkable Format (ELF) V1.1* from the Portable Formats Specification V1.1, Tools Interface Standards (TIS) Committee.

Go to: <http://developer.intel.com/vtune/tis.htm>.

- (1993) *Debugging Information Format (DWARF) V1.1* from the Portable Formats Specification V1.1, UNIX International, Inc.

Go to: <http://developer.intel.com/vtune/tis.htm>.

I INDEX

Symbols

- .ASM assembly files [A-3](#)
- .DAT data initialization files [A-3](#)
- .DLB library files [A-6](#)
- .DXE [2-3](#)
- .DXE executable files [A-6](#)
- .H_ files [A-13](#)
- .LDF (Linker Description Format) files [A-4](#)
- .ldf file [2-7](#)
- .LDR [2-3](#), [2-18](#)
- .LDR file [2-7](#)
- .LDR files [A-10](#)
 - boot kernels [2-7](#)
 - hex-format [A-7](#)
 - include-format [A-10](#)
 - specifying host bus width [4-31](#)
- .MAP memory map files [A-7](#)
- .OVL [2-15](#)
- .OVL overlay memory files [A-6](#)
- .S_ files [A-11](#)
- .SM [2-15](#), [2-17](#)
- .SM files
 - omitting [3-30](#)
- .SM shared memory files [A-6](#)
- .STK files [A-13](#)
- .TXT ASCII text files [A-5](#)

A

- Additional Options box [2-9](#)
- ADDR23-0 lines [4-15](#)
- ADSP-21065L control registers
 - DMAC8 [3-20](#)
- ADSP-2106x control registers
 - DMAC0 [3-20](#)
 - DMAC6 [3-23](#)
- ADSP-2106x/21160 control registers
 - DMAC6 [3-20](#)
 - DMAC8 [3-20](#)
- ADSP-21160 control registers
 - DMAC10 [3-20](#)
 - DMAC8 [3-23](#)
- ADSP-21161 control registers
 - DMAC10 [4-14](#)
 - DMAC8 [4-21](#), [4-23](#)
- ADSP-21161N loader
 - header words [4-9](#)
 - HostWidth # switch [4-31](#)
 - include output file [4-11](#)
 - include-format files [4-11](#)
- ADSP-218x loader/splitter [1-4](#)
- ADSP-TSxxx DSPs
 - boot modes [2-6](#)
- archive files [A-6](#)
 - see library files [A-6](#)

INDEX

- ASCII [2-17](#)
- assembler
 - source files (.ASM) [A-3](#)
- assembling [1-2](#)
- assembly initialization data files (.DAT)
 - [A-3](#)
- AUTODMA register [2-6](#)
- B**
- b switch [2-16](#), [2-17](#), [2-19](#)
- bboot [2-16](#)
- bhost [2-19](#)
- binary [2-17](#)
- binary-format files [A-10](#)
- blink [2-19](#)
- BMS pin [2-5](#)
- boot kernel [1-7](#), [2-5](#)
- Boot Kernel Files [2-7](#)
- boot kernel files [2-15](#)
- boot kernels [2-7](#)
 - modifying [3-11](#), [4-7](#)
 - rebuilding [3-11](#), [4-7](#)
- boot loading process [3-8](#)
- boot memory formats [4-9](#)
- boot mode pins [3-6](#), [4-5](#)
 - BMS [3-6](#), [4-5](#)
 - EBOOT [3-6](#), [4-5](#)
 - LBOOT [3-6](#), [4-5](#)
- Boot Mode Select (BMS) pin [2-6](#)
- boot modes [1-3](#), [1-5](#), [3-2](#), [4-2](#), [4-3](#)
 - boot sequence [3-5](#), [4-4](#)
 - EPROM [3-2](#), [3-15](#), [4-2](#), [4-3](#), [4-12](#), [4-14](#)
 - host [3-2](#), [3-19](#), [4-2](#), [4-3](#)
 - host (ADSP-21161) [4-16](#)
 - IIVT bit [3-14](#), [4-10](#)
 - kernel loading [3-5](#), [4-4](#)
 - link [3-2](#), [3-23](#), [4-2](#), [4-3](#)
 - link (ADSP-21161) [4-20](#)
 - no boot [4-23](#)
 - no-boot [3-3](#), [3-24](#), [4-3](#), [4-23](#)
 - reset vector address [3-6](#), [4-5](#)
 - SPI [4-22](#)
- boot ROM [1-7](#)
- boot scenarios [2-2](#)
- boot sequences [1-4](#), [1-7](#)
- boot type [2-3](#)
- Boot Types [2-5](#)
- boot types
 - ADSP-TSxxx DSPs [2-6](#)
 - working with [3-2](#), [4-2](#)
- Boot-file format [2-17](#)
- Booting [2-5](#), [2-6](#)
- booting [1-5](#), [2-5](#), [2-6](#)
 - about [3-3](#)
 - ADSP-TSxxx DSPs [2-3](#), [2-6](#)
 - DATA23-16 bits [4-13](#)
 - EPROM mode selection [3-15](#), [4-13](#)
 - host mode selection [3-19](#), [4-16](#), [4-17](#)
 - include-format files [4-11](#)
 - interrupt vector table [3-14](#), [4-10](#)
 - link mode selection [3-23](#), [4-20](#)
 - link port [3-23](#)
 - multiprocessor system [3-24](#)
 - NOP or IDLE instruction [4-13](#), [4-17](#)
 - SPI mode selection [4-22](#)
 - wake-up sequence [4-16](#), [4-19](#)
- boot-kernels

