Technical notes on using Analog Devices DSPs, processors and development tools
Contact our technical support at dsp.support@analog.com and at dsptools.support@analog.com
Or visit our on-line resources http://www.analog.com/ee-notes and http://www.analog.com/processors

A Boot Compression/Decompression Algorithm for Blackfin® Processors

Contributed by Bob Nakib

Rev 1 – December 9, 2004

Introduction

This application note describes a boot compression/decompression algorithm for ADSP-BF533 and ADSP-BF561 Blackfin® processors.

The code included with this document was verified using the VisualDSP++® 3.5 tools suite for Blackfin processors. The project was run on an EZ-KIT Lite® evaluation system for ADSP-BF533 Blackfin processors (ADDS-BF533-EZLITE, Rev 1.2) and on an EZ-KIT Lite evaluation system for ADSP-BF561 Blackfin processors (ADDS-BF561-EZLITE, Rev 1.1).

Purpose

Normally, loader files created by VisualDSP++ are not compressed, except possibly with a simple run-length compression scheme used on strings of 0s in the compiled code. For large projects that must be stored in non-volatile memory, compression provides significant savings in hardware, such as on-board flash. Decompression code can then be executed at boot time by creating a *2nd-stage loader* or *initialization code* containing the decompression algorithm. For more information regarding the initialization code or any aspect of the booting process, refer to *ADSP-BF533 Blackfin Booting Process (EE-240)* [1].

Boot Compression/Decompression Algorithm

This compression algorithm requires about 50 KB of overhead (extra) memory. Therefore, it is useful only in applications of a significantly larger size. The compression/decompression functions are handled by a simple open-source compression library called Zlib. For more information, see

`http://www.gzip.org/zlib/`.

As an example, a compressed Blink application is included with this EE-Note for your convenience. It is suggested that you try the following steps using the included Blink application to familiarize yourself with the compression process, then try programming the last generated file (`CompressedAppIntelHex.ldr`) with the Flash Programmer to verify correct functionality. Figure 3 shows the entire process.

Step 1

You must allocate an unused area of SDRAM memory in your user-application to store the uncompressed user-application loader image. The `.LDF` file of your user-application project must be modified to free a section of SDRAM memory of appropriate size. This region must be large enough to hold an entire loader build of your user-application, allowing `uncompress()` to deflate to that area in memory.

The user-application loader image will then be loaded by the Boot ROM out of SDRAM.

For ADSP-BF561 processors, you must also free L2 memory addresses `0xFEB1 FC00` through `0xFEB1 FFE7` by keeping them out of any section in your user-application's `.LDF` file. These addresses are reserved for the 2nd-stage boot kernel. This is done in the `.LDF` file of the example ADSP-BF561 Blink program.

Step 2

Set the `UNCOMPR_ADDR` field at the top of the `Init_code.c` file (Init folder) to the intended destination address for the output of the uncompress function (that is, the starting address of the *reserved* section or kept empty in the SDRAM memory of your user-application in Step 1). In the given example, this address is set to `0x4000`.

