Using OFD Utility to Create a DSP Boot Image

Jelena Nikolic-Popovic *DSP Field Applications*

ABSTRACT

The object file display (OFD) utility processes common object file format (COFF) files and converts the information contained in such files into XML format. This application note describes how to create a DSP boot image using COFF and XML files, and a simple Perl script. The resulting image is a C source file, which can be included into the host processor's application program and downloaded to the DSP via HPI or PCI interfaces.

The OFD utility is available in Code Composer Studio Release 3.0 or greater.

This application report contains project code that can be downloaded from this link. http://www−s.ti.com/sc/psheets/spraa64/spraa64.zip

Contents

List of Figures

1 Motivation

In a signal processing system consisting of one or more DSPs and a host processor where the DSP code initially resides in host's memory space, the DSP code needs to be converted into a particular format; so that, it can be included into host's application program, and then copied to the DSP memory space upon system startup. Typically, the DSP image is included in the form of a C source file.

The traditional way to create an image is:

- to generate COFF (.out file) using TI's linker
- to convert COFF file into an ASCII image using TI's HEX utility
- to write a script or a C program to convert the ASCII image into a C header file

This process is described, for example, in [1].

This method has some shortcomings: the HEX utility itself only supports a limited set of commonly used formats, and users often write additional utilities to reformat the HEX output or extract information about the object file, such as section length information. In addition, changes to the utility are required with each change to the object format. Hence, it would benefit DSP users to have a more robust and more flexible way to process object files, without having intimate knowledge of the object file formats.

Starting with Code Composer Studio version 3.0, a new tool is available which makes it easier to generate the above mentioned DSP image, and, in general, makes it easier to perform any custom post-processing of a COFF file. This tool is called object file display (OFD) utility. It can be seen as an API into TI's object file format, (i.e. it provides all relevant information about the object file and eliminates the need for the user to understand the details of the object format).

For developers who use Code Generation Tools (CGT) such as OFD outside of Code Composer Studio, OFD is introduced in the releases listed in Table 1.

Table 1. OFD Availability

Device Family	CGT Version
C6000	5.00 or greater
TMS470 (ARM)	3.00 or greater
C5500	3.00 or greater
C5400	4.00 or greater
C2800	4.00 or greater

The OFD utility takes COFF file as the input, and generates an output in XML format. This output can be customized and processed further to convert the COFF file into a desirable format, such as C source format; or extract specific information out of the COFF file, for example, the number of sections and their respective types and sizes, or object endianess. In order to extract information from an XML file, it is common to use already available XML parser modules, which parse an XML document and provides access to its data. XML parsers are available for various programming languages including Perl, Visual Basic, C++, and Java.

2 COFF Format

Since the main purpose of the OFD utility is to convert a COFF file, and most of the tags defined in the XML output have a corresponding entry in the COFF file, we first briefly recap some of the main elements of the COFF format. For more detailed information on COFF file structure, the user is referred to the relevant sections in [6]−[10].

The general structure of a COFF file is shown in Figure 1. The main components of a COFF file are sections. Each section has a header containing information on the type of the section, physical and virtual addresses, and its size. Each initialized section also has raw data associated with it. If the object is relocatable, relocation information and line number tables are also generated for each section. A section header contains pointers to raw data, relocation information, and the line number table. Examples of initialized sections are .text, .cinit, and .switch. Examples of uninitialized sections are .stack, .sysmem, .bss, and .far.

In addition to section-related contents, a COFF file has a file header and an optional file header. The file header and optional file header contains information on COFF version, the number of sections, the number of symbols, size and start address of certain sections, as well as entry point, endianess, relocation information, etc.

