

6 ADSP-2192 LOADER

Overview

The loader converts executable files into boot-loadable files for an ADSP-2192 DSPs. From the loader command line (or the **Load** tab of the **Project Options** dialog box when working within the VisualDSP++ environment), you can set options for the type of boot-loadable file that the loader produces. This document contains the following information on the loader:

- [“ADSP-2192-12 Boot Loader Utility and Run-Time Loader” on page 6-2](#)

This section contains reference information on the loader commands and operations.

- [“Loader Glossary” on page 6-14](#)

This section contains a glossary of loader-related terms.

ADSP-2192-12 Boot Loader Utility and Run-Time Loader

The ADSP-2192-12 DSP can be boot-loaded through either the PCI or USB interface. For PCI loading, this requires that the loadable image reside in the PC host's memory space before it can be downloaded to the DSP. VisualDSP++ includes a boot loader utility (BLU) for repackaging a DXE (and associated OVL and SM files) for usage within a run-time boot loader (RTBL). The RTBL is implemented by the user, but VisualDSP++ does include a reference RTBL that allows users to boot-load to the ADSP-2192-12 EZ-Kit Lite on the Windows 98 and Windows 2000 platforms.

This section contains the following information:

- [“Using the Boot Loading Utility” on page 6-3](#)
- [“Running the Boot Loader Utility” on page 6-4](#)
- [“ADSP-2192 Boot Loader Utility Command-Line Reference” on page 6-5](#)
- [“Using the Run-Time Boot Loader” on page 6-10](#)
- [“Run-Time Boot Loader and Overlays Over PCI” on page 6-11](#)
- [“Using OvlPciAdrTbl and OvlMgrTbl Symbols” on page 6-12](#)

Using the Boot Loading Utility

The VisualDSP++ linker produces files with file extensions DXE, OVL, and SM. These files are of the ELF format and are usually used by a debugger. Typically, these files are not shipped with a user's final end product. Instead, the linker's output is run through a boot loader utility (BLU) that repackages the output for use by some bootable device, the PCI or USB interface in the case of the ADSP-2192-12 DSP.

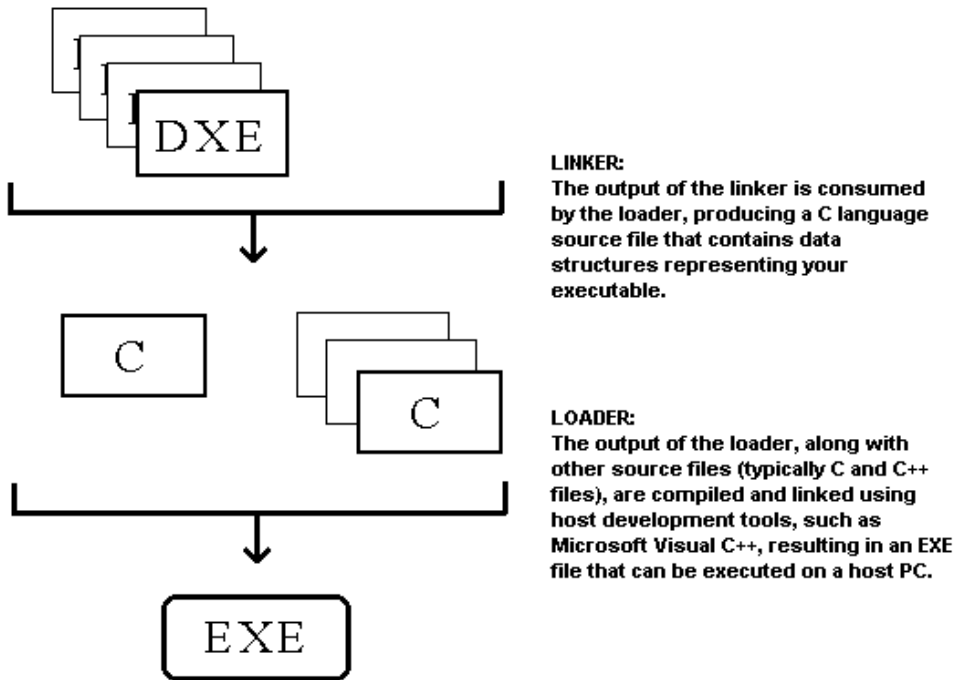
Given that booting from PCI or USB involves code running on the PC host, you need to create a program that executes on the host to initiate and conduct the actual PCI or USB transfer. Because of this, the output of the BLU is the C language source code for inclusion and compilation into a host program using a Windows compiler (like Microsoft Visual C++) to create the RTBL (see [Figure 6-1](#)).

The source code emitted by the BLU is essentially a linked list of arrays representing the executable's sections and their contents. Refer to the comments contained within the generated file for specific information about the data structures and their usage.



BLU must be rerun and the RTBL rebuilt whenever the DSP executable image changes. A convenient way to automate this in the VisualDSP++ environment is to select a project output type of "loader file" and then create a post-build command that invokes a Makefile to build the RTBL.

Figure 6-1. Boot Loader Utility Usage



Running the Boot Loader Utility

There are two ways to run the boot loader utility, one is through the VisualDSP++ environment, and the other is through a command line. The boot loader utility in either case generates a single output file with the extension of `.h`.

When developing a DSP project, you may find it useful to modify the loader's default option settings in the VisualDSP++ environment.

Loader options let you control how the loader processes your executable files. These options (switches) appear on the loader command line. These

switches correspond to the **Load** dialog's options. Use the **Additional Options** field (from the **Load** tab of the **Project Options** dialog box) to enter the appropriate file names and command-line switches to support the loader operation.

For more information on IDDE, see the *VisualDSP++ 2.0 User's Guide for ADSP-21xx DSPs*.

ADSP-2192 Boot Loader Utility Command-Line Reference

The boot loader utility (`elfloader.exe`) generates a loader file for ADSP-2192-12 DSPs by processing executable, overlay and shared memory files. The boot loader utility's output file is a C-language header file with the file name extension of `.h`. To produce a C-language header output file, the boot loader utility processes data from one or two executables (`.DxE`) along with a number of overlay (`.OVL`) and shared memory (`.SM`) files generated by the linker.

Before running the boot loader utility, make sure that the overlay (`.OVL`) and shared memory (`.SM`) files are all in the same working directory as the executables. The boot loader utility will automatically open those overlay (`.OVL`) and shared memory (`.SM`) files and process the data from them while the boot loader utility is processing the executables (`.DxE`).

This section provides reference information on the boot loader utility command line. When you run the boot loader utility, you can put the switched in any order. A list of all switches and a description for each switch appears in [Table 6-2 on page 6-8](#).

Use the following syntax for the boot loader utility command line if you have only one input executable (`.DxE`):

```
elfloader -core0 sourcefile -dADSP-2192 -switch [-switch...]
```

or

ADSP-2192-12 Boot Loader Utility and Run-Time Loader

```
elfloader -core1 sourcefile -ADSP-2192 -switch [-switch...]
```

where:

- `-core0` -- tells the boot loader utility that the sourcefile is for core 0. The boot loader utility makes up the array and structure names according to this core number supplied in the output file.
- `-core1` -- tells the boot loader utility that the sourcefile is for core 1. The boot loader utility makes up the array and structure names according to this core number supplied in the output file.
- *sourcefile* -- the name of the input executable file (.DxE) to be processed for a single-processor. A file name can include the drive, directory, file name and file name extension; enclose long file name within straight-quotes "long file name".
- `-dADSP-2192` -- a mandatory switch which tells the boot loader utility to produce an output file for ADSP-2192 processor. In the default, the output file is a C-language header file.
- *-switches* -- other option(s) to be passed to the loader.

Use the following syntax for the boot loader utility command line if you have two input executables (.DxE):

```
elfloader -core0 sourcefile0 core1 sourcefile1 -dADSP-2192  
-switch [-switch...]
```

where:

- `-core0 sourcefile0` -- tells the boot loader utility that the *sourcefile0* is the input file to be processed for the core 0. The boot loader utility makes up the array and structure names according to the supplied core number.
- `-core1 sourcefile1` -- tells the boot loader utility that the *sourcefile1* is the input file to be processed for the core 1. The boot loader utility makes up the array and structure names according to the supplied core number.

- `-dADSP-2192` -- a mandatory switch which tells the boot loader utility to produce an output file for ADSP-2192 processor. In the default, the output file is a C-language header file.
- `-switches` -- other option(s) to be passed to the loader.

Example:

The following command line, for example:

```
elfloader -dADSP-2192 -core0 p0.dxe -core1 p1.dxe
```

runs the boot loader utility with

`-dADSP-2192` -- Tells the boot loader utility to produce an output file for a ADSP-2192 processor.

`-core0 p0.dxe` -- Provides the input file `p0.dxe` to be processed for the core 0.

`-core1 p1.dxe` -- Provides the input file `p1.dxe` to be processed for the core 1.

Since no output file is specified, the default output file, in this case, is `p0.h`.

File searches are important in the boot loader utility process. The boot loader utility supports relative and absolute directory names, default directories, and user-selected directories for search paths.

File searches occur as follows:

1. *Specified path*—If you include relative or absolute path information in a file name, the loader only searches in that location for the file.
2. *Default directory*—If you do not include path information in the file name, the loader searches for the file in the current working directory.

ADSP-2192-12 Boot Loader Utility and Run-Time Loader

When you 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 name extension to each file.

The boot loader utility follows the conventions for file name extensions that appear in [Table 6-1](#). Descriptions for each switch appear in [Table 6-2](#).

Table 6-1. File Name Extension Conventions

Extension	File Description
.dxe ¹	Executable files and boot-kernel files
.h	Loader output file for a C header file

¹ The loader recognizes shared memory (.SM) and overlay memory (.OVL) files, but does not expect these files on the command line. Place .SM or .OVL files in the same directory as the .DXE that refers to them, and the boot loader utility finds them when processing the .DXE file.

The switches in [Table 6-2](#) may be used in any order on the command line.

Table 6-2. ADSP-2192 Boot Loader Utility Command-Line Switches¹

Switch	Description
-core0 <i>file</i>	The input executable file filename is processed for the core 0.
-core1 <i>file</i>	The input executable file filename is processed for the core 0
-dADSP2192	The switch -dADSP-2192 directs the boot loader utility to produce an output file for ADSP-2192-12 processors.
[-f <i>format</i>]	The -f (boot file format) switch directs the loader to prepare a output file in the specified format. Currently, the boot loader utility supports the C style header file only, and it is a default.

Table 6-2. ADSP-2192 Boot Loader Utility Command-Line Switches¹

Switch	Description
-h	The -h (command line help) switch directs the loader to output the list of command line switches to standard output and exit.
-o <i>file</i>	This switch directs the loader to use the <i>filename</i> parameter to write to an output file.
-v	The -v switch directs the boot loader utility to show process information while it processes the input files.
-M	The -M switch directs the boot loader utility to show the dependencies only
-MM	The -MM switch directs the boot loader utility to show the dependencies while processing the input files and producing an output file.
-Mo <i>filename</i>	The -Mo switch directs the boot loader utility to put the dependency information to the output file specified in name filename. -Mo switch must be used in conjunction with -M or -MM.
Mt <i>TargetName</i>	The -Mt switch directs the boot loader utility to use the <i>targetname</i> in the dependency file. -Mo switch must be used in conjunction with -M and -MM.

¹ The switches may be used in any order on the command line. Items shown in [] are optional. Items shown in *italics* are user defined and are described with each switch.

Using the Run-Time Boot Loader

As discussed above, a host compiler is used in conjunction with the BLU's output and other user-written code to create a host executable, known as the run-time boot loader (RTBL). The RTBL, executed on the end-user system, traverses the data structures created by the BLU (and subsequently compiled into the RTBL). The content of these data structures are downloaded to the ADSP-2192-12 DSP through the use of a Windows driver, also provided by the user.

Note that a program running in virtual memory space (as all Windows applications do) cannot access the PCI-mapped memory of the ADSP-2192-12 DSP directly; therefore, use of a driver is mandatory.

To create a RTBL file:

1. Generate the DXE, OVL, and SM files.
2. Run the boot loader utility with the DXE, OVL, and SM files as inputs.
3. The boot loader utility will generate a C header file, `rtbl.h`, containing all the necessary information extracted from these files.
4. Using the generated header file, (`rtbl.h`), and a user-created C source file, build a Windows executable (`.EXE`) file referred to as the run-time boot loader (RTBL). The RTBL will use the data from the generated header file and make calls to a Windows driver in order to communicate with both cores.
5. Execute the RTBL in order to load and run the application.

Creation of the RTBL and driver can be a complex task, especially the first time a programmer undertakes such a task. To assist you in this endeavor, VisualDSP++ includes a reference RTBL in the form of a Microsoft Visual C++ 6.0 project named `reference_rtbl.dsp`. This project can be found in the `...\VisualDSP++\219x\ldr` subdirectory. Refer to the files con-

tained in this project for more information on the operation of the RTBL and the *driver interface provided.

The RTBL downloads the code via the PCI bus to the ADSP-2192-12 EZ-Kit Lite through the use of the EZ-Kit's PCI driver. The reference can be used as an example for how to traverse and use the data structures emitted by the BLU, and then download those structures to the target through a driver.



The reference RTBL does not support USB booting.



The EZ-Kit driver can likely be reused for targets other than Analog Devices' EZ-Kit, but this driver is extremely simple and will likely be inadequate for anything other than prototyping. Furthermore, the EZ-Kit driver can only be reused on so-called "plug 'n' play" Windows operating systems like Windows 98 and Windows 2000.

Run-Time Boot Loader and Overlays Over PCI

The usage of overlays in an executable can greatly complicate the use of the RTBL and PCI driver for two main reasons:

- The overlay's "live space" is on the PC host and not the DSP. As a result, the linker cannot not know these addresses at build time, so the PCI driver must patch executables as they are downloaded to the DSP to insert the correct addresses at run-time.
- The DSP cannot access the Windows' virtual memory space. An overlay must be copied from virtual memory space to PCI memory space, which can only be allocated in limited quantities by the PCI driver.

However, the output of the BLU considers the need for run-time patches of the executable image. A portion of the data structure created by the BLU is for the expressed purpose of enabling the user code in the RTBL to

ADSP-2192-12 Boot Loader Utility and Run-Time Loader

fix up these addresses at run-time. Refer to the reference RTBL for an example of how to do this.

Due to the DSP's inability to access Windows' virtual memory space, the RTBL and driver must be coordinated to make overlays available in PCI memory space.

Typically, the overlay's image and size information is sent down to the driver. The driver then `malloc()`'s a memory buffer equal in size to the overlay and then copies the overlay to this space, thus making the overlay available in the PCI memory space.

Refer to the reference RTBL for an example of how to do this.

Using Ov1PciAdrTbl and Ov1MgrTbl Symbols

The boot loader utility, when running, looks for the symbols `Ov1PciAdrTbl` and `Ov1MgrTbl`. If the loader could not resolve these two symbols, it will set the values of both to be zero. Users need to declare one or both symbols in the assembly source file in order to make the run-time boot loader handle the overlay properly.

The `Ov1PciAdrTbl` symbol holds the start address of the overlay live address table. The loader gets the value of this symbol from the input `.DXE` file and places it in the "offset" of the first `relocation_type` array. The value of the "offset" of the second `relocation_type` array is then the value of the first `relocation_type` "offset" plus 2.

Consequently, each next "offset" gets a value which is the previous "offset" plus 2. The run-time boot loader will determine the exact the overlay live address of each overlay according to the "offset" value provided.

Instead of defining the `Ov1PciAdrTbl` symbol, you can choose to define the `Ov1MgrTbl` symbol in the assembly source code. This symbol should hold the start address of the overlay table, and the overlay table can be declared and defined in the user's assembly source code. The boot loader

will get the value of this symbol and make `#define` statement along with the other `#define` statement. In a simple case, they look like:

```
#define CORE0_OVL_MGR_TBL  0
#define CORE0_OVL_COUNT    3
```

The first `#define` statement gives the start address of the overlay table, and it will be a zero if the boot loader fails to find the symbol in the input `.dxe` file. You can either manually change the value or define it in the assembly source code and re-build the boot loader file to make the value correct.

The second `#define` statement gives the number of overlay for core 0. The run-time boot loader will later use the overlay table start address, provided to find the entry for the live start address, for each overlay and change it before loading the overlay table to the DSP memory.

Loader Glossary

Boot-kernel—The boot-kernel refers to the executable file that performs the memory initialization on the target.

Boot Loader Utility—See Loader.

Boot-loadable file—The boot file refers to the output of the loader that contains the boot loader and the formatted system configurations.

Boot-loading or booting—Booting refers to the process of loading the boot loader, initializing system memory, and starting the application on the target.

Loader—The loader refers to `elfloader` contained in the software release.

Run-Time Boot Loader—Host executable responsible for downloading the code for the DSP.