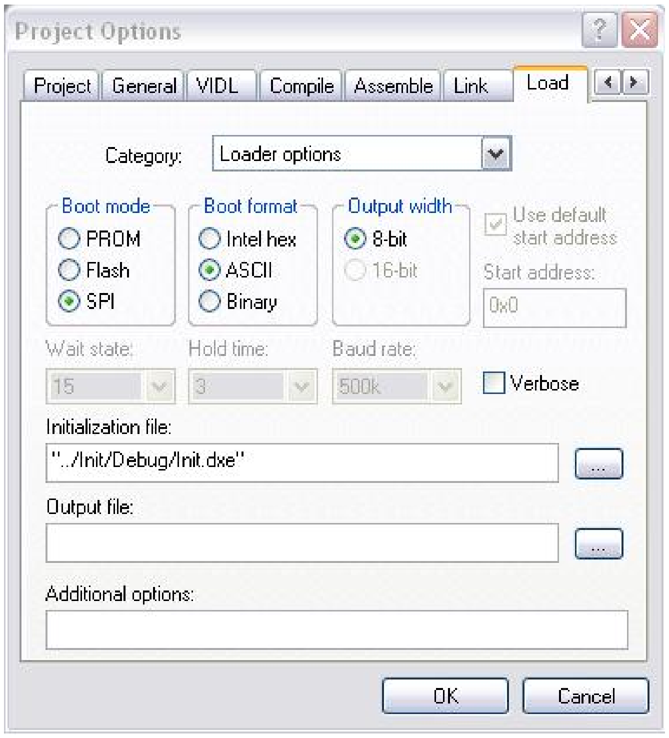
Step 3

Specify the user-application project to be built as a `Loader file` (via Load page of Project Options dialog box) and use `SPI/ASCII/8-bit` format. Note that the `SPI` setting is a workaround used to generate the output in ASCII format.

For ADSP-BF533 processors, specify the following path to the initialization project file: `../Init/Debug/Init.dxe`. (See Figure 1.)

For ADSP-BF561 processors, select `Boot kernel options` from the `Category` list, select `Use boot kernel`, and specify the path to the kernel file as: `../Init/Debug/p0.dxe`. Also, under `Additional options,` insert: `./debug/p0.dxe`.

When executed at the start of the boot process, this object file initializes SDRAM memory, finds the compressed user-application code, and decompresses it to the appropriate area in SDRAM memory.



Category: Loader options

Boot mode: ( ) PROM ( ) Flash [✓ SPI]

Boot format: ( ) Intel hex [✓ ASCII] ( ) Binary

Output width: [✓ 8-bit] ( ) 16-bit

[✓] Use default start address

Start address: 0x0

Wait state: 15

Hold time: 3

Baud rate: 500k

[ ] Verbose

Initialization file: "../Init/Debug/Init.dxe"

Output file:

Additional options:

Figure 1. Loader Options for ADSP-BF533 Processors

Step 4

Build the Init code project first, and then build your user-application. Ensure that the project to be built is active (highlighted in the Project window) and that project dependencies are set, if necessary. This should create an `.LDR` file in the Debug directory of your user-application.

Step 5

In the root folder, edit the `compress.bat` file (use any text editor) to specify the path of the target loader file that has just been built from your user-application, as well as the desired level of compression (see Table 1 in the Appendix for compression levels). Note that the entered path must be enclosed in double-quotes (`"`) and must use forward-slashes or double-backslashes. An incorrectly specified path name or compression level may cause undesirable results.

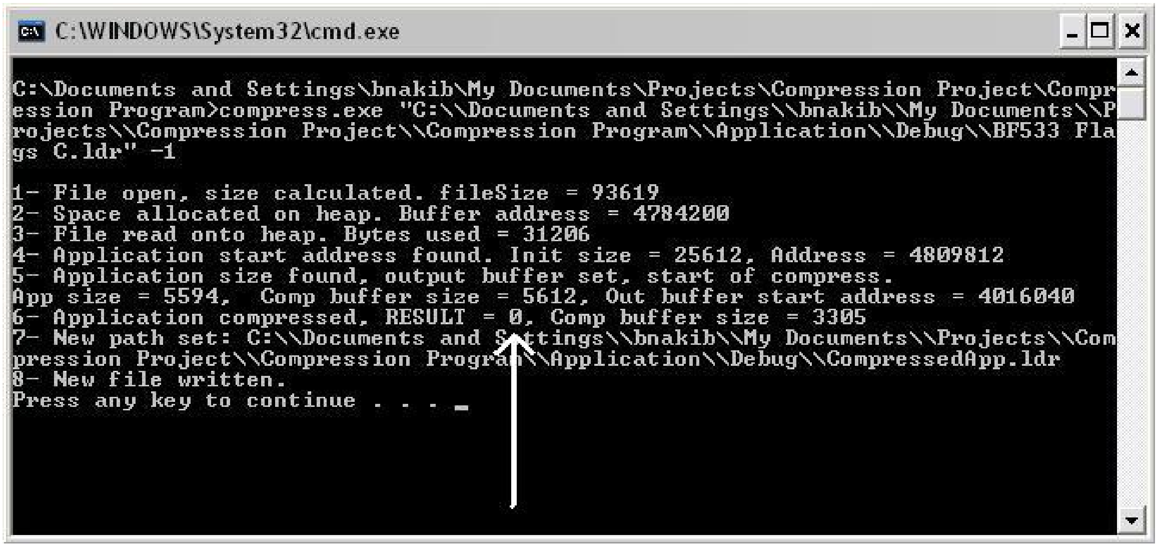
Step 6

Execute `compress.bat` (root folder) and ensure the result of the compression (returned on line 6 of the Output window) is successful by matching your return code with those listed in Table 2 in the Appendix.