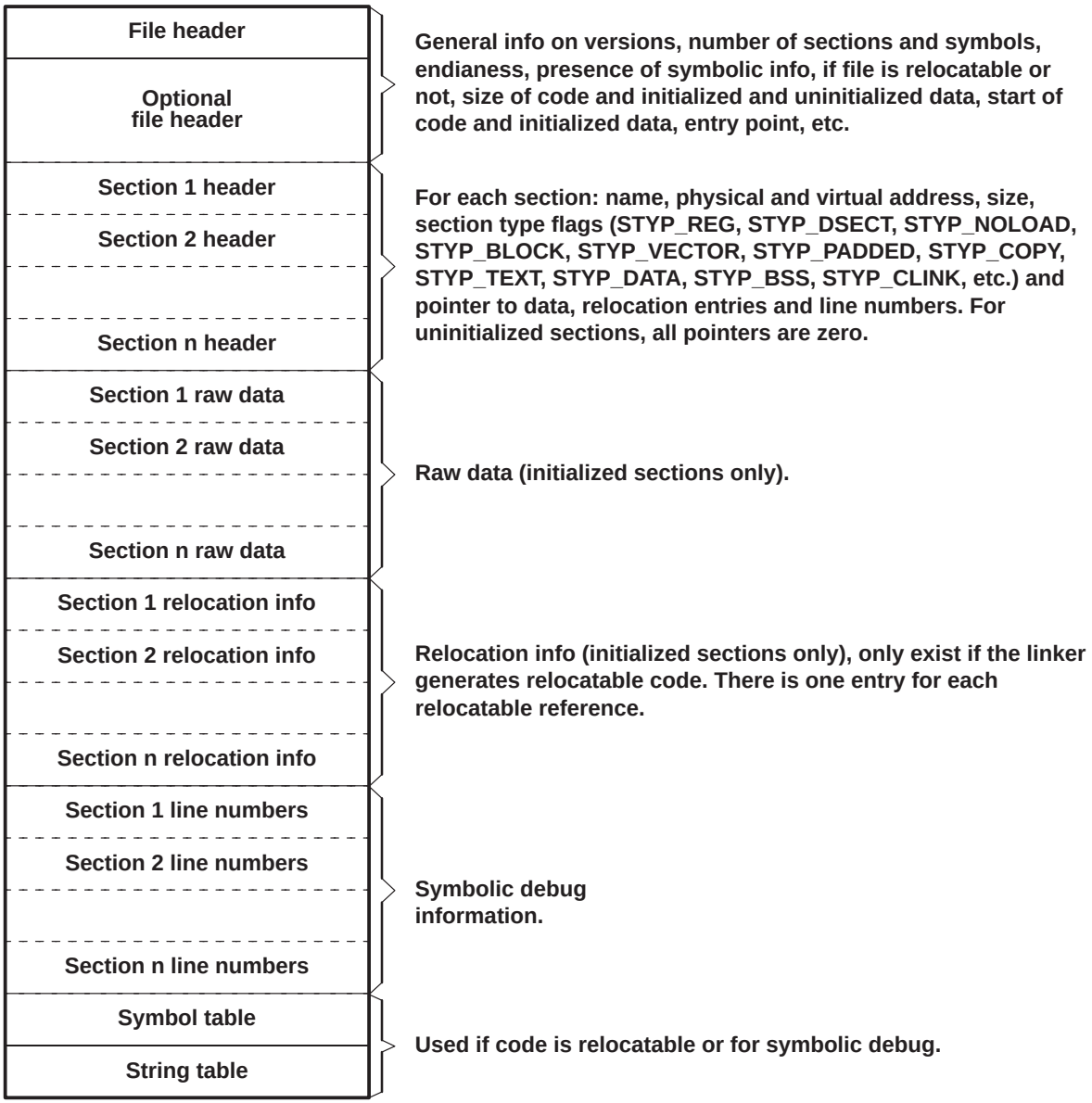

Figure 1. COFF File Structure

3 OFD Text Output

To get a better understanding of what the OFD utility does, you will first examine the human readable output format, which is obtained by invoking OFD without any options, with the object file name as the only argument. An abbreviated example is shown in Figure 2. Notice that the OFD utility translates the information contained in the COFF file structure described in the previous section into a format which is easily understandable by a human reader. It displays the basic information about the overall object file and each individual section, while eliminating the need to understand the specifics of the COFF format. Also, the OFD output does not contain the raw data, it only gives COFF file pointers to the raw data.

```
OBJECT FILE:  Debug\example1.out

 FILE HEADER RECORD INFORMATION :
 -------------------------------
   COFF Version : 0302 (octal) [COFF2]
   Target id    : 0231 (octal) [C60]
   Current file has 12 sections and  571 symbol definitions
   Object file flags = 0x11a3
      Relocation info stripped
      Executable (no unresolved refs)
      Tags, etc were merged - no duplicates
      CPU generation = 0xa0
      Little endian (object code is LSB first)
   File is NOT SWAPPED
   File length = 50580 bytes

 OPTIONAL HEADER INFORMATION :
 -----------------------------
   Magic = 0x108      Version stamp = 436
   Size of ".text" section = 0x8b60 bytes, starting at address >0x200
   Size of ".data" section = 0x0 bytes, starting at address >0x200
   Size of ".bss" section  = 0x0 bytes
   Entry point into module = 0x8b80

 DEFINITION OF SECTION "$BRID"
 -------------------------------
   Physical addr = 0x0, Virtual addr = 0x0, Size = 0x62c,
   Number of relocation entries = 0,  line number entries = 0
   Flags = 0x10,  Memory Width = 0 bits
  [ Alignment = 1 units, Type = COPY ]

   File pointers : Raw data at 0x272
                 : Relocation at 0x0
                 : Line # entries at 0x0

 DEFINITION OF SECTION ".text"
 -------------------------------
   Physical addr = 0x200, Virtual addr = 0x200, Size = 0x8b60,
   Number of relocation entries = 0,  line number entries = 0
   Flags = 0x520,  Memory Width = 0 bits
  [ Alignment = 32 units, Type = TEXT ]

   File pointers : Raw data at 0xa9e
                 : Relocation at 0x0
                 : Line # entries at 0x0
DEFINITION OF SECTION ".stack"
 -------------------------------

   Physical addr = 0x8d60, Virtual addr = 0x8d60, Size = 0x400,
   Number of relocation entries = 0,  line number entries = 0
   Flags = 0x380,  Memory Width = 0 bits
  [ Alignment = 8 units, Type = BSS ]

   File pointers : Raw data at 0x0
                 : Relocation at 0x0
                 : Line # entries at 0x0

 [...]
```

Figure 2. OFD Output (Text Format)

Although the interpretation of this text output is for the most part straight forward, the following points are interesting to note:

- For each initialized section (such as .text), the “Raw data” pointer is a pointer into the COFF file where the actual section data is found. This pointer is used when extracting the data from the COFF file. For uninitialized sections, such as .stack and .bss, the raw data pointer is 0x0.
- In the above example, there are no line number entries (the number is zero, and the pointer to them is also zero). The fact that there are no line number entries in the file indicate that the COFF file was generated without any symbolic debug options. If symbolic debug were turned on, for example, using –g option, line number information would appear.
- There are no relocation entries because, by default, the linker creates an absolute executable image. Such files do not require any relocation information.
- Virtual address does not have any significance in TI’s COFF files. This particular entry was inherited from the UNIX format where virtual addressing is supported.
- The COFF file, and therefore the OFD output, may contain compiler-internal sections, such as “$BRID” in Figure 2. Such sections are COPY sections that do not consume any target memory. Therefore, COPY sections are typically ignored.

Also, the OFD utility can be run on archives (.lib), not only .obj or .out files. For every .obj included in the archive, information similar to that given in Figure 2 is displayed.

4 OFD XML Output

The XML output format is generated when the OFD utility is invoked with −x option. The information contained in this output is the same as the information contained in OFD text output discussed in the previous section. It is in a format which is both industry-standard and easy to process using a wide variety of existing tools which will be discussed in the next section.

Additional information on XML can be found, for example, in [2], [3], [4], [5]. For the purpose of understanding and processing the OFD output, the user only needs to be familiar with the high-level structure of an XML document, the specific set of tags generated by the OFD utility, and the tools used to extract the information associated with those tags.

An XML document has a tree-like structure where each entity is enclosed by start tags and end tags, and all entities are nested within one another. The tree structure of the OFD output along with the tags it generates are shown in Figure 3. Note that, not all of the tags are shown, for example, relocation and symbolic information is not expanded in the given view. For an exhaustive list of tags generated by the utility, the reader is referred to the relevant sections in [6]−[10].

It can be seen from Figure 3 that the `<ofd>` tag denotes the root of the XML document. The information contained in COFF headers corresponds `to ofd->object_file->ti_coff->file_header` and `ofd->object_file->ti_coff->optional_file_header` entities, respectively. This information (for example, the total file length, or the total number of sections or symbols), could be used to further process the COFF file. A simple example is that QualiTI, the XDAIS Algorithm Compliance Tool, could read the `ofd->object_file->ti_coff->file_header->endian` entry to confirm that a submitted third-party algorithm is indeed little-endian.