- changes and software issues [3-10](#), [4-6](#)
 - default [4-6](#)
 - features & boot-loading process [3-13](#), [4-6](#)
 - boot-loadable files [1-4](#), [2-1](#), [2-5](#), [2-16](#)
 - boot-modes
 - no-boot [4-10](#)
 - bootstraps [2-5](#)
 - bprom [2-19](#)
 - build
 - files, description of [A-5](#)
 - build options
 - splitter [5-2](#)
 - byte-stacked files [A-13](#)
 - used with splitter [5-10](#)
- C**
- C/C++
 - source files [A-2](#)
 - C/C++ compiler [2-8](#)
 - chip reset [2-5](#)
 - command line [2-9](#)
 - ADSP-21xxx loader [3-26](#), [4-26](#)
 - ADSP-TSxxx loader [2-12](#)
 - loader [3-26](#), [4-26](#)
 - splitter [5-8](#)
 - command line example [2-13](#)
 - Command-line help [2-18](#)
 - compiling [1-2](#)
 - customization of the boot kernel [2-8](#)
- D**
- data blocks
 - header words [4-9](#)
 - debugger
 - files [A-15](#)
 - default boot kernel file [2-19](#)
 - default boot-type format [2-17](#)
 - default option settings [2-9](#)
 - Determining Boot Mode [2-6](#)
 - development flow [1-1](#)
 - dm (include data memory) splitter
 - command-line switch [5-8](#)
 - dm (include Data Memory) splitter
 - switch [5-8](#)
 - DMA channel 10 (ADSP-21161) [4-13](#), [4-17](#)
 - DMAC0 control register
 - host booting [3-20](#)
 - DMAC10 control register [4-12](#)
 - EPROM booting [4-14](#)
 - host booting [3-20](#), [4-16](#), [4-18](#)
 - initialization [4-14](#), [4-18](#)
 - DMAC6 control register
 - EPROM booting [3-16](#)
 - link port booting [3-23](#)
 - DMAC8 control register
 - EPROM booting [3-16](#)
 - host booting [3-20](#)
 - link port booting [3-23](#), [4-21](#)
 - SPI port booting [4-22](#)
 - DWARF format
 - references [A-16](#)
- E**
- e filename [2-17](#)
 - EE-174 [2-2](#), [2-8](#)
 - EE-200 [2-2](#), [2-8](#)

INDEX

ELF file dumper
 references [A-16](#)
ELF standard [2-3](#)
elfloader.exe [2-3](#), [2-9](#), [3-26](#), [4-26](#)
emulator [1-2](#)
EPROM booting
 ADDR 31-0 [3-18](#)
 ADSP-21161N loader [4-12](#)
 BMS pin [3-19](#)
 boot-loading sequence [3-17](#)
 data packing [3-17](#)
 DMA settings [3-16](#), [4-12](#)
 external port data bus lines [3-18](#)
 mode selections [3-15](#), [4-12](#)
 multiprocessor systems [3-24](#), [4-24](#)
 program counter settings [3-17](#)
 single-processor systems [3-19](#)
 transfer settings [4-14](#)
 WAIT register [3-19](#)
EPROM/flash booting [2-5](#)
Executable and Linkable Format (ELF)
 [1-2](#)
Executable files [2-15](#)
executable files [1-4](#), [2-1](#), [2-3](#), [A-6](#)
EXORciser format files [5-10](#)
EXORMAX format files [5-10](#)
external memory [1-4](#), [2-20](#)
external pull-down resistor [2-6](#)
EZ-KIT Lite boards [1-3](#)