The Output window (Figure 2) returns important values calculated during compression, notably:

- Source file size (line 1)
- Source buffer start address (line 2)
- Number of bytes used for the source buffer (line 3)
- Application start address and number of bytes used for the Init code buffer (line 4)
- Application size and output buffer start address (line 5)
- Compression result (0 = success) and compressed application size (line 6)
- Path of the output file (line 7)

If successful, a file called `CompressedApp.ldr` is created in the same directory as your source loader file (that is, the Debug directory of your user-application). This ASCII format file contains both the Init (decompression) code and your compressed user-application code.



```
C:\Documents and Settings\bnakib\My Documents\Projects\Compression Project\Compr
ession Program>compress.exe "C:\\Documents and Settings\\bnakib\\My Documents\\P
rojects\\Compression Project\\Compression Program\\Application\\Debug\\BF533 Fla
gs C.ldr" -1

1- File open, size calculated. fileSize = 93619
2- Space allocated on heap. Buffer address = 4784200
3- File read onto heap. Bytes used = 31206
4- Application start address found. Init size = 25612, Address = 4809812
5- Application size found, output buffer set, start of compress.
App size = 5594,  Comp buffer size = 5612, Out buffer start address = 4016040
6- Application compressed, RESULT = 0, Comp buffer size = 3305
7- New path set: C:\\Documents and Settings\\bnakib\\My Documents\\Projects\\Com
pression Project\\Compression Program\\Application\\Debug\\CompressedApp.ldr
8- New file written.
Press any key to continue . . . _
```

Figure 2. Compression Script Output Window, RESULT = 0 (SUCCESS)

Step 7

Copy the `CompressedApp.ldr` file to the root folder, where you should then execute `IntelHex.bat`. This converts the compressed application loader file into Intel hexadecimal format and creates a new file called `CompressedAppIntelHex.ldr`. This conversion is necessary for the loader file to be understood by the Flash Programmer.

Next, run the VisualDSP++ Flash Programmer plug-in, specifying `CompressedAppIntelHex.ldr` as the source file. This copies the compressed loader image into flash memory, starting at address `0x2000 0000`. Upon reset, the Boot ROM will begin booting from this address in flash memory, loading the initialization code into L1 SRAM memory. The initialization code will set up SDRAM and then uncompress the user-application image to the specified address in SDRAM memory. Booting then proceeds from the uncompressed user-application address. (see Figure 3 below).

Allocating Memory Sections in SDRAM

Listing 1 shows the default SDRAM sections allocated in `adsp-BF533_C.ldf` and the default linker file used for ADSP-BF533 Blackfin processor C projects. (This file can be found under the path:

```
…\Analog Devices\VisualDSP 3.5 16-Bit\
Blackfin\ldf
```

You can create your own sections and specify start and end addresses for any section. By doing so, you can control linker memory allocation partially. (For proper data alignment, all address blocks should be 32-bit addressable.)

However, if blocks of code or data are to be placed into any user-defined section of memory (by using the `.section` command, for example), edit the block entitled `SECTIONS`. An example appears in Listing 2.