Each section header in a COFF file corresponds to an `ofd->object_file->ti_coff->section` entity in the XML file. The number of such entities is equal to the number of sections in a COFF file, which can be retrieved from the `ti_coff->file_header->section_count` entity.

For the purpose of building a boot image, it is necessary to determine, for each initialized section, the length and the location of the raw data. This information is contained in the `section->raw_data_size` and `section->file_offsets->raw_data_ptr` elements, respectively. To determine if a section is initialized or not, use the tags which correspond to the section header flags. For example, if tags `<text>` or `<data>` are defined, the section is initialized. On the other hand, if tags `<bss>`, `<copy>`, `<dummy>` or `<noload>` are defined, the section is not initialized.

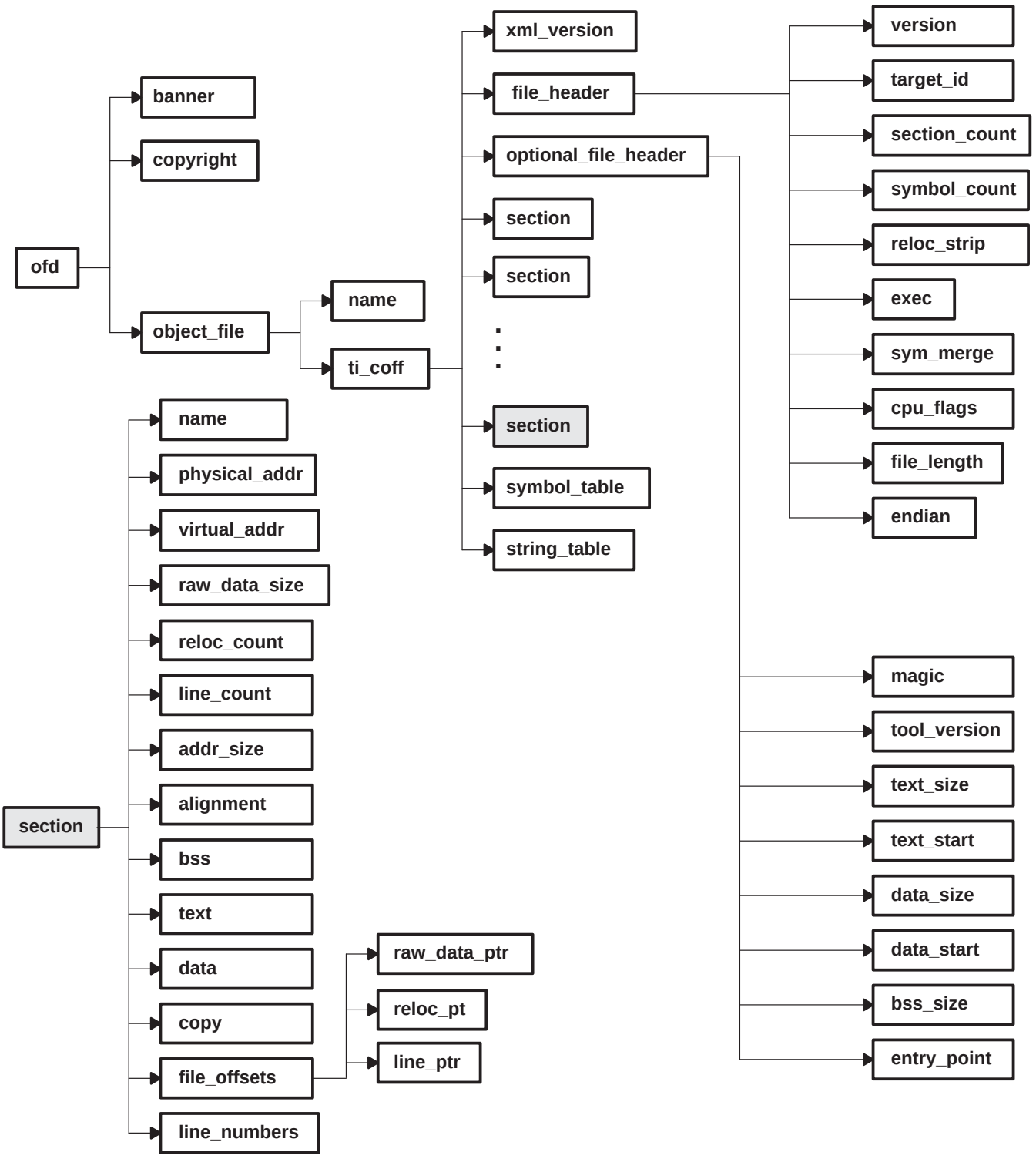

Figure 3. Tree Representation of XML Tags Generated by OFD

The tree structure can be easily visualized by displaying the XML output of OFD in a Web browser such as Internet Explorer. An example listing is shown in Figure 4. Note that “+” and “–“ signs denote collapsed and expanded contexts, respectively.

```
<?xml version="1.0" ?>
- <ofd>
    <banner>TMS320C6x Object File Display PC v5.0.0</banner>
    <copyright>Copyright (c) 1996-2004 Texas Instruments Incorporated</copyright>
  - <object_file>
      <name>Debug\example1.out</name>
    - <ti_coff>
        <xml_version>0x2</xml_version>
      - <file_header>
          <version>0xc2</version>
          <target_id>0x99</target_id>
          <section_count>0xc</section_count>
          <symbol_count>0x0</symbol_count>
          <reloc_strip>true</reloc_strip>
          <exec>true</exec>
          <lnno_strip>true</lnno_strip>
          <localsym_strip>true</localsym_strip>
          <sym_merge>true</sym_merge>
          <cpu_flags>0xa0</cpu_flags>
          <file_length>0x9b3a</file_length>
          <endian>little</endian>
        </file_header>
      + <optional_file_header>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      + <section>
      </ti_coff>
    </object_file>
  </ofd>
```

Figure 4. Sample OFD XML Output

5 Tools for Processing OFD Output