F
-f (PROM file format) splitter
 command-line switch [5-10](#)
-f switch [2-17](#)

-fASCII [2-17](#)
-fbinary [2-17](#)
-fformat [2-17](#)
-fhex [2-17](#)
file extension [2-14](#)
File Extensions [2-14](#)
file extensions
 ADSP-21161 loader [4-27](#), [4-29](#)
 ADSP-21xxx loader [3-28](#)
 ADSP-TSxxx loader [2-12](#)
 splitter [5-7](#)
file formats [1-3](#)
File Names [2-13](#)
file searches [1-9](#), [2-14](#)
files
 .ASM (assembly) [A-3](#)
 .H_ [A-13](#)
 .LDR (binary format) [A-10](#)
 .LDR (hex format) [A-7](#)
 .LDR (include-format) [A-10](#)
 .S_ [A-11](#)
 .STK [A-13](#)
 build [A-5](#)
 byte-stacked [A-13](#)
 C/C++ [A-2](#)
 data files [A-3](#)
 debugger [A-15](#)
 executable [A-6](#)
 EXORciser format [5-10](#)
 EXORMAX format [5-10](#)
 format references [A-16](#)
 formats [A-1](#)
 include format [A-10](#)
 input [A-2](#)

- Intel hex-32 format [A-13](#)
- library [A-6](#)
- library files [A-6](#)
- loader include-format (.LDR) [A-10](#)
- memory map [A-7](#)
- Motorola S-record format [A-11](#)
- overlay memory (OVL) [A-6](#)
- shared memory (SM) [A-6](#)
- text files [A-5](#)
- Flash Programmer plug-in [1-3](#)
- fS1 [2-17](#)
- fS2 [2-17](#)
- fS3 [2-17](#)

- G**
- generating
 - PROM files [5-1](#)

- H**
- h [2-18](#)
- h switch [2-18](#)
- header words
 - data blocks [4-9](#)
- help for TigerSHARC DSPs [2-18](#)
- hex format [2-17](#)
- hex-format files
 - .H_ [A-13](#)
 - .LDR [A-7](#)
- hexutil utility [A-9](#)
- HOST
 - boot type [2-16](#)
- host booting [2-6](#), [2-17](#), [3-19](#), [4-16](#)
 - BMS pin [3-19](#), [4-16](#)
 - boot-loading sequence [3-21](#)
- BUSLCK bit [3-22](#)
- DMA interrupts [3-22](#)
- IMASK register [3-21](#), [3-22](#)
- IMDW register [3-22](#)
- NOP or IDLE instruction [3-22](#)
- program counter settings [3-20](#)
- transfer settings [4-18](#)
- host booting mode, overview [1-6](#)
- host bus width
 - specifying [4-31](#)
- host processors [1-4](#)

- I**
- id#exe loader switch [3-25](#)
 - processor ID [3-33](#), [4-29](#)
- id#exe=file [2-18](#)
- include [2-17](#)
- include format
 - booting [4-11](#)
- initialization type tags [4-9](#)
- Intel hex-32 (.H_) files [A-13](#)
- Intel Hex-32 format [2-17](#)
- interrupt vector table [3-14](#), [4-10](#)

- K**
- kernels [1-6](#), [2-7](#), [2-8](#)

- L**
- l kernelfile [2-19](#)
- l switch [2-16](#), [2-19](#)
- library files [A-6](#)
- LINK
 - boot type [2-16](#)
- link booting [2-17](#), [3-23](#), [4-20](#)

INDEX

- ADSP-21161N 4-20
- BMS pin 3-23, 4-20
- data packing 3-23
- DMA channel interrupts 3-23
- external clock signal 3-23
- IMASK register 3-23
- link buffers
 - buffer 0 4-20
 - buffer 4 3-23
- Link Port Booting 2-6
- linker
 - command-line files (.TXT) A-5
 - description file (LDF) see .LDF files
 - executable files A-6
 - memory map files (.MAP) A-7
 - settings 1-2
- linking 1-2
- Load page 2-9
- Load page of the Project Options dialog box 2-3
- loadable files 1-3
- loader
 - ADSP-TSxxx switches 2-3
 - boot kernel 1-7
 - guide 4-2
 - hex-format files A-7
 - setting options 2-9, 3-26, 4-26
- loader command line 2-3
- loader driver help 2-18
- loader kernel 2-3
- loader kernels
 - see "boot kernels" 4-3
- Loader options 2-3
- Loader output file 2-15

- loader output file name 2-19
- loader routines 4-3
- long file names 2-12, 2-14