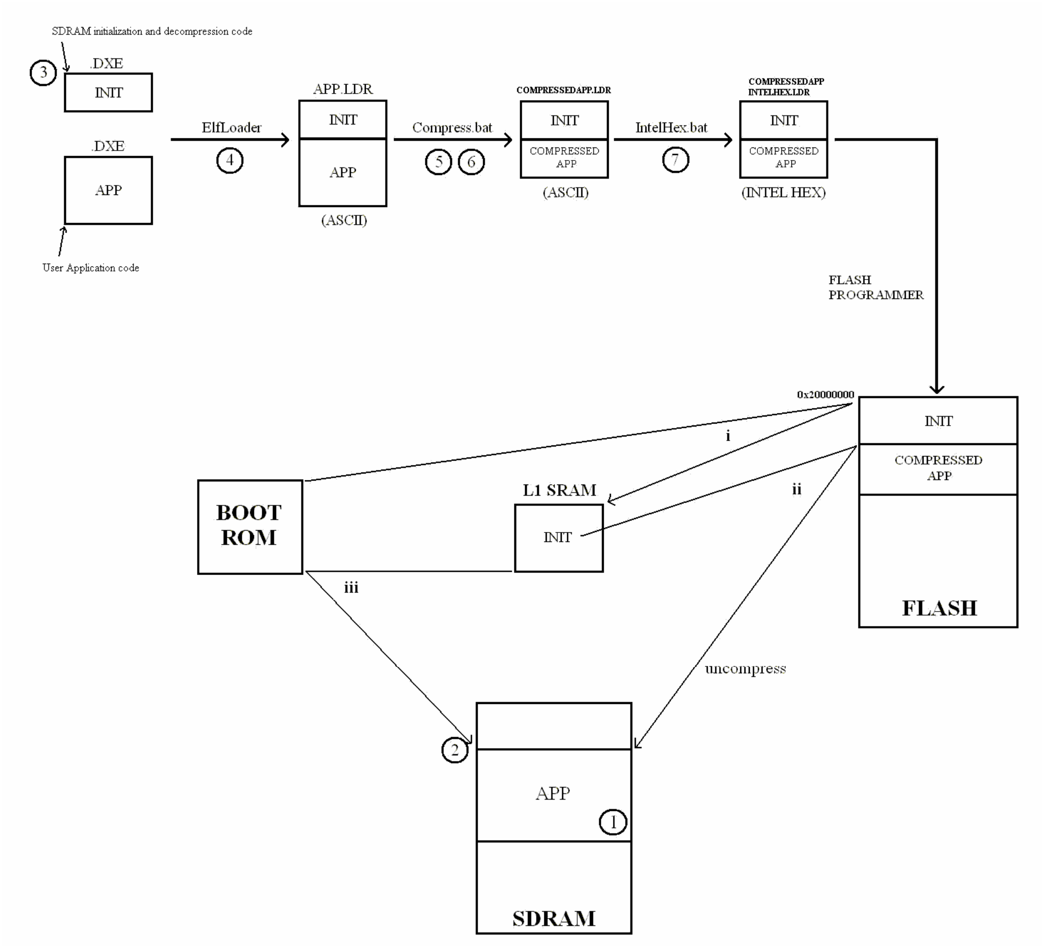
```
 (Line 285:)

MEM_SDRAM0     {
    TYPE(RAM) WIDTH(8)
    START(0x00004000) END(0x07FFFFFF)
}
MEM_SDRAM0_HEAP     {        TYPE(RAM)
WIDTH(8)
    START(0x00000004) END(0x00003FFF)
}
```

Listing 1. Default Section Allocation

```
 (Line 285:)

MEM_SDRAM0 {
    TYPE(RAM) WIDTH(8)
    START(0x00004000) END(0x06FFFFFF)
}

NEW_SDRAM_SECTION {
    TYPE(RAM) WIDTH(8)
    START(0x07000000) END(0x07FFFFFF)
}

MEM_SDRAM0_HEAP {
    TYPE(RAM) WIDTH(8)
    START(0x00000004) END(0x00003FFF)
}
… … … …

SECTIONS
{
  my_section
  {
    INPUT_SECTION_ALIGN(4)
    INPUT_SECTIONS(
      $OBJECTS(my_section)
      $LIBRARIES(my_section)
    )
  } >NEW_SDRAM_SECTION

  … … … …
```

Listing 2. User-Defined Section Allocation

For more information , refer to *Guide to Blackfin Processor LDF Files (EE-237)* [2].



1. Allocate an unused area of SDRAM memory in which to store your uncompressed user-application loader image.
2. Set the UNCOMPR_ADDR field at the top of the Init_code.c file (Init folder) to point to the start address of this reserved memory area.
3. Build the Init code project.
4. Build your user-application. This should create an .LDR file in the Debug directory of your user-application.
5. Edit compress.bat to specify the path of the target loader file that has just been created, as well as the desired level of compression.
6. Execute compress.bat.
7. Execute IntelHex.bat to convert your compressed application loader file into Intel hexadecimal format. This will create a new file called CompressedAppIntelHex.ldr.

Now run the VisualDSP++ Flash Programmer, and specify CompressedAppIntelHex.ldr as your source file. At boot time, this will initiate the following:

i – Boot ROM begins booting from the start of flash memory, at address 0x2000 0000. Init code is executed in L1 memory.

ii – Init code initializes SDRAM and decompresses the user-application code to the appropriate area in SDRAM memory.

iii – Init code completes execution, and booting proceeds from the uncompressed user-application address.

Figure 3. Boot Compress/Decompress Process

Optional User Modifications

You can edit certain files or parameters in the project folder manually. The following guidelines describe the files or parameters that may need modification.

Application Folder

Replace the sample Blink application project with your user-application project.

Init Project

If you are not using an ADSP-BF533 or ADSP-BF561 EZ-KIT Lite board, you may need to modify the default linker sections in the `.LDF` file of the Initialization project.

Init_code.c

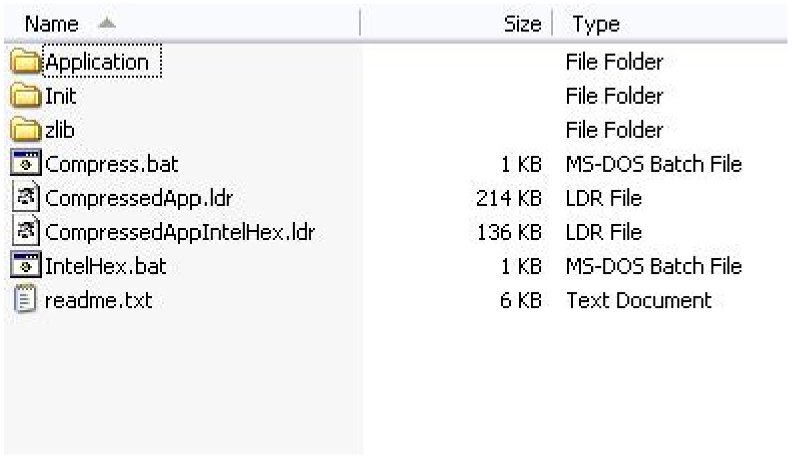
Set the `UNCOMPR_ADDR` field to the intended destination address for the output of the uncompress function.

Compress.bat

The source file path and compression level may be changed.

IntelHex.bat

The output file name (2nd parameter) may be changed if desired.



Name	Size	Type
Application		File Folder
Init		File Folder
zlib		File Folder
Compress.bat	1 KB	MS-DOS Batch File
CompressedApp.ldr	214 KB	LDR File
CompressedAppIntelHex.ldr	136 KB	LDR File
IntelHex.bat	1 KB	MS-DOS Batch File
readme.txt	6 KB	Text Document

Figure 4. Main Project Directory

Appendix. Zlib Compression Levels and Return Codes

Below are the different compression levels and return codes used by the Zlib library. These values are taken from the `zlib.h` file accompanying this EE-Note.

Compression Levels

Best Speed	1
Default Compression	(-1)
Best Compression	9

Table 1. Compression Levels

Return Codes

Return codes for the compression/uncompression functions. Negative values are errors, positive values are used for special but normal events.

Z_OK	0
Z_STREAM_END	1
Z_NEED_DICT	2
Z_ERRNO	(-1)
Z_STREAM_ERROR	(-2)
Z_DATA_ERROR	(-3)
Z_MEM_ERROR	(-4)
Z_BUF_ERROR	(-5)
Z_VERSION_ERROR	(-6)

Table 2. Return Codes

References

[1] *ADSP-BF533 Blackfin Booting Process (EE-240).* Rev 1. June 2004. Analog Devices, Inc.

[2] *Guide to Blackfin Processor LDF Files (EE-237).* Rev 1. May 2004. Analog Devices, Inc.

[3] *zlib Home Site (http://www.gzip.org/zlib/).* November 2003. Greg Roelofs and Jean-loup Gailly

Document History

Revision	**Description**
Rev 1 – December 9, 2004 by B. Nakib and T. Lukasiak	Initial Release