The processing of OFD’s XML output starts with parsing the XML file. Since XML is basically a text file, it is possible to write a program from scratch to parse it, but this is typically not needed since XML parser modules are available for various programming environments. For example, MS Visual Studio .NET includes an XML parser for use with VB scripts and Visual C++ programs (available via msxml.dll). Also, ActiveState’s ActivePerl distribution includes a number of different XML parser modules (see for example [11]).

In this application note, we show how to work with the OFD output in the ActivePerl environment. We will assume that the reader is new to Perl scripting.

The attachment associated with this application note includes a sample Perl script (bootimage.pl) which processes an .out file by using its corresponding OFD XML output. In this section, we show how to run this script. In the next section, we explain the details of what the script actually does. The steps to run the Perl script are:

1. Download ActivePerl from ActiveState, see [12].
2. Install it and reboot your PC to get updated path information.
3. Unzip the zip file associated with this application note which contains a sample project.
4. Open the project in Code Composer Studio and build. The Perl script (bootimage.pl) runs as a “final build step”, see Project → Build Options → General. (Note: the OFD utility is called from the Perl script, so it is not necessary to run it before running the script).
5. The script generates app.out.c and app.out.h files to be included in host application.

In the next section, we explain in detail what bootimage.pl does. For absolute novices to Perl, it may be useful to go through a Perl tutorial, for example, [13], or refer to [14].

6 Creating a Host Image

In a system where a microprocessor boots the DSP (via HPI or PCI ports on the DSP), the DSP image needs to be included as a part of the host application. In addition to the actual HEX data which is contained in a COFF file, the host needs information about the destination addresses and length of different sections, all of which can be obtained from the XML file.

In our example, the boot image is in the form of a C header file (filename.out.h) and an associated source file (filename.out.c). The header file contains extern array declarations for each initialized section, as shown for example in Figure 5.

```
extern const unsigned char _vectors[0x200];
extern const unsigned char _const[0x138];
extern const unsigned char _text[0x8c00];
extern const unsigned char _cinit[0x35c];
```