M

- mem21k.exe 3-11
- modifying
 - boot kernel source files 4-7
 - boot kernels 3-11, 4-7
- Motorola S-record (.S_) files A-11
- Motorola S-records 2-17
- MP systems
 - processor IDs 4-29
- multiple DSPs from a single EPROM 2-18
- multiprocessor booting 3-24, 4-24
 - from EPROM 3-24, 4-25
 - from single EPROM 3-25, 4-25
- Multiprocessor ID 2-18
- multiprocessor systems 1-4
 - processor IDs 3-33, 4-29

N

- no boot mode 3-24
- No Kernel option 2-7
- no-boot mode 3-3, 4-3, 4-10, 4-23
 - overview 1-5
- NoKernel 2-18
- NoKernel command-line switch 2-7
- no-mem (disable memory initialization) compiler switch 3-11
- non-bootable files 1-4
- norom (no ROM in PROM) splitter
 - command-line switch 5-9

O

- o (output file) splitter command-line switch [5-8](#)
- o (output file) splitter switch [5-8](#)
- o filename [2-19](#)
- omitting from .LDR files [2-7](#)
- output file format [2-3](#)
- overlay memory [2-15](#)

P

- p address [2-20](#)
- p switch [2-20](#)
- pm (include program memory) splitter command-line switch [5-9](#)
- pm (include Program Memory) splitter switch [5-9](#)
- porting
 - from previous SHARCs [4-13](#), [4-17](#), [4-20](#)
- power-ups [1-5](#)
- proc [2-16](#)
- proc ADSP-TSxxx [2-12](#), [2-18](#)
- processor IDs [3-33](#), [4-29](#)
- processor-loadable files [1-5](#)
- program development flow [1-1](#)
- Project Options dialog box [2-9](#)
- Project page [2-9](#)
- PROM
 - boot type [2-16](#)
 - booting mode [1-6](#)
 - memory [1-3](#), [1-7](#)
- PROM files
 - generating [5-1](#)
- PROM image [2-5](#)

PROM start address [2-20](#)

R

- r (ROM widths) splitter command-line switch [5-9](#)
- ram (include RAM in PROM) splitter command-line switch [5-9](#)
- rebuilding
 - boot kernels [3-11](#), [4-7](#)
- references
 - file formats [A-16](#)
- reset vector address [3-17](#)

S

- s (include memory segment) splitter command-line switch [5-10](#)
- S1 format [2-17](#)
- S2 format [2-17](#)
- S3 format [2-17](#)
- seg_ldr [2-7](#)
- shared memory [2-15](#), [2-17](#)
- show version information [2-21](#)
- si -version [2-21](#), [3-31](#), [4-31](#), [5-10](#)
- simulator [1-2](#), [1-4](#)
- single-processor systems [2-18](#)
- slave processors [1-4](#)
- software resets [1-5](#)
- source files [1-2](#)
 - assembly instructions [A-3](#)
 - C/C++ [A-2](#)
 - fixed-point data [A-3](#)
- sourcefile [2-16](#)
- specifying
 - splitter options [5-2](#)

INDEX

SPI control register (SPICCTL) [4-22](#)
SPI port booting [4-22](#)
SPIRX register [4-3](#), [4-22](#), [4-23](#)
splitter
 byte-stacked (.STK) files [A-13](#)
 byte-stacked files [5-10](#)
 command-line parameters [5-8](#)
 command-line reference [5-4](#)
 command-line switches [5-5](#)
 command-line syntax [5-4](#)
 file extensions [5-7](#)
 guide [5-2](#)
 hex format files
 .H_ [A-13](#)
 Motorola S-record files [A-11](#)
 producing PROM files [5-6](#)
 reference [5-4](#)
 using [5-2](#)
splitting [1-4](#)
SPORT data files [A-15](#)
-switch [2-12](#)
SYSCON register [3-10](#), [3-14](#), [3-18](#),
 [3-20](#), [4-6](#), [4-10](#)

T

-t timeout [2-20](#)
Txxxx_host.asm [2-7](#)
Txxxx_link.asm [2-7](#)
Txxxx_prom.asm [2-7](#)
-type [2-16](#)

U

-u (user flags) splitter switch [5-10](#)
user interface [2-9](#)
utilities
 elfloader.exe [3-26](#), [4-26](#)
 loader [3-26](#), [4-26](#)
 mem21k [3-11](#)

V

-v [2-20](#)
-version [2-21](#)
version for the build [2-21](#)
VisualDSP++
 splitter [5-2](#)

W

word width
 setting for loader output file [4-31](#)