Figure 5. Boot image .h file

The source file contains the actual HEX data for each section (one C array for each section), as well as, the physical and the virtual addresses for that section. An (abbreviated) example is shown in Figure 6.

```
/*******************************************************
** _vectors[0x200]: paddr = 0x00000000 vaddr = 0x00000000
*******************************************************/
const unsigned char _vectors[0x200] = {
0x2a, 0x60, 0x46, 0x00, 0x6a, 0x00, 0x00, 0x00, 0x62, 0x03, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
...
}

/*******************************************************
** _text[0x8c00]: paddr = 0x00000200 vaddr = 0x80000000
*******************************************************/
const unsigned char _text[0x8c00] = {
0xf1, 0x18, 0xbc, 0x0f, 0xf4, 0xd4, 0x3d, 0x06, 0x45, 0x61, 0x7c, 0x05, 0xa0,
0x06, 0x10, 0x05, 0x64, 0x02, 0xa8, 0x03, 0x00, 0x00, 0x00, 0x00, 0xf6, 0x42,
0x60, 0x8d, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x60,
0xa5, 0x00, 0x00, 0xd8, 0x97, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
...
}

/*******************************************************
** _cinit[0x35c]: paddr = 0x00009220 vaddr = 0x00009220
*******************************************************/
const unsigned char _cinit[0x35c] = {
0x30, 0x02, 0x00, 0x00, 0xd8, 0x9b, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
....
}
```

Figure 6. Boot image .c file

The .c and .h files are generated using the attached Perl script bootimage.pl. The script does the following:

1. Takes as the input argument an existing .out file.
2. Runs the .out file through OFD with –x option (to generate the XML file)
3. Parses the XML file using `XML::Simple` module. This operation returns a data structure (nested hashes and arrays) consisting of nodes which represent various document components (elements, text contents, etc). For example, the following code snippet is used to get access to all “section” entities:

```
my $xml      = ‘ofd6x –x $filename‘;
my $config   = XMLin($xml);
my $sections = $config->{’object_file’}->{’ti_coff’}->{’section’};
```

4. Opens destination files (e.g. filename.out.c and filename.out.h)
5. For all section entities within the XML file:
   a. Gets the size of the section (`section->raw_data_size`)
   b. Gets the file pointer inside the .out file section (`section->file_offsets->raw_data_ptr`)
   c. If the section should be copied (i.e. if its size is greater than zero, if it is in the .out file, and if it is not a BSS, COPY, DUMMY or NOLOAD type section):

- Find the appropriate position in the .out file using the above obtained pointer and a `seek()` function
- Generate a declaration in the header file (.h)
- For the entire length of the section

  read a byte from the .out file and write it into a C file.

The script, as is, is generic and may not handle various special cases that can be encountered in a COFF file. Therefore, it should be thought of as a baseline which can be modified to fit specific needs.

The part which will be most likely modified is step (5c) above, (i.e. deciding if the section needs to be copied or not). One special case would be if –fill command is used in the linker .command file on certain uninitialized data sections (for example, a DSP/BIOS application fills the .stack section with value 0x00c0ffee). Such sections may not need to be be part of the host image and the script can be modified to exclude them.

7 Summary and Conclusion

This application note shows how to create a DSP boot image directly from a .out file, using the OFD utility and a simple Perl script. This approach eliminates the need to use the HEX utility and a custom C program which reformats the HEX utility output. Thus, the development flow becomes simpler, more robust and more flexible. For example, the user can easily include or exclude certain sections, or analyze their properties (for example, size or address). A Perl script which generates the boot image is explained in the application note and provided with the associated code. The script can be easily modified to accommodate various application specific needs related to processing individual sections.

This is only one of the many possible uses of the OFD utility. The main purpose of this utility is to provide information about a COFF file, without exposing the COFF format itself. The information is provided in the XML format, which is both easy to understand and easy to process using already available libraries such as Perl XML or MSXML. As such, this utility enables custom post-processing of COFF files.

8 References

1. *TMS320C6000 HPI Boot Operation* (SPRA512)
2. http://www.xml.org
3. *XML tutorial*, http://www.w3schools.com/xml/default.asp
4. *XML: The ACSII of the Future?* http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarxml/html/xmlfinal.asp
5. *TMS320C6000 Assembly Language Tools User's Guide* (SPRU186)
6. *TMS320C55x Asembly Language Tools User's Guide* (SPRU280)
7. *TMS470R1x Assembly Language Tools User's Guide* (SPNU118)
8. *TMS320C54x Assembly Language Tools User's Guide* (SPRU102)
9. *TMS320C28x Assembly Language Tools User's Guide* (SPRU513)
10. *Perl-XML Frequently Asked Questions*, http://www.perl-xml.sourceforge.net/faq/
11. *ActivePerl distribution from ActiveState*, http://www.activestate.com
12. *Introductory Perl Tutorial Course for Windows*, http://www.gossland.com/course/
13. *Comprehensive Perl Archive Network*, http://www.cpan.org